Automated reasoning

Gerard A.W. Vreeswijk
Utrecht University

Version January 22, 2010

Contents

Chapter 1

Introduction

Automated reasoning is an immense area with many side-paths. It is also a difficult area—some even claim that it is extremely difficult.[1] The following is a modest overview, in which we briefly describe its history and development and mention a number of successful applications.

This introduction attempts to place the forthcoming chapters in some perspective. It is merely meant has a warm-up and does not aspire historical or territorial completeness.

1.1 Some history

Reasoning exists as long as mankind, and ancient logicians already made allusions to mechanizing it. Therefore the desire to mechanize reasoning is very old.

Figure 1.1: Gottfried Wilhelm Leibniz (1646-1716).

The first time when the ideas of calculation and reasoning were concretely merged and articulated was probably by the German philosopher, physician and mathematician Gottfried Wilhelm Leibniz (1646-1716). In 1666, Leibniz sketched in his dissertation *De arte combinatoria* a picture of a so-called *calculus ratiocinator*: a system of more or less algorithmic instructions that should make it possible to derive new knowledge from existing facts in a purely mechanical manner. Leibniz maintained that such a calculus was only possible on the basis of a *lingua philosophica* or *characteristica universalis*: a universal language that would reflect the structure of the material world adequately. Around 1672 Leibniz improved Pascal's calculator, the *Pascaline* by creating a machine that could also multiply. Like its predecessor, Leibniz's mechanical multiplier worked by a system of gears and dials. Although Leibniz began so see, throughout the years, that a realization of his ambitious project involved all sorts of problems and might even be infeasible, the ideal of a calculus ratiocinator seemed to have never left him. More than eighteen years after his dissertation, he claims with a certain amount of pride that

[1] "Despite some impressive individual achievements, the extreme difficulty of Automated Theorem Proving (ATP) means that progress in ATP is slow relative to, e.g., some aspects of commercial information technology." [111].

highly-levelled philosophical discussions and controversies could in principle be decided by simply calculating the right answer.[2] In [50], we can read that Leibniz on his turn was inspired by work of Raymond Lullus (1235-1315) and Thomas Hobbes (1588-1679). According to Hobbes, reasoning is actually nothing more than computing with words. In this respect, the title of Hobbes' logic, *Logica sive Computatio!* is illustrative and significant [69].

Sadly, nothing other than vague generalities about Leibniz' goals for logic were published until 1903–well after symbolic logic was in full blossom. Thus one could say that, great though Leibniz' discoveries were, they were virtually without influence in the history of automated reasoning.

Apart from Leibniz' ideas, the history of automated reasoning may be put on a par with the history of formal logic until the rise of the computer after 1945, simply because there were no computers before World-War II

[50, 65, 61]. Soon after the birth of the first-generation computers like the ENIAC, the EDVAC and the UNIVAC (1944-1950), automated deduction (AD), or automated theorem proving (ATP), became a dedicated research area.[3] Early important work includes the (classic) *Davis-Putnam procedure*[4] and the resolution proof calculus proposed by Robinson in 1965 [96]. See Chapter 5 for a historical footnote on resolution.

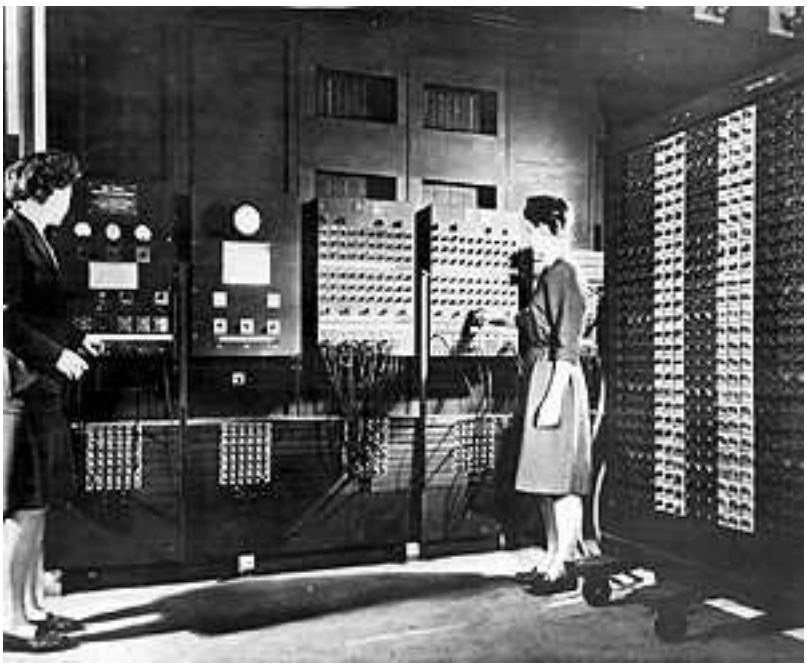


Figure 1.2: The ENIAC (1646-1716).

After Robinson's seminal paper in 1965 numerous improvements and refinements to resolution were proposed. Larry Wos and his fellow researchers at Argonne National Laboratory began to set the ATP-scene [130, 129, 83]. Over the following seven years, two major systems were constructed, viz. P1 (for "Program 1") and RW1 (for "Robinson-Wos 1"). These systems supplied the tools for the very first experiments with resolution, and set the tradition of exploring the applicability of theorem proving to problems in mathematics and logic. At this point, it is important to mention that, during the years, Wos and his co-workers more or less claimed the term "automated reasoning," for what is essentially automated theorem proving (ATP). As a result, many books on ATP have "automated reasoning" in their title, which you have to be aware of when scanning your local library.
The popular theorem prover OTTER is currently the most prominent offspring of the Wos research school in ATP. OTTER is currently being used by mathematicians, and has been used in ways only indirectly related to mathematical theorem proving, such as circuit design and verification. For a historical overview of Argonne-style theorem proving through the years, see [60]. Later on, other methods than resolution came into the centre of interest, in particular

[2] According to Gerhardt, this particular statement of Leibniz must be dated after 1684 [32].

[3] For a historical and sociological view of AD, see [61]. For an overview of the first twenty-five years of AD see [58]; see [102] for a comprehensive collection of the "classic" papers of 1957-1970.

[4] Not be confused with the currently far more popular variant of Davis, Longemann and Loveland [23].

"model-finding"-approaches such as semantic tableaux, and stochastic methods.

From approximately 1960 on, interest began to rise in automated forms of *non*-deductive reasoning, also known as *intelligent reasoning*, or *commonsense reasoning*. In a spirit of time that was influenced by electronic chess and smart combinatorial algorithms, it was assumed that intelligence is essentially a form of symbol manipulation. This scientific paradigm later became known as *symbol system hypothesis*, or "symbolicism," for short. Today, the supposition that intelligence could, or even *should*, be defined in symbolic terms is somewhat ironically termed "good old fashioned artificial intelligence" (GOFAI). One of the first GOFAI-attempts to mechanize commonsense reasoning was the so-called General-purpose Problem Solver (GPS). GPS was a procedure developed by Newell and Simon (1973) and was meant as a basis for intelligent computers that could reason with domain-independent knowledge. The project was deemed unsuccessful, because it turned out to be more difficult then expected to draw specific conclusions on the basis of general domain-independent information. As a result, so-called *expert systems* were introduced (1960-1970). Expert systems are computer programs that are particularly good at reasoning within a small and restricted domain (such as specific forms of medical diagnosis, classification or planning), just like experts are used to do. Later, expert systems were renamed into *knowledge-based systems*, because of the insight that domain-knowledge, rather than the experts that provide it, forms the heart of expert systems. Also other sorts of intelligent systems came into existence, such as industrial robots, and intelligent assistants. Work on intelligent systems has been important all along, because it creates a need for new and (more) sophisticated sorts of automated reasoning (Section 1.2).

Recent developments in automated reasoning include forms of connectionistic reasoning, holistic theories of rationality and reasoning with knowledge that is latent on the internet. An example of a holistic theory of rationality is Thagard's theory of *explanatory coherence*, in which an initially informal notion of (in)coherence is made mathematically precise and implemented on the computer (Chapter 12 on page 233). Recent developments on reasoning with knowledge on the internet indicate that it is possible to draw conclusions regarding the semantic content of web-pages on the basis of their underlying hyperlink structure (the so-called "semantic web"). Further, recent developments in XML have shown how semantically-typed markup languages may establish logical relations among web-pages, so that automated reasoners can reason about the ontology or classification of pages, detect outdated, inconsistent and incomplete information in web-pages, or discover social networks of real-world communities on the internet.

1.2 Taxonomy

Automated reasoning can be divided in two areas, viz. automated deduction (abbreviated as AD, or ATP), and non-deductive forms of automated reasoning. Here is an (incomplete) taxonomy:

1. *Automated deduction.* The challenge of automated deduction is to prove (crunch) hard theorems with smart algorithms ("theorem crunchers"). Goal theorems are taken from, e.g, algebra, combinatorics, logic, and number theory. ATP is a large and active area with a long history. The annual conference on automated deduction (CADE) brings together researchers who discuss progression in AD/ATP, present their systems, and compare the performance of their theorem proving software.

   Somewhat paradoxically, much of ATP is not theorem proving. (This is further explained in Section 2.1 on page 25.) Instead, just about all of ATP is based on the idea of finding a countermodel, that is, finding a truth-assignment to the variables under which the formula in question comes out false. If a countermodel has been found, the theorem is invalid. If no countermodel has been found, and the search was exhaustive, then the theorem is valid. Accordingly, ATP falls apart in roughly two areas, viz. *symbolic* approaches, such as semantic tableaux, resolution, and binary decision diagrams, and *connectionistic* approaches, such as stochastic local search and simulated annealing. The working

hypothesis of symbolic approaches is that a symbolic representation of the things we are reasoning about is an essential prerequisite for reasoning. The working hypothesis of connectionistic, or sub-symbolic, approaches on the other hand is that it is the computer's task to represent the data and that it is the computer's task to discover the interdependencies between the various data elements. In particular, connectionistic approaches depart from the assumption that knowledge need not be represented in human-readable format to come to sensible conclusions.

(a) *Symbolic approaches.*

i. *Semantic tableaux.* The idea of semantic tableaux is to try to disprove a formula by recursively analyzing its subformulas. Chapter 2 (page 25) is completely dedicated to semantic tableaux, which is probably the most elegant and transparent form of theorem proving.

ii. *Resolution.* Resolution does not analyze formulas, but rather *combines* them in all sorts of ways. First a formula is converted to a conjunctive normal form (CNF).[5] This CNF is represented as a so-called *clause set*. The resolution process consists of repeatedly adding clauses to the clause set, while other (redundant) clauses, are dropped. This process continues, until either the empty clause (corresponding to "false") is derived, or computational resources are expended. Currently, resolution is the most powerful form of theorem proving.

iii. *Binary decision diagrams.* BDDs are graphical representations of if-then-else normal forms (INFs), which are canonical representations of boolean expressions, like CNFs and DNFs (page 97). Procedurally speaking, a BDD is a binary decision tree over the boolean variables with node unification for identical subformulas. A BDD is a rooted, directed, acyclic graph (DAG) with an unconstrained number of in-edges and two out-edges, one for each of the one and zero decision paths of any given variable. Binary decision diagrams are attractive to use due to their canonicity, simplicity, and the availability of mature manipulation tools.

(b) *Connectionistic approaches.* The idea here is to try to guess satisfying models by local search techniques, such as hill climbing. Probably the most promising form of theorem proving.

2. *Non-deductive forms of automated reasoning.* This area tries to automate practical forms of reasoning, such as reasoning with incomplete, uncertain, or inconsistent information.

(a) *Symbolic (or qualitative) approaches* depart from the logical symbol hypothesis, which says that reasoning is a process of symbol manipulation. It doesn't involve numbers in the first place.

i. *Argumentation-based approaches.* Defeasible reasoning, dialogue games.

ii. *Qualitative probabilistic networks.* Symbolic abstraction from Bayesian belief networks. Probabilities are replaced by signs, e.g. ($-$, 0 and $+$), or ($--$, $-$, 0, $+$ and $++$).

iii. *Non-monotonic reasoning.* Tries to model the type of reasoning conclusions may be drawn until further evidence proves otherwise (this is the non-monotonicity.). Based on classical logic. Threatened with extinction.

(b) *Numeric (or quantitative) approaches.* The "soft" department of non-deductive reasoning. Departs from the assumption that reasoning is not a yes-or-no matter, but a gradual process with different nuances.

i. *Probabilistic or probabilistically-oriented approaches.*

[5] However, it is known that such a conversion cannot be done in linear time.

A. *Bayesian belief networks, a.k.a. probabilistic networks.* Currently the most popular approach and scientifically accepted approach to reason with uncertain information. It is unfortunately one of the most demanding approaches, in the sense that many probabilities must be known in advance before a probabilistic network can be of any value.

B. *Dempster-Shafer belief functions.* Subtle, but computationally intractable.

C. *Certainty factors.* Outdated.

ii. *Fuzzy logics.*

iii. *Possibilistic approaches.*

(c) *Connectionistic and/or holistic approaches.*

i. *Inference systems based on coherence.* The idea of coherence is that mutually reinforcing data should be given more credibility than contradictory information. Promising approach.

(d) *Hybrid approaches.*

3. *Application-oriented forms of automated reasoning.*

(a) *Abduction, explanation, and causal reasoning.*

i. *Classical / hypothetico-deductive forms of explanation.*

ii. *Abductive logic programming.*

iii. *Argument-based forms of explanation.*

(b) *Inductive reasoning.*

i. *Inductive logic programming.*

ii. *Learning new rules.*

4. *Applications / implemented systems.*

(a) *AR systems for research.*

i. *Theorem provers.*

ii. *Other experimental systems.*

(b) *Industrial systems.*

i. *Knowledge-based systems.*

ii. *Hardware-verification.*

Evidently, this taxonomy does no justice to reality. Many sub-areas overlap. For example, connectionistic approaches such as fuzzy SAT, or GSAT, are applied to tackle discrete satisfiability problems, such as "ordinary" (i.e., $\{0, 1\}$-valued) satisfiability. Conversely, qualitative approaches to reasoning with uncertain information, such as formal argumentation theory and qualitative probabilistic networks, are applied to tackle problems that are essentially of continuous type.

Further, the taxonomy isn't really a taxonomy, i.e, the taxonomy isn't hierarchically ordered, as the text suggests, because a number of activities "combine in different dimensions". For example, explanatory reasoning (Chapter 10) can be done deductively but it can also be done les strict, i.e., non-deductively. Thus, rather than searching for a proof that explains everything once and for all, we might settle for less and search for a plausible and convincing argument that may equally well serve as a satisfactory explanation. All this is up to the people who design and (more importantly) use such systems.

Two purposes for reasoning

The above taxonomy subsumes different types of reasoning. But there is yet another dimension, since different types of reasoning may serve different purposes. There are roughly two important purposes for reasoning.
- The first purpose is mere "belief hunger," viz. the desire to extend or broaden one's knowledge with the sole purpose to know as much as possible. Scientists are a good example of this. The purpose to broaden one's knowledge is served by the type of reasoning that is known as *epistemic reasoning*. Epistemic reasoning is concerned with the question what should be believed and what not.
- A second purpose for reasoning is to seek support for (practical) decisions. This is called *practical reasoning*. Thus, practical reasoning is concerned with the question what should be done and what not. A difference between practical and epistemic reasoning is that practical reasoning is action-oriented while epistemic reasoning is belief-oriented. Further, practical reasoning is subsumed by epistemic reasoning since all rational decisions are, by definition, based on knowledge. A famous example in which epistemic reasoning is of no use is *Buridan's ass*, who starved to death because the poor soul couldn't decide from which bale of hay to eat first [85]. An example of a situation in which even practical reasoning is of no use, is when your hand touches a heat source accidentally. In that case, your reflexes probably intervene before you can even *think* about what to do next.

The upshot of the above analysis is that all forms of reasoning mentioned in the above taxonomy can be used to support practical decision making processes. *Decision-support systems*, for instance, are a special type of knowledge-based systems, where practical reasoning supports decision making processes. In the area of artificial intelligence (AI) and cognitive robotics, practical reasoning is known as *reasoning about action*. But automated reasoning has (far) more applications:

1.3 Applications

Applications of ATP can be found in various domains. There are successes of automated theorem provers (ATPs) in planning, where ATPs perform better than specialized planning systems. In model-based diagnosis, tableau-based proof procedures are used to compute explanations of faulty circuits. In program verification, ATPs are embedded to achieve a higher degree of automation. In systems to support software reuse, ATPs are used to identify modules from a library to match a given specification. In natural language understanding ATP techniques are used to integrate world knowledge into the interpretation process. Also work on the (computationally intractable) propositional satisfiability problem of direct importance to other problems that are computationally intractable.

An important application area of automated reasoning is *mathematical theorem proving*. In 1976, for example, two German researchers Appel and Haken proved the Four Color Theorem by means of automated reasoning techniques. Appel and Haken's proof cannot be verified by hand, and even the part that is supposedly hand-checkable is extraordinarily complicated and tedious, and as far as I know, no one has verified it in its entirety. Checking the computer part would not only require a lot of programming, but also entering 1476 graph-descriptions into the computer.

In 1996, McCune *et al.* solved the Robbins problem—are all Robbins algebras Boolean? We now know that every Robbins algebra is Boolean, but for a long time this was unknown. This theorem was proved automatically by EQP, a theorem proving program developed at Argonne National Laboratory.

Another important application area of automated reasoning is planning and scheduling. The last decade, research in ATP has had considerable succes in solving planning problems by encoding them into propositional logic and then using both complete procedures like DPP (Chapter 4)

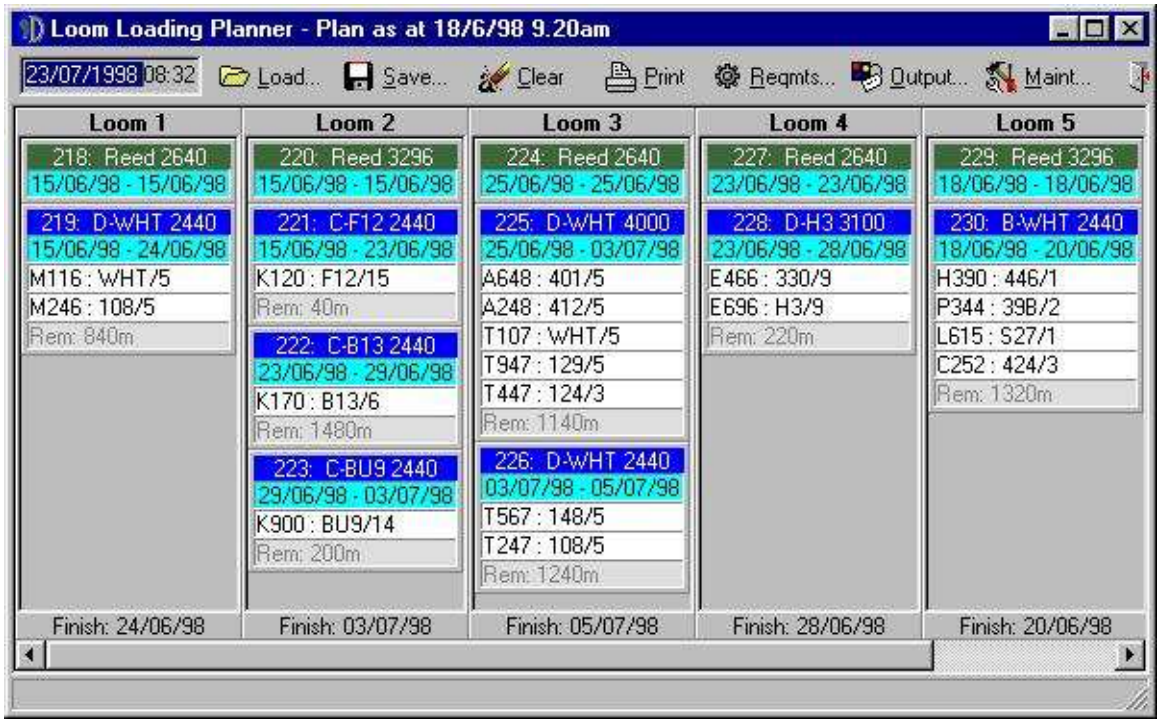


Figure 1.3: Screen of Loom Loading Scheduling Tool v3.1.3 (Orbital Decisions).

and local search methods like GSAT (Chapter 6).
Machine-shop scheduling problems are also successfully tackled by ATP-techniques. A machine-shop scheduling problem consists of scheduling a set of jobs subject to a set of resource constraints. start and due dates and sequencing constraints. By now a respectable number of scheduling benchmark problems have been solved by encoding them in monadic predicate logic and solving them with DPP or iterative sampling [12, 18].

An example of practical scheduling software is Loom Loading, which is is used at a textile factory to configure various loom loading scenarios, comparing their requirements and outputs to arrive at the optimal solution for achieving the production targets with the available raw materials in an efficient manner (eg: minimizing the number of reed changes), cf. Fig. 1.3. The rules built into the program and database try to optimize the use of resources and attempt to prevent costly errors.

Other important applications of automated reasoning in the area of knowledge-based systems include hardware and software verification, deductive databases, railway interlocking, weather forecasting, tide monitoring, configuration and planning, specification, design and implementation of scheduling systems, legal reasoning, car diagnosis, validation of knowledge bases, and the development expert database systems.

Applications of non-deductive reasoning

Problems that can be solved by automated reasoning techniques are typically questions that expose an interest in the consequences of being in some (possibly hypothetical) situation, maintaining a particular state, or advocating some particular an opinion. Since such forms of reasoning are (or at least pretend to be) more akin to practical reasoning, it should come as no surprise that non-deductive forms of automated reasoning give occasion even more applications

in the practical domain.

- Argumentation has been used as a component of negotiation protocols, where arguments for an offer should persuade the other party to accept the offer [51, 74]. Argumentation is also part of some recent formal models and computer systems for dispute mediation [37, 36, 10] and intelligent tutoring [122]. In such applications the dialectical proof theory can be used as a protocol for dispute: it checks whether the users' moves are legal, and it determines given only the arguments constructed by the users, which of the participants in a dispute is winning.
- Abductive and explanatory reasoning is mostly used in diagnostic systems. Examples are causal disease models, and physiological models of regulatory mechanisms in the human body. Model-based approaches have the potential to facilitate the development of knowledge-based systems, because they can be partially based on models described in the medical literature. Systems based on explanatory reasoning have the potential of explaining generated advice to users.

1.4 What *is* automated reasoning?

Above, we described how automated reasoning is put to practice, and what has been established so far. Let us now approach the matter from a purely analytical point of view and ask: what *is* automated reasoning?," i.e, what should it mean, and what are we allowed to expect from it? Thus, the question here is asked in a normative sense, rather than in a descriptive sense. We do not ask how automated reasoning is carried out in practice, but ask how it *should*, or *could*, be carried out, ideally. Accordingly, an appropriate answer would consist of a solid analysis of the subject, followed by an accurate definition. However, I do not give such a definition here. (I could say: "I leave it up to you, after you have read the notes," but maybe that would not be fair.)

A purely analytical answer to the question as to what automated reasoning is, would be that "automated reasoning" consists of two words, viz. "automated" and "reasoning," that must be analyzed further, before we can even think about defining the compound concept "automated reasoning".

1.4.1 What is reasoning?

Let us see what dictionaries have to say.

Collins COBUILD English language dictionary, developed and compiled in the English language Department at the University of Birmingham. Editor in chief John Sinclair. London, Glasgow, 1991:

reasoning is

1. The process of coming to a particular conclusion by thinking carefully about all the facts. E.g., "The facts don't matter, nor does the quality of his reasoning... What is the reasoning behind that decision? ... ...the reasoning powers of the mind.
2. The actual arguments, statements, and conclusions that you produce by this process. E.g., "I won't bother you with pages of tedious chemical reasoning...

The Oxford Dictionary of Philosophy, Simon Blackburn, 1994:

reasoning Any process of drawing a conclusion from set of premises may be called a process of reasoning. If the conclusion concerns what to do, the process is called practical reasoning, otherwise pure or theoretical reasoning.[6] Evidently such

[6]Some philosophers would qualify this as epistemic reasoning. *Epistemic reasoning* is reasoning with the purpose to find out what should be believed and what not. It is contrasted with *practical reasoning*, which is reasoning with the purpose to find out what should be done and what not.

processes may be good or bad: if they are good, the premises support or even entail the conclusion drawn; if they are bad, the premises offer no support to the conclusion. Formal logic studies the cases in which conclusions are validly drawn from premises. But little human reasoning is overtly of the forms logicians identify. Partly, we are concerned to draw conclusions that 'go beyond' our premises, in the way that conclusions of logically valid arguments do not (*see* abduction, induction). Partly, it has to be remembered that reasoning is a dynamic process, and that what to a logician looks like a static contradiction may be the sensible replacement of one set of assumptions with others as the process develops. Furthermore, if we reason we make use of an indefinite lore or commonsense-set of presumptions about what is likely or not (*see* frame problem, narrative competence). A task of an automated reasoning project is to mimic this casual use of knowledge of the way of the world in computer programs.

The American Heritage, 3rd Edition. Houghton Mifflin, 1992, 1996:

rea·son (n.)

1. The basis or motive for an action, a decision, or a conviction.
2. A declaration made to explain or justify an action, a decision, or a conviction: "inquired about her reason for leaving."
3. An underlying fact or cause that provides logical sense for a premise or an occurrence: "There is reason to believe that the accused did not commit this crime."
4. The capacity for logical, rational, and analytic thought; intelligence.
5. Good judgment; sound sense.
6. A normal mental state; sanity: "He has lost his reason."
7. (Logic.) A premise, usually the minor premise, of an argument.

rea·soned, rea·son·ing, rea·sons (v. intr.)

1. To use the faculty of reason; think logically.
2. To talk or argue logically and persuasively.
3. (Obsolete.) To engage in conversation or discussion.

(v. tr.)

1. To determine or conclude by logical thinking: "reasoned out a solution to the problem."
2. To persuade or dissuade (someone) with reasons.

Idioms:

"by reason of"	because of
"in reason"	with good sense or justification; reasonably
"within reason"	within the bounds of good sense or practicality
"with reason"	with good cause; justifiably.

[Middle English from Old French `raison`, from Latin `ratio`, from `ratus`, past participle of `reri` *to consider, think*; see ar- in Indo-European Roots.]

rea'son·er (n.) Synonyms: reason, intuition, understanding, judgment. These nouns refer to the intellectual faculty by means of which human beings seek or attain knowledge or truth.

Reason is the capability to think rationally and logically and to draw inferences:
- "the rationalist whose reason is not sufficient to teach him those limitations of the powers of conscious reason" (Friedrich August von Hayek).
- "Mere reason is insufficient to convince us of its [the Christian religion's] veracity" (David Hume).

Intuition is perception or comprehension, as of truths or facts, without the use of the rational process:
- "Because of their age-long training in human relations—for that is what feminine intuition really is—women have a special contribution to make to any group enterprise" (Margaret Mead).

Understanding is the faculty by which one understands, often together with the comprehension resulting from its exercise:
- "So long as the human heart is strong and the human reason weak, Royalty will

be strong because it appeals to diffused feeling, and Republics weak because they appeal to the understanding" (Walter Bagehot).
- "The greatest dangers to liberty lurk in insidious encroachment by men of zeal, well-meaning but without understanding" (Louis D. Brandeis).

Judgment is the ability to assess situations or circumstances and draw sound conclusions:
- "my salad days, // When I was green in judgment (Shakespeare)."
- "At twenty years of age, the will reigns; at thirty, the wit; and at forty, the judgment" (Benjamin Franklin). See also synonyms at **cause**, **mind**, **think**.

rea·son·ing (n.)

1. Use of reason, especially to form conclusions, inferences, or judgments.
2. Evidence or arguments used in thinking or argumentation.

Webster's Revised Unabridged Dictionary. MICRA, 1996, 1998:

reasoning \Rea"son*ing\, n.

1. The act or process of adducing a reason or reasons; manner of presenting one's reasons.
2. That which is offered in argument; proofs or reasons when arranged and developed; course of argument.

His reasoning was sufficiently profound. –Macaulay.

Syn: Argumentation; argument.

Usage: *Reasoning*, *Argumentation*. Few words are more interchanged than these; and yet, technically, there is a difference between them. Reasoning is the broader term, including both deduction and induction. Argumentation denotes simply the former, and descends from the whole to some included part; while reasoning embraces also the latter, and ascends from a part to a whole. See *Induction*. Reasoning is occupied with ideas and their relations; argumentation has to do with the forms of logic. A thesis is set down: you attack, I defend it; you insist, I prove; you distinguish, I destroy your distinctions; my replies balance or overturn your objections. Such is argumentation. It supposes that there are two sides, and that both agree to the same rules. Reasoning, on the other hand, is often a natural process, by which we form, from the general analogy of nature, or special presumptions in the case, conclusions which have greater or less degrees of force, and which may be strengthened or weakened by subsequent experience.

reason \rea"son\, v.t. [imp. & p.p. Reasoned; p.pr. & vb. n. reasoning]

1. To exercise the rational faculty; to deduce inferences from premises; to perform the process of deduction or of induction; to ratiocinate; to reach conclusions by a systematic comparison of facts.
2. Hence: To carry on a process of deduction or of induction, in order to convince or to confute; to formulate and set forth propositions and the inferences from them; to argue. Stand still, that I may reason with you, before the Lord, of all the righteous acts of the Lord. –1 Sam. xii. 7.
3. To converse; to compare opinions. –Shak.

WordNet 1.6. Princeton University, 1997.

reasoning adj: endowed with the capacity to reason [syn: intelligent, reasoning, thinking] n: thinking that is coherent and logical [syn: logical thinking, abstract thought]

If we see what the dictionaries have to say, a number of observations strike the eye:

i. Reasoning can be a process, as well as a rational reconstruction of that process.
ii. Reasoning can be used as a basis for action.
iii. The author of the Oxford Dictionary of Philosophy maintains that the task of an automated reasoning project is to mimic our casual and commonsense way of reasoning in computer

Gems

- Logic n. The art of thinking and reasoning in strict accordance with the limitations and incapacities of the human misunderstanding. AMBROSE BIERCE, *The Devil's Dictionary* (1881-1911).
- Man has such a predilection for systems and abstract deductions that he is ready to distort the truth intentionally, he is ready to deny the evidence of his senses only to justify his logic. DOSTOEVSKY, *Notes from Underground* (1864), 1.7, tr. Constance Garnett.
- Reason is also choice. MILTON, *Paradise Lost* (1667), 3.108.
- Rational thought is interpretation according to a scheme which we cannot escape. NIETZSCHE, "Notes" (1887), 522, in *The portable Nietzsche*, tr. Walter Kaufmann.
- The last function of reason is to recognize that there are an infinity of things which surpass it. PASCAL, *Pensées* (1670), 345.
- Reason? That dreary shed, that hutch for grubby schoolboys. THEODORE ROETHKE, "I Cry, Love! Love!" *The collected verse of Theodore Roethke* (1961).
- Reason deserves to be called a prophet; for in showing us the consequence of our actions in the present, does it not tell us what the future will be? SCHOPENHAUER, "Further Psychological Observations," *Parerga and Paralipomena* (1851), tr. T. Bailey Saunders.
- The supreme triumph of reason, the analytical (...) faculty, is to cast doubt upon its own validity. MIGUEL DE UNAMUNO, "The Rationalist Dissolution," *Tragic Sense of Life* (1913), tr. J.E. Crawford Flitch.
- Somebody has to have the last word. If not, every argument could be opposed by another and we'd never be done with it. ALBERT CAMUS, *The Fall* (1956).

Table 1.1: What philosophers have said about reasoning

programs (p. 14, these notes).

iv. Webster's puts argumentation on a par with deduction (p. 16, these notes). We will not do this. (Cf. Chapter 8 on page 149.)

v. What Webster's (and other texts, including philosophical texts) call induction, we call non-deductive reasoning, or *ampliative reasoning* (C.S. Peirce 1839-1914). Ampliative reasoning is the kind of reasoning in which more is concluded than what is contained in the premises, and hence amplify the scope of our beliefs. Non-deductive arguments and arguments to the best explanation are not deductively valid, but may yield credible conclusions.
In these notes, the term *induction* is reserved for the mode of reasoning in which one tries to compress a large number of particular observations into a small number of general principles. This theme is further explored in Chapter 11 on page 217. (Cf. also the triangle of Peirce, on page 195.)

vi. Most reasoning takes us to conclusions that go beyond our data, in ways that interest us (Blackburn, p. 14, these notes).

vii. Many dictionaries make mention of concepts such as argument, counterargument, argumentation, dialogue, dispute, attack and defense. Thus, reasoning takes often the form of a process in which an argument does not stand once and for all, but may be defeated by stronger counterarguments. Such processes are often subsumed under the header

argumentation. Argumentation is covered in Chapter 8. Explanatory reasoning is covered in Chapter 10.

viii. Conclusions may have more or less degrees of force, among others because they depend on reasons that may be strengthened or weakened by experience. (Webster's, p. 16, these notes). Thus, most practical arguments are based on reasons that, in turn, are based on practical experience. In Chapter 11 on page 217 it is described how rules may be learned from experience. In Chapter 8 on page 149 it is described how such reasons can be chained into arguments of varying conclusive force.

The various dictionary definitions are perhaps unsatisfactory because some of them are vague, while others are imprecise, circular, controversial, or seemingly false. They also do not seem to converge to one uniform concept of reasoning.

What the definitions *do* show, however, is that reasoning is more than theorem proving. Indeed, the above definitions show that reasoning ranges from plain deduction to commonsense reasoning, from strict mathematical reasoning to intuitive, imprecise, speculative, heuristic, and sometimes even holistic symbol-manipulating processes. Thus, if reasoning is more than theorem proving, then automated reasoning should be more than trying to prove theorems with computers. Accordingly, the basic principle of the present notes is that automated reasoning should try to reflect the wide diversity of different kinds reasoning that occur in practice. Still, the present collection of notes does not attempt to cover all topics. Apart from the fact that this would be impossible, there exists many splendid introductory texts for almost every mainstream topic on automated reasoning one can think of. Instead, the approach I take here is to highlight a number of topics in the area of automated reasoning, that I consider as particularly significant and/or practically useful.

1.4.2 Formalization of reasoning

What makes automated reasoning with computers different from reasoning the way we humans do? An important supposition in automated reasoning is that patterns of reasoning possess a formal structure. Consider, for instance, the following quotation (*A Study in Scarlet*, Sir Arthur Conan Doyle):

> "... You appeared to be surprised when I told you, on our first meeting, that you had come from Afghanistan."
> "You were told, no doubt."
> "Nothing of the sort. I *knew* you came from Afghanistan. From long habit the train of thought ran so swiftly through my mind that I arrived at the conclusion without being conscious of intermediate steps. There were such steps, however. The train of reasoning ran. 'Here is a gentleman of a medical type, but with the air of a military man. Clearly an army doctor, then. He has just come from the tropics, for his face is dark, and that is not the natural tint of his skin, for his wrists are fair. He has undergone hardship and sickness, as his haggard face says clearly. His left arm has been injured. He holds it in a stiff and unnatural manner. Where in the tropics could an English army doctor have seen much hardship and got his arm wounded? Clearly in Afghanistan.' The whole train of thought did not occupy a second. I then remarked that you came from Afghanistan, and you were astonished."
> "It is simple enough as you explain it," I said, smiling.

Some would say that such a piece of reasoning possesses sufficient structure to be formalized into something that is displayed in Figure 1.4 on the facing page.

A second supposition in automated reasoning is that it is the inferences between the propositions, the *patterns of reasoning*, that matter, rather than the propositions themselves. Accordingly, propositions are replaced by letters, or symbols, under the assumption that the

$$\dfrac{\dfrac{\text{Here is a gentleman of a medical type}\quad \text{with the air of a military man}}{\text{an army doctor}}\ \text{Clearly}\ \dfrac{\text{his face is dark}\quad \dfrac{\text{his wrists are fair}}{\text{that is not the natural tint of his skin}}\ \text{for}}{\text{He has just come from the tropics}}\ \text{for}\ \dfrac{\text{his haggard face says clearly}}{\text{He has undergone hardship and sickness}}\ \text{as}\ \dfrac{\text{He holds it in a stiff and unnatural manner}}{\text{His left arm has been injured}}}{\text{[Where in the tropics could an English army doctor have seen much hardship and got his arm wounded?] Clearly in Afghanistan.}}$$

Figure 1.4: A "train of reasoning" (Sir Arthur Conan Doyle)

$$\dfrac{\dfrac{A \quad B}{C}\ \text{Clearly}\ \dfrac{D}{E}\ \text{for}\ \dfrac{\dfrac{F}{G}\ \text{for}}{H}\ \text{for}\ \dfrac{I}{J}\ \text{as}\ \dfrac{K}{L}}{M}$$

Figure 1.5: The same "train of reasoning," but now symbolized

meaning of the propositions involved is unimportant (Figure 1.5). The hypothesis that reasoning is a process that exists by the grace of a symbolic representation of propositions, is known as the *symbol system hypothesis*. Antagonists of the symbol system hypothesis maintain that reasoning is a synthetic process that emerges from the activity of millions of neurons, and that this process cannot be analyzed, or decomposed, into a formal and transparent structure.
Cf. [93, 110, 105, 24, 6].

1.5 Which problem do you want to solve?

One point of departure is to ask: "what is automated reasoning, which techniques do exist, and what can I do with them?" Although these are noble questions in their own right, they are also somewhat academic and theoretically oriented.

I think the real reason to read a book on automated reasoning (or any book whatsoever) is to say: "I have an interesting and important problem that I think can be solved by automated reasoning. Let's get a book on automated reasoning and see how it can help to solve my problem.". Thus, which reasoning technique you want to use depends on which problem you'd like to solve.

1.6 Programming

I assume you read this book because you are interested in building your own automated reasoner. (If not, then I presume you are least interested in the principles *behind* building an automated reasoner.) If you know which problem you'd like to solve, and if you know which technique you will use, and the ideas and concepts are clear, then the next question to face is the selection of a programming language.

Which programming language shall I use?

This question deserves ample consideration, because your choice must be based on substantial considerations—not on the last programming language you had in class, or on the trend of the day in the wonderful world of programming languages.

Prolog (Programming in logic) is usually seen as the most appropriate programming language to implement various types of reasoning. This is understandable, because Prolog is designed to deal with logical formulae in the first place. However, don't be fooled by Prolog's name. It would be a mistake to see Prolog as the only serious candidate to program in logic, i.e., to implement reasoning. I think that exact philosophers, logicians, and researchers in artificial intelligence are overly committed to Prolog as an automated reasoning language. This commitment to Prolog is unnecessary. Other programming languages, including functional programming languages (Lisp, Haskell, Gofer), and general purpose [scripting] languages (C, Java, Perl, awk) do the job just as well, and often better. There is even a point *against* using dedicated languages such as Prolog, since such languages are strongly committed to the syntax and semantics of first-order predicate logic which, after all, is just one out of the many types of reasoning.

- If your choice is Prolog, I would recommend downloading a Prolog interpreter somewhere. If you are not familiar with Prolog, it is best to start playing with the introductory examples (or demo's) that are usually delivered along with the Prolog interpreter. Buying a Prolog book is not necessary, but convenient. I recommend [16] and [9] as practical introductions to Prolog.

- Perl is another language I'd like to recommend as a tool to implement ideas in reasoning. Originally, Perl was forwarded as a string and file-manipulation language for system administrators. But this characterization is out of date and does no justice to the power of Perl. Exact philosophers, logicians, and researchers in artificial intelligence may consider Perl as a general purpose language for rapid prototyping. It is also less logic-oriented than Prolog, which I consider as an advantage. Perl does not put you in the straight-jacket of first-order predicate logic when you try to implement deviant logics or modes of reasoning that are beyond the expressive or inferential scope of dedicated languages such as Prolog. Perl is publically available. (Read the FAQ, section "What machines support Perl?" and "Where do I get it?".)

- Then there is Haskell. Haskell is a strongly typed functional language with a particularly clean semantics. (In fact, these properties would make it the opposite of Perl.) Haskell is a lazy functional language, which means that it defers the computation of sub-expressions as long as possible, and returns the results just in time. Haskell's lazy evaluation mechanism would make it an excellent choice to represent infinite lists of formulas, such as infinite Herbrand domains (page 57). Beware of choosing Haskell because small Haskell programs "look cute". Like any other programming language, large Haskell programs can get ugly just as well. Moreover, because Haskell is so ardent on eliminating side-effects, it is not the first language of choice to implement a trace facility in, although it is very well possible with built in and external libraries (cf. `Hat` and `nhc13`).

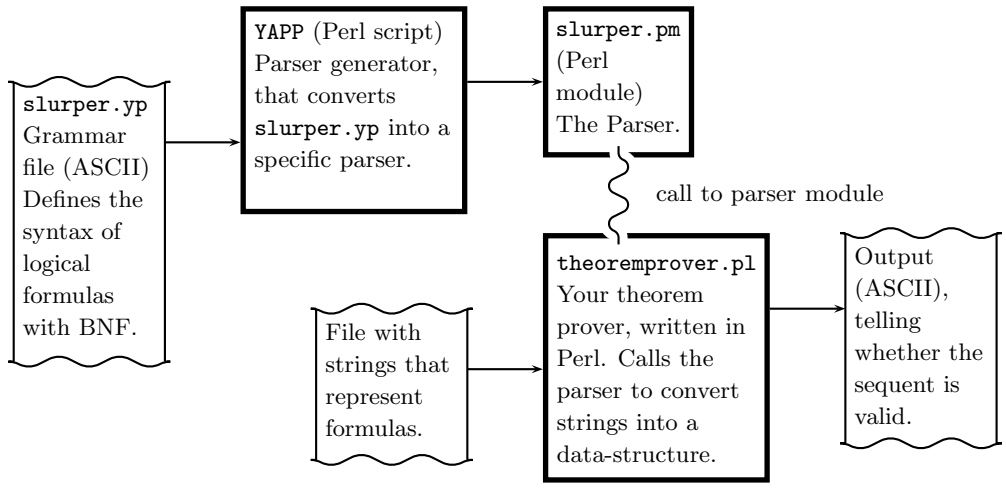


Figure 1.6: Construction of theorem prover in, e.g., Perl

- My personal pet is Ruby: a syntactically clean and fully object-oriented crossbreed of Perl and Smalltalk. In its home Japan more popular than Python, they say. Please check out the excellent "Programming Ruby" by Thomas *et al.* [117], also accessible online. The book is readable, accessible *and* covers Ruby almost completely.

The second problem is to make sure that the logical language of your problem domain can be read, or recognized, by your theorem proving program.

Parsing logical formulas

An important consequence of the fact that most types of logical formulas are constructed recursively, is that logical formulas are strings with parentheses that must match. Since regular expressions correspond to finite automata, and since the distance between a "(" and a ")" can be arbitrarily large, logical formulas cannot be recognized, or parsed, by means of regular expressions. This means that you will have to have a program that parses the logical formulas for you: a *parser*.

Getting the "parsing part" right can be a major hurdle in the process of writing your own automated reasoner. Here are some instructions how you deal with the situation.

Assuming that the logical syntax of your language of reasoning is determined, you may proceed as follows:

1. Choose a programming language.
2. Try to determine how logical formulas must be read. Perhaps it suffices if you read the input line by line, split lines in words, and proceed from there. Otherwise you will have to use a parser.
3. If you will have to use a parser, try to find a *parser generator* for your programming language. (If XYZ is the name of your programming language, an internet search on "parser generator for XYZ" probably gives what you want.)
   A parser generator is a program that reads a BNF grammar file, and outputs a module that is able to read and parse (i.e., "understand") logical formulas that are stated in the syntax defined. A parser is a program that converts flat strings from he input file into nested data-structures (often trees) that can be manipulated by your program.

```
# PropositionSlurper.yp

%left   '='       # definition of logical operator precedence
%left   '|' '>'   # (= equivalence, | disjunction, > implication,
%left   '&'       # & conjunction, and ~ negation; operators with
%left   '~'       # highest precedence first; %left means
%%                #  left-associative

                            # BELOW ARE SEMANTIC ACTIONS
input:   #  empty line       # input consists of 0 or more lines
         |  input line       { push(@{$_[1]},$_[2]); $_[1] }
;

line:       '\n'            { $_[1] } # line is a return, a proposition
         |  prop '\n'       { $_[1] } # followed by a return, or else
         |  error '\n'      { $_[0]->YYErrok } # the input contains
;                                               # an error

prop:       ATOM            # atom is recognized with the help of a
         |  prop '=' prop   { ['=', $_[1], $_[3]] } # regular expr.;
         |  prop '|' prop   { ['|', $_[1], $_[3]] } # notice how
         |  prop '>' prop   { ['>', $_[1], $_[3]] } # semantic actions
         |  prop '&' prop   { ['&', $_[1], $_[3]] } # create a tree
         |  '~' prop        { ['~', $_[2]] }        # structure
         |  '(' prop ')'    { $_[2] }
;
```

Figure 1.7: BNF grammar in YACC/YAPP for the language of propositional logic.

Nearly every language has a parser generator. For C it is YACC (yet another compiler-compiler) or Bison (GNU-YACC). For Perl it is YAPP (yet another Perl parser), and for Haskell it is HAPPY (don't know where the acronym stands for).
Most versions of Prolog support special grammar rule notation, better known as *definite clause grammars* (DCG). In that case, the parser runs on the standard Prolog search engine, and semantic actions can be realized by means of extra arguments in the predicates. This makes Prolog parsers belong to the class of so-called recursive descent parsers with backtracking. Recursive descent parsers are inherently slow and can't handle left-recursive grammar rules [82].

4. *Read the documentation of the parser generator.* It most likely describes, among others, the format of the grammar file, how to insert so-called *semantic actions* into the grammar, how to run the parser generator on the grammar file, and how to call the parser from within your own theorem proving module.

5. Write a grammar file. Figure 1.7 shows a grammar file in BNF, that has been used to define the syntax of propositional formulas.

6. You only have to run the parser generator once to let it write a parser module, unless you decide to change the grammar of the logical formulas. In that case, you will have to change the grammar file accordingly, and run YACC to generate a new parser. (Don't make changes in the parser directly, since the parser generator overwrites them.)

Enough preparation. Let us step into the area of automated deduction.

Part I

Automated deduction

Chapter 2

Propositional theorem proving

Remember your first course in logic? Remember fiddling with proofs of obscure formulas such as $p \supset (q \supset p)$? Such proofs are like doing a long division on elementary school: you must know how to do it, but once you did it you know it, and it becomes tedious. Here is the good news: propositional theorem proving can be automated straightforwardly (within 50 lines of code, say). In order not to spoil the euphoria, the bad news is deferred to Section 2.5 on page 37. (You may peek.)

Thus, there are programs that prove propositional theorems, as well as there are calculators that do long divisions.

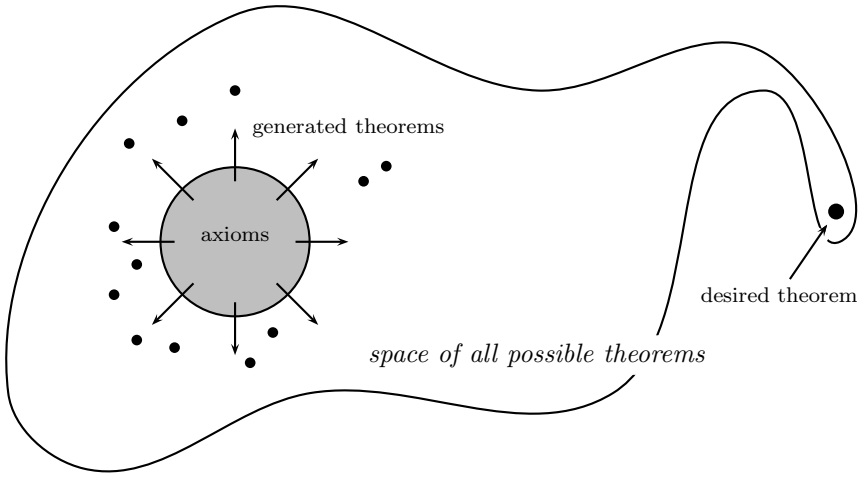


Figure 2.1: The British Museum Algorithm

2.1 The nature of theorem proving

One way to prove theorems is to begin with axioms and rules of inference, and to infer theorems on the basis of the axioms and theorems that were inferred earlier. A good example of this is

Hilbert's system (page 40), that has three axioms and one rule of inference.[1] The formula $p \supset p$, for instance, can be proven in Hilbert's system by chaining forward from the tree axioms, using Modus Ponens (MP) as the sole rule of inference. Now do Exercise 1. (I'll wait...) Was it easy to find a proof? Probably not. Did you peek at the solutions? Probably, you did. Whether you consulted the solutions or not, the point is that computers can't. Computers (and badly programmed automated theorem provers) would just crank out theorems, hoping that the desired theorem is among them. This approach is known as the British Museum Algorithm: examine all things, until you encounter something interesting. Obviously, the British Museum Algorithm is not going to work (Figure 2.1 on the preceding page).

So the British Museum Algorithm is obviously not the way to go for ATP. How does it work then? Somewhat paradoxically, ATP is not done by trying to prove theorems. Although there are rare exceptions (such as the Rete-algorithm), virtually 99% of the "real" theorem proving is done by refutation. This method will also be followed in the next section.

PROBLEMS (Sec. 2.1)

1. Consider Hilbert's system on page 40. Here is an example of a proof of $p \supset q, q \supset r \vdash p \supset r$ in that system:

1. $p \supset q$	(Hypothesis)
2. $q \supset r$	(Hypothesis)
3. $(q \supset r) \supset (p \supset (q \supset r))$	(Instance of Axiom Schema 1)
4. $(p \supset (q \supset r)) \supset ((p \supset q) \supset (p \supset r))$	(Instance of Axiom Schema 2)
5. $p \supset (q \supset r)$	(From 2, 3 by MP)
6. $(p \supset q) \supset (p \supset r)$	(From 4, 5 by MP)
7. $p \supset r$	(From 1, 6 by MP)

   Try to produce a proof of $p \supset p$. (Solution.)

2. The *Rete* algorithm is a forward-chaining algorithm that is used in the area of knowledge-based systems [35, 109]. It supposes a set of rules, R, and a set of facts, or axioms, A. For example,

   $$R = \{p, r \rightarrow \neg q;\ r \rightarrow s\}$$

   and $A = \{p, r, t\}$. The central data-structure of the algorithm is a list of conclusions, or theorems, T. This set is initialized to A, and is supposed to store all theorems during the theorem proving process. Then a cycle starts. At each round, the antecedent of one or more rules is *matched* against the contents of T. (The antecedent of, e.g., $p, r \rightarrow \neg q$ is p, r.) If all elements of an antecedent are in T (order of the elements is not important), the rule *fires*, and its conclusion is added to T. In the running example, $\{p, r\} \subseteq T$, so that $p, r \rightarrow \neg q$ fires and $\neg q$ is added to T.

   For reasons of simplicity, we have described the propositional variant of the Rete algorithm. The full version works with unification (page 72).

   (a) It is difficult to apply the Rete algorithm to Hilbert's system. Why? (Hint: consult page 40. What does the notion of axiom schema mean?) (Solution.)

   (b) Suppose the following rules are given:

   $$\begin{array}{rr} a, e \rightarrow d & \neg f \rightarrow b \\ e, \neg c \rightarrow a & a, b \rightarrow t \\ d, \neg c, e \rightarrow \neg f & \end{array}$$

   Let $A = \{\neg c, e\}$. Show how t can be proved with the Rete algorithm. (Solution.)

[1] The first-order version has an extra rule, called *generalization*: from ϕ we may infer $(\forall x)(\phi)$, where x is any variable.

(c) Draw a tree that shows how the conclusions (theorems and intermediary theorems) follow from other conclusions.

(d) Propose an algorithm that proves theorems on the basis of Hilbert's system, using the Rete algorithm. (Hint: the axioms can be enumerated.) (Solution.)

(e) Probably, the Algorithm proposed at Item (2d) is inefficient, in the sense that it cranks out theorems until the desired formula shows up. Thus, an improvement of the algorithm proposed in Item (2d), would be highly desirable. Try to explain why such an improvement is unlikely to exist.

3. The Rete algorithm is a forward-chaining algorithm: it starts with axioms, and chains rules into proofs with the aim to arrive at the desired theorem. A *backward-chaining inference procedure* is one that starts with the theorem, and tries to work its way back to the axioms. (Prolog, for instance, possesses a backward-chaining inference engine.)

(a) Try to formulate a backward-chaining algorithm for Hilbert's system. Why does it not work? (Solution.)

(b) Any idea why backward-chaining *does* work with Prolog (contrary to backward-chaining with Hilbert's system)? (Solution.)

2.2 Searching for a counterexample

Suppose we want to investigate the validity of a certain sequent in propositional logic, for example

$$\neg q, \neg(p \wedge q) \vdash p \supset q. \tag{2.1}$$

(This sequent is actually invalid, but that is not our major concern here.)

A systematic way to determine the status of (2.1), is to try to make it false. If we have considered every possibility to falsify it and fail, we have proven that the sequent is valid.

Let us try to falsify (2.1). To this end, will have to make both of $\neg q, \neg(p \wedge q)$ true and $p \supset q$ false. Let us write this as

$$\text{TRUE: } \neg q, \neg(p \wedge q) \ ; \quad \text{FALSE: } p \supset q$$

From here, our analysis proceeds systematically. Let us begin with the implication on the RHS. (Any formula is okay, as long as it contains a connective.) To make the implication $p \supset q$ on the RHS false, we will have to make p true and q false. Thus, we discard $p \supset q$, put p at the left and q at the right:

$$\text{TRUE: } \neg q, \neg(p \wedge q), p \ ; \quad \text{FALSE: } q$$

To make $\neg q$ true, we will have to make q false:

$$\text{TRUE: } \neg(p \wedge q), p \ ; \quad \text{FALSE: } q, q$$

The RHS now contains two q's. It is possible (and permitted) to discard one occurrence of q, but in this case we proceed. To make $\neg(p \wedge q)$ true, we will have to make $p \wedge q$ false:

$$\text{TRUE: } p \ ; \quad \text{FALSE: } q, q, p \wedge q$$

To make $p \wedge q$ false, it suffices to falsify either one of them. Hence, our search for a counterexample branches into two directions: (1) one in which we try to falsify p and (2) one in which we try to falsify q:

$$\text{(1) TRUE: } p \ ; \quad \text{FALSE: } q, q, p \qquad\qquad \text{(2) TRUE: } p \ ; \quad \text{FALSE: } q, q, q$$

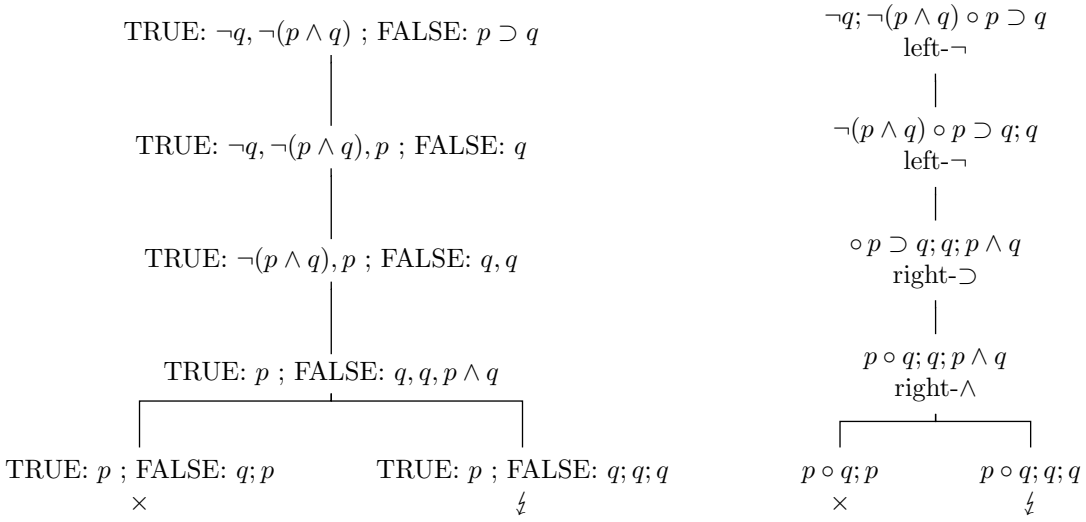


Table 2.1: Refutation trees of $\neg q, \neg(p \wedge q) \vdash p \supset q$.

At this point all formulas are atomic and we can inspect the endpoints of all branches for possible counterexamples. The first branch does not contain a counterexample, because it is impossible to make p both true and false. The second branch does contain a counterexample, if make p true and q false. Thus, $\{p = 1,\ q = 0\}$ is a counterexample for (2.1). Hence, (2.1) is invalid.

The search for a counterexample can be written down in a more compact form. When we do so, we obtain Fig. 2.1 (left), or better still, Fig. 2.1 (right). The trees that we obtain are known as *refutation trees*, *semantic trees*, or *(semantic) tableaux*. The $\circ$-symbol is used to separate formulas that must be made true (LHS) from formulas that must be made false (RHS). If two equal atoms occur on both sides of the "$\circ$," the node cannot serve as a counterexample and we put a $\times$-symbol below the branch to indicate that search has terminated for this particular branch and we say that this particular branch is *closed*. If the LHS and RHS of a node do *not* share a common atom, the branch remains *open*. If a branch cannot be further extended (either because it is closed or because all formulas on it have been analyzed) it is said to be *complete*, or *completed*. A completed but open branch is called *saturated*. In propositional logic, the leaf (end-node) node of a saturated branch represents a counterexample, indicated by a $\lightning$. This counterexample can be created by making all LHS-atoms true and all RHS-atoms false. In first-order logic, a saturated branch may be infinite. In such cases, a countermodel is specified by the so-called ground-literals that occur on a saturated branch.

Example 2.1 Above, we have seen a refutation that succeeds. Figure 2.2 on page 30 shows a refutation that fails. Since our search for a counterexample is exhaustive, we may conclude that the sequent in question, namely $p; q; ((p \supset (q \supset r)) \vee (p \supset r)) \vdash r$, is valid.

PROBLEMS (Sec. 2.2)

1. What does it mean for a branch to be open and completed? Similar questions for the combinations open/incomplete, closed/completed, and closed/incomplete. (Solution.)

2. Complete the following table. How many different cases are there to analyze?

Rules for building a refutation tree	
- If a conjunction $p \wedge q$ must be made true, then try to make both p and q true.	- If a conjunction $p \wedge q$ must be made false, then it suffices to make either p or q false: split.
- If a disjunction $p \vee q$ must be made true, ...	...
⋮	⋮

(Solution.)

3. Construct refutation trees for the following sequents. Specify a counterexample if the sequent turns out to be invalid.

   (a) $\neg(p \vee q), \neg p \supset (\neg q \supset \neg r) \vdash r$

   (b) $k \supset (e \wedge o), o \supset (b \wedge t) \vdash \neg k \wedge t$

   (c) $b \wedge \neg d, a \supset (b \supset (c \supset d)) \vdash c \supset \neg a$

   (d) $j \supset ((k \wedge l) \supset m), k \wedge \neg m \vdash l \supset j$

   (e) $(h \supset i) \supset (j \vee k), (\neg k \wedge \neg h) \vee (\neg k \wedge i) \vdash j$

   (Solution.)

4. What is wrong with the following statement? "The refutation tree of $p \wedge q \wedge r \vdash p$ closes." (Solution.)

5. The idea of proof-by-refutation is somewhat indirect, in the sense that a failed refutation is an indirect proof of the validity of a sequent. Why not taking a more straightforward approach by trying to *confirm* a sequent (i.e., by trying to make it true)? (A sequent is confirmed as soon as either the LHS is made false or the RHS is made true.) If this succeeds the sequent is valid, else it is invalid. Explain under which circumstances this could work. Why is it not a good idea, after all? (Solution.)

6. Prove that the number of sub-formulas of a formula linearly depends on the length of the main formula. (Solution.)

7. Give left- and right-reduction rules that reduce the complexity of the formula, for the following connectives:

   (a) Equivalence: $\equiv$. [A candidate answer would be a rule that reduces $p \equiv q$ to $(p \supset q) \wedge (q \supset p)$. This would work, but it does not reduce the complexity of the formula. Try to find a rule analogous to rules for $\wedge$ and $\vee$.] (Solution.)

   (b) Exclusive or: $\oplus$. [If you find a rule that reduces $p \oplus q$ to $(p \vee q) \wedge (\neg(p \wedge q))$, try to do better].

An overview of all rules is given in Table 2.2 on page 31. These rules possess three crucial properties:

1. *Sub-formula property.* Each rule analyzes a formula on the basis of its outermost connective.
2. *Complete analysis.* Each non-atomic formula in a sequent can be analyzed by at least one rule.
3. *Unique analysis.* Each non-atomic formula in a sequent can be analyzed by at most one rule.

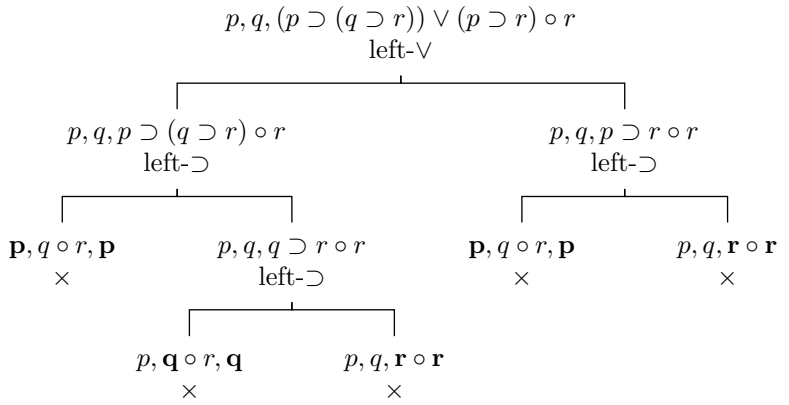


Figure 2.2: Failing refutation of $p, q, p \supset (q \supset r)) \vee (p \supset r) \vdash r$.

Property 1 says that, refutation (according to the rules displayed in Table 2.2) is an analytic process. The system displayed in Table 2.2 is therefore known as *analytic refutation*. An advantage of analytic refutation is that the new formulas that occur along the tree are sub-formulas of the formulas that appear in the original sequent. In particular, the sub-formula property guarantees that only relevant formulas are introduced. Another advantage of the sub-formula property is that relatively few new formulas are introduced in the refutation proces, because the number of sub-formulas linearly depends on the length of a formula (Excercise 6 on the previous page). Property 2 says that a refutation never gets stuck at non-atomic formulas. So all connectives can be eliminated, if necessary. Property 3 says that the number of rules that can be applied at a specific formula is small, which is an advantage from a computational point of view. Properties (1-3) together show that analytic refutation is a process that allows little freedom. The only degree of freedom we have is the choice which formulas in the sequent we reduce first (Exercise 4a on page 33). It may seem that properties (1-3) are trivial, and could not have been otherwise. In Section 2.6 we will see that this is not the case.

The main reason why analytic refutation is simple, is that it possesses the sub-formula property. The sub-formula property amounts to the fact that we are only analyzing sub-formulas of the original formula during the refutation process. We begin with one big formula at the top of the tree, and break it down to smaller and smaller sub-formulas, until we discover a countermodel or until we have closed all branches. Another positive feature of analytical refutation, is that the number of sub-formulas linearly depends on the size of the original formula (Exercise 6 on the previous page). Unfortunately, the linearity result does not guarantee that the number of newly generated formulas remains within reasonable bounds, for one and the same formula may occur several times in a refutation tree.

The next section, analyzes the process of building refutation trees in more detail for the purpose of mechanizing it.

2.3 Turning a refutation tree into a proof

If all branches in a refutation tree are closed, and no branch ends in a counterexample, it may be concluded that the refutation has failed. We may thus consider that tree as a *proof* of the sequent in question. The axioms of such a proof are, by definition, sequents in which the LHS and the RHS share one or more common proposition variables.

To turn a refutation tree into a "real" proof, we turn it upside down and supply every step with

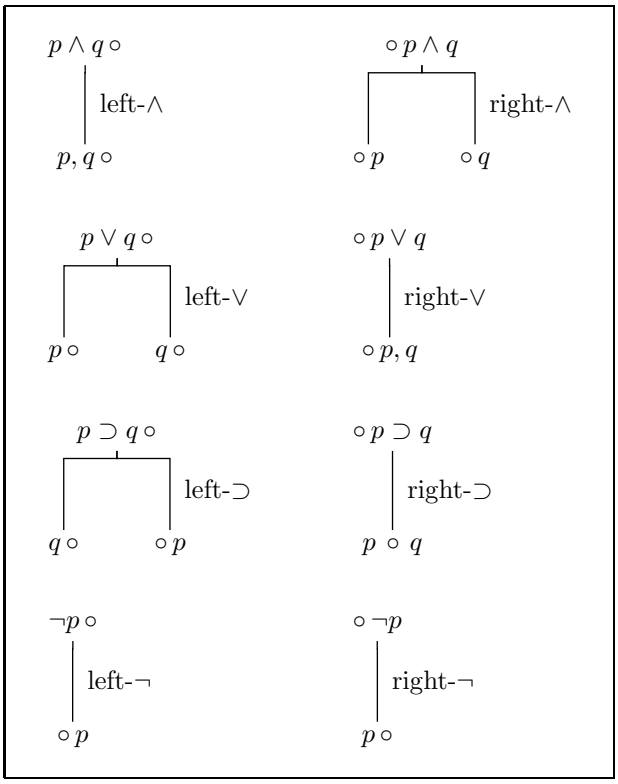


Table 2.2: Analytic refutation rules.

a justification. In this way, we obtain a so-called *cut-free* proof in the *Gentzen sequent calculus*. (See also page 39.) For instance,

$$\dfrac{\dfrac{p, q \Rightarrow r, p \qquad \dfrac{p, q \Rightarrow r, q \quad p, q, r \Rightarrow r}{p, q, (q \supset r) \Rightarrow r}\ \text{left-}\supset}{p, q, (p \supset (q \supset r)) \Rightarrow r}\ \text{left-}\supset \qquad \dfrac{p, q \Rightarrow r, p \quad p, q, r \Rightarrow r}{p, q, (p \supset r) \Rightarrow r}\ \text{left-}\supset}{p, q, ((p \supset (q \supset r)) \vee (p \supset r)) \Rightarrow r}\ \text{left-}\vee$$

is the proof of the formula that was refuted in Fig. 2.2 on the facing page. Fig. 2.3 on the next page shows how a tableau-based theorem prover searches for a countermodel, fails to find one, and prints a linear proof (Fig. 2.4 on page 33).

The complete table of rules of the Gentzen sequent calculus is given on page 34. Note that a sequence of formulas one the RHS of a sequent has to be red disjunctively. For example, a sequent such as $p, q \vdash q, p \wedge q$ reads "from p and q we can infer q, or $p \wedge q$, or both". Also note that the dots in Table 2.3 represent a context that does not change. For example,

$$\dfrac{r \vdash p \quad s \vdash q}{r, s \vdash p \wedge q}\ \text{wrong!} \qquad \text{and} \qquad \dfrac{r \vdash s, p \quad r \vdash q, t}{r \vdash s, p \wedge q, t}\ \text{wrong!}$$

are improper inferences, because Gentzen does not permit to merge contexts. The inference

$$\dfrac{r \vdash s, p, t \quad r \vdash s, q, t}{r \vdash s, p \wedge q, t}\ \text{right-}\wedge$$

is an example of a proper application of $\wedge$-right.

```
Sequent1: q,~p&q => p>q.
Left contains unique formulas.
Right contains unique formulas.
Not an axiom.
Applying left-& yields Sequent2: q,~p,q => p>q.

Next: Sequent2: q,~p,q => p>q (from 1).
Applying cloning q yields Sequent3: q,~p => p>q.

Next: Sequent3: q,~p => p>q (from 2).
Left contains unique formulas.
Right contains unique formulas.
Not an axiom.
Applying left-~ yields Sequent4: q => p>q,p.

Next: Sequent4: q => p>q,p (from 3).
Left contains unique formulas.
Right contains unique formulas.
Not an axiom.
No simple left reductions.
Applying right-> yields Sequent5: q,p => q,p.

Next: Sequent5: q,p => q,p (from 4).
Left contains unique formulas.
Right contains unique formulas.
Not an axiom.
Applying thinning yields Sequent6: q => q.

Next: Sequent6: q => q (from 5).
Left contains unique formulas.
Right contains unique formulas.
Sequent is an axiom.  Proceeding.

Queue empty.

No countermodel found.  Provable.
```

Figure 2.3: Searching for a countermodel for $q, \neg(p \wedge q) \vdash p \supset q$.

PROBLEMS (Sec. 2.3)

1. In Exercise 3 on page 29, you constructed refutation trees for a number of sequents. Some of these refutations failed. Use the refutations that failed to construct proofs in the Gentzen cut-free sequent calculus. Identify the axioms in each proof. (Solution.)

2. Let $a_1, \ldots, a_m$ and $b_1, \ldots, b_n$ be propositions. Show with the help of refutation trees that the following statements are equivalent.

   (a) $a_1, \ldots, a_m \vdash b_1, \ldots, b_n$ iff

   (b) $a_1 \wedge \ldots \wedge a_m \vdash b_1 \vee \ldots \vee b_n$ iff

   (c) $\vdash [a_1 \wedge \ldots \wedge a_m] \supset [b_1 \vee \ldots \vee b_n]$ iff

   (d) $\vdash \neg[a_1 \wedge \ldots \wedge a_m \wedge \neg b_1 \wedge \ldots \wedge \neg b_n]$ iff

   (e) It is impossible to make all of $a_1, \ldots, a_m$ true, and all of $b_1, \ldots, b_n$ false, iff

   (f) It is impossible to make $\vdash a_1 \wedge \ldots \wedge a_m$ true, and $b_1 \vee \ldots \vee b_n$ false

```
== Proof: ================================
  1. q => q (axiom).
  2. q,p => q,p (1, thinning).
  3. q => p>q,p (2, right->).
  4. q,~p => p>q (3, left-~).
  5. q,~p,q => p>q (4, cloning q).
  6. q,~p&q => p>q (5, left-&).
========================================= QED =
```

Figure 2.4: Printing a proof of $q, \neg(p \wedge q) \vdash p \supset q$.

```
[snip]

Next Sequent13:  p,q,s => r,t (from 7).
Left contains unique formulas.
Right contains unique formulas.
Not an axiom.
No simple left reductions.
No simple right reductions.
No splitting left reductions.
No splitting right reductions.

Counterexample: p,q,s true; r,t false.

Halt.
```

Figure 2.5: Discovery of a counterexample.

What if m or n are equal to 0? What is the meaning of such sequents? (Solution.)

3. Show that, in propositional logic, the process of building a refutation tree is a finite process. Hint: call a propositional formula complex if it contains many connectives. Define a measure of complexity ≥ 0 on formulas, and show that something happens with the complexity in the refutation of a formula.

4. Describe an algorithm for deciding whether a sequent is valid. Take into consideration the following problems:

   (a) How do you scan a sequent? Left to right, or largest formulas first?

   (b) Which type of formulas on the left are reduced first? Why? Same question for the RHS of a sequent.

   (c) What do you do with a reduced sequent? Do you continue processing it, or is it put aside waiting for other sequents to be processed first? What type of data structure do you use to store the sequents that are put aside? A stack (last in, first out), a queue (first in, first out), or a heap (prioritized queue). If you use a heap, what prioritization do you use?

   (d) Is your algorithm *fair*? I.e., can you guarantee, or even prove, that every non-axiomatic sequent produced will eventually be taken up for further processing?

   All choices are okay, but be sure motivate them. Implement the algorithm in a programming language of your choice.

left-$\wedge$: $\dfrac{\ldots, p, q, \ldots \vdash \ldots}{\ldots, p \wedge q, \ldots \vdash \ldots}$	right-$\wedge$: $\dfrac{\ldots \vdash \ldots, p, \ldots \quad \ldots \vdash \ldots, q, \ldots}{\ldots \vdash \ldots, p \wedge q, \ldots}$
left-$\vee$: $\dfrac{\ldots, p, \ldots \vdash \ldots \quad \ldots, q, \ldots \vdash \ldots}{\ldots, p \vee q, \ldots \vdash \ldots}$	right-$\vee$: $\dfrac{\ldots \vdash \ldots, p, q, \ldots}{\ldots \vdash \ldots, p \vee q, \ldots}$
left-$\supset$: $\dfrac{\ldots \vdash \ldots, p, \ldots \quad \ldots q \ldots \vdash \ldots}{\ldots, p \supset q, \ldots \vdash \ldots}$	right-$\supset$: $\dfrac{\ldots, p, \ldots \vdash \ldots, q, \ldots}{\ldots \vdash \ldots, p \supset q, \ldots}$
left-$\neg$: $\dfrac{\ldots \vdash \ldots, p, \ldots}{\ldots, \neg p, \ldots \vdash \ldots}$	right-$\neg$: $\dfrac{\ldots, p, \ldots \vdash \ldots}{\ldots \vdash \ldots, \neg p, \ldots}$

Table 2.3: Gentzen system for propositional logic.

Sound- and completeness of refutation trees

Two questions remain to be answered:

1. *Soundness.* Soundness means that we must be certain that refutations fail only for valid sequences. Thus, in order to prove soundness it suffices show that all refutations of invalid sequences succeed, that is, always yield a counterexample.
2. *Completeness.* Completeness means that we must be certain that refutations fail for all valid sequences, in finitely many step. Thus, in order to prove completeness it suffices show that refutations succeed for invalid sequences only.

Let us first try to see why invalid sequents are always refuted (soundness). To this end, suppose

$$\phi_1, \ldots, \phi_m \vdash \psi_1, \ldots, \psi_n \tag{2.2}$$

is invalid. This means that there is a valuation, or model, w, such that $w(\phi_1) = \cdots = w(\phi_m) = 1$ and $w(\psi_1) = \cdots = w(\psi_n) = 0$. Now if we try to refute (2.2) by means applying rules from Table 2.2 on page 31, we are always led to at least one other invalid sequent (Exercise 1). Since invalid sequents never close (Exercise 2 on the next page), we produce a branch in the refutation tree of (2.2) that is open and stays open. Since a refutation is, by definition, a branch for which we have show that it cannot be cannot be closed, we have found a refutation of (2.2).

Conversely, suppose that there exists a refutation of (2.2). This means that there is a refutation tree with an open branch. We will have to show that (2.2) is, indeed, invalid. This is easy within the context of propositional logic, since refutation trees in propositional logic have finite branches. Let w be the valuation, or model, w that corresponds with the leaf (end-node) of the branch that falsifies (2.2). By analyzing the rules from Table 2.2 on page 31 in reverse, we observe that all sequents above this end-node are falsified by w as well. In particular, the top-node (2.2) is falsified by w.

PROBLEMS (Sec. 2.3)

1. (a) i. Show that if $\phi_1, \ldots, \phi_m, \phi \wedge \psi \nvdash \psi_1, \ldots, \psi_n$, then $\phi_2, \ldots, \phi_m, \phi, \psi \nvdash \psi_1, \ldots, \psi_n$.

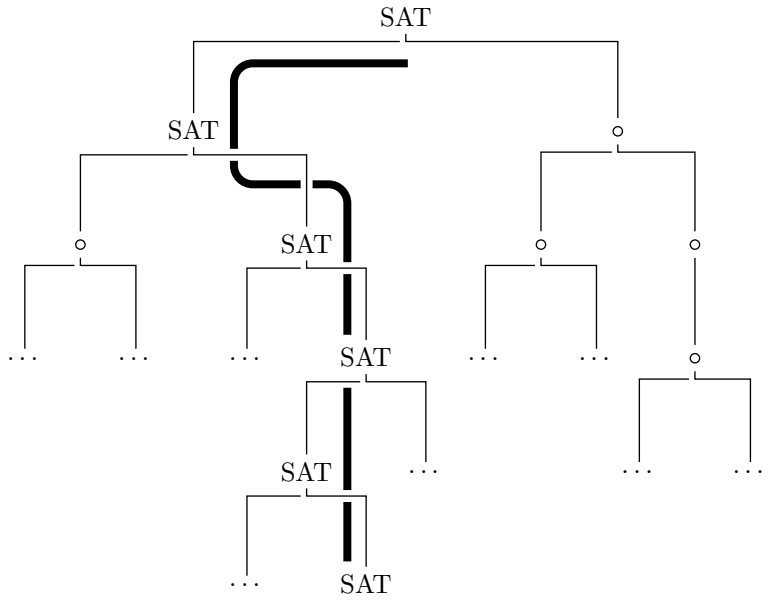


Figure 2.6: An invalid sequent preserves falsifiability in at least one branch

ii. Show that if $\phi_1, \ldots, \phi_m, \phi \vee \psi \nvdash \psi_1, \ldots, \psi_n$, then either $\phi_2, \ldots, \phi_m, \phi \nvdash \psi_1, \ldots, \psi_n$ or $\phi_2, \ldots, \phi_m, \psi \nvdash \psi_1, \ldots, \psi_n$.

(b) Argue that the invalidity of a falsifiable sequent at the root of a refutation tree propagates downwards through at least one branch of a refutation tree.

2. Why can't a branch be closed by an invalid sequent? I.e., why is it impossible to put a cross near a node in a refutation tree with an invalid sequent?

2.4 A hard case

Because of the sub-formula property, analytic refutation is conceptually simple, and can therefore be straightforwardly implemented on a computer.

If you have actually implemented the analytic tableau method, and tried it out with a dozen of standard textbook examples, then everything probably went just fine. (Your theorem prover produced either proofs or counterexamples, and that was it.) The fact that an algorithm is conceptually simple, however, does not necessarily mean that it is fast. (You knew that, of course.) Maybe your algorithm *is* fast on the examples you gave it. But what about other examples? Is analytic refutation really as good as a handful of experiments would suggest?

Unfortunately, the answer is no. A major drawback of analytic refutation is that trees sometimes may grow extremely fast, even for formulas of relatively small size. The following set of exercises is meant to illustrate this.

PROBLEMS (Sec. 2.4)

1. Consider the following formula:

$$(p \wedge q) \vee (p \wedge \neg q) \vee (\neg p \wedge q) \vee (\neg p \wedge \neg q). \tag{2.3}$$

To give this formula some meaning, let p stand for "Pete spoke the truth," $\neg p$ for "Pete lied," q for "Quincy spoke the truth," and $\neg q$ for "Quincy lied". What does (2.3) mean? If $p =$ "Parrots fly," and $q =$ "Elephants fly," then what does (2.3) mean? Is (2.3) valid? (Solution.)

2. Investigate the validity of (2.3) with the help of an analytic refutation tree. (Solution.)
   (a) How many branches, or paths, does the refutation tree of (2.3) have? How many of them result in a contradiction? (Solution.)
   (b) Are there parts of the search space that already have been explored? If so, indicate the doublures.

3. Let, for each $n \geq 1$,

$$f_n =_{Def} \vee\{ \pm p_1 \wedge \ldots \wedge \pm p_n \mid \pm p_i \text{ is either } p_i \text{ or } \neg p_i \}. \tag{2.4}$$

   Thus,

$$\begin{aligned} f_1 &= p_1 \\ f_2 &= (p_1 \wedge p_2) \vee (p_1 \wedge \neg p_2) \vee (\neg p_1 \wedge p_2) \vee (\neg p_1 \wedge \neg p_2) \\ f_3 &= (p_1 \wedge p_2 \wedge p_3) \vee (p_1 \wedge p_2 \wedge \neg p_3) \vee (p_1 \wedge \neg p_2 \wedge p_3) \vee (p_1 \wedge \neg p_2 \wedge \neg p_3) \vee \\ &\quad (\neg p_1 \wedge p_2 \wedge p_3) \vee (\neg p_1 \wedge p_2 \wedge \neg p_3) \vee (\neg p_1 \wedge \neg p_2 \wedge p_3) \vee (\neg p_1 \wedge \neg p_2 \wedge \neg p_3) \\ \vdots &\quad \vdots \end{aligned}$$

   (a) How many disjuncts does f_n has? How many conjuncts does each disjunct has? What is the length of f_n? (The last answer may be given modulo some fixed constant.) (Solution.)
   (b) Construct a truth-table for f_2. Is f_2 valid? Is f_n valid? (Solution.)
   (c) How many disjuncts has f_n? How many rows does the truth-table of f_n has? (Solution.)
   (d) What can be said about the efficiency of the truth-table method compared to the length of f_n? (Solution.)
   (e) Investigate the validity of f_2 by means of analyzing it with the help of a refutation tree. How many branches does this tree have? Do the same for f_3. (Solution.)
   (f) Let $B(n)$ be the number of branches of a minimal refutation tree of f_n. Show that $B(n) \geq n!$. This question is best answered by starting out making a refutation tree with root "$f_n \circ$" and describe what happens. (Solution.)
   (g) What does $B(n)$ tell us about the efficiency of analytic refutation of f_n? (Solution.)

The above exercises show that there exist hard cases for which analytic refutation performs very badly—sometimes even worse than the truth-table method. This behavior is caused by the fact that the analytic refutation method explores parts of the search space that already *have* been explored. After building the left subtree for (2.3), for example, we know that assuming that p is true leads to a contradiction, yet we repeat this search even after assuming that b is true. (Don't be misled by the tree into thinking that using graphs instead of trees might help. Refutation trees are really decorated at each node with the set of formulas along the path from the root, and each node in the above tree corresponds to a different set.) This kind of redundancy can be repeated inside the redundant sub-trees, so that a combinatorial explosion results. Thus, the problem is that analytic refutation does not faithfully match the search space that we are trying to explore. With Exercise 3 we have in fact shown that there exists a class of formulas for which analytic refutation is embarrassingly outperformed by a simple brute-force method like truth-tables. This is of course unacceptable, so that we must try to improve on analytic refutation or look for another method.

2.5 Propositional theorem proving is NP-complete

Before trying to improve on analytic refutation, we will have to face a less friendly side of propositional ATP.

Most (if not all) propositional ATP amounts to finding a countermodel for a propositional formula. This problem is known as the *satisfiability problem*, or *SAT*. It is a well-known fact within the theory of computational complexity that SAT is NP-complete. If a problem is NP-complete, this means that it falls within a large class with thousands of other problems for which it can be proved that they are equally hard. All problems that are NP-complete can be solved by algorithm in exponential time. Thus, NPC $\subseteq$ EXP, and in practice this is the best we can get. No polynomial time algorithms are known to exist for any of the NP-complete problems and is very unlikely that polynomial time algorithms should indeed exist although nobody has yet been able the to prove their non-existence. (Better known as the big P =? NP question.)

All this means, among other things, that it is extremely likely, and in fact almost certain, that no algorithm exists that solves SAT in a computationally efficient way [30]. Since checking for a tautology amounts to a check of non-SAT, this means that propositional ATP is co-NPC. Thus, it is extremely likely, and in fact almost certain, that no algorithm exists that decides on all propositional formulas in a computationally efficient way.

Knowing that SAT is NP-complete lets you stop beating your head against a wall trying to solve it, and do something better:

- *Use a heuristic.* If you can't quickly solve the problem with a hard case, maybe you can come up with a method for solving a reasonable fraction of the common cases. This is what we do in the next section.
- *Use an exponential time solution anyway.* If you really have to solve the problem exactly, you can settle down to writing an exponential time algorithm and stop worrying about finding a better solution. (Which is what we did in the previous sections.)
- *Solve the problem approximately instead of exactly.* A lot of the time it is possible to come up with a provably fast algorithm, that doesn't solve the problem exactly but comes up with a solution you can prove is close to right. This is what we do in Chapter 6 (page 127ff).
- *Choose a better abstraction.* The NP-complete logic you're dealing with presumably comes from incorporating unimportant linguistic details. Perhaps some of those details should be ignored, and make the difference between what you can and can't solve. This approach is followed in Chapter 8 onwards, where we abstract from the logical language and deal with rules that are constituted from a syntax that is less fine-grained than the language of propositional and first-order logic.

All the above by no way means that propositional ATP is deemed worthless. It just means that every propositional theorem prover has its own "black region" of formulas that cannot be proven in polynomial time. In fact, most theorem provers are able to prove or disprove almost all propositional formulas within reasonable time. Moreover, it is impossible to fabricate a problem that is hard for *all* theorem provers. (This is an interesting exercise with a surprisingly simple solution. See page 39) Thus, perfect theorem provers do not exist, but it always makes sense to try to improve on them.

2.6 The system KE

What could be "wrong" with analytic refutation? And what could cause the explosive growth of trees for some formulas? One possible answer is that the different branches of refutation trees do not correspond to mutually exclusive alternatives. For example, when we try to refute $\phi \vee \psi$, and ϕ is a subformula of ψ, we may end up analyzing ϕ more than once, particularly if all branches below the LHS of $\phi \vee \psi$, i.e. below ϕ, close. In that case, analytic refutation is forced to "eat"

(break down, analyse) ψ. If, in the process of analyzing ψ, we encounter ϕ before having produced a counterexample, then traditional analytic refutation "eats" ϕ again. The fact that ϕ *was* already analyzed (without success), is simply forgotten.

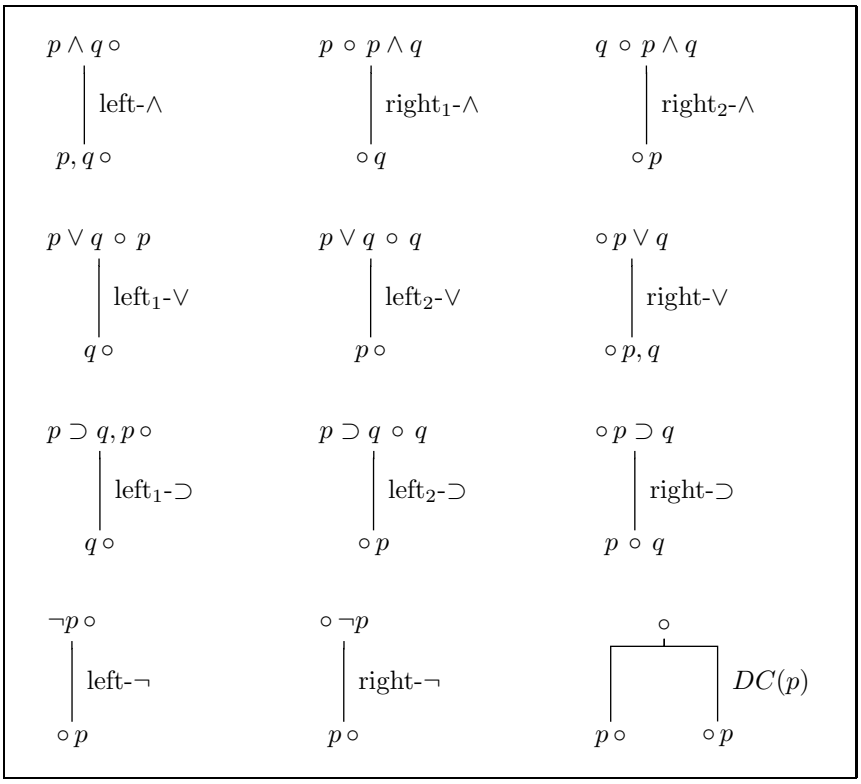


Table 2.4: The system KE

One way to prevent the exponential growth of branches is to abandon all branching rules (viz. right-$\wedge$, left-$\vee$, and left-$\supset$). In this way, refutation becomes a linear process that runs along a single branch.

However, it probably does not take much imagination to foresee that by abandoning a number of crucial rules, our refutation method is no longer complete. To compensate this loss of "analytic power," we introduce six new linear rules that are able to analyse the formerly tree-splitting RHS-conjunctions, LHS-disjunctions, and LHS-implications, but now linearly so (Table 2.5). The resulting system is called *linear KE*.

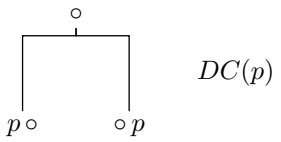
Although the nett number of rules has been increased with three, some of the RHS-conjunctions, LHS-disjunctions, and LHS-implications cannot be analyzed (why? – look at the rules). Thus, the new rules fail to cover the rules they are meant to replace, which means that linear KE, i.e., the arsenal

left-$\wedge$,	right$_1$-$\wedge$,	right$_2$-$\wedge$,
left$_1$-$\vee$,	left$_2$-$\vee$,	right-$\vee$,
left$_1$-$\supset$,	left$_2$-$\supset$,	right-$\supset$,
left-$\neg$,	right-$\neg$	

is incomplete in the sense that more rules are needed to establish logical soundness. Hence, the present system is logically complete (Exercise 2b on the facing page) but logically unsound (Exercise 2a on the next page).

It can be proven[2] that every complete refutation method needs at least one rule that branches. So what is missing here is one (or more) branching rules that complete the system.

One branching rule that might do the job is



(2.5)

or, in sequent calculus,

$$\frac{\ldots \vdash \ldots, p, \ldots \quad \ldots, p, \ldots \vdash \ldots}{\ldots \vdash \ldots} \; DC(p)$$

Let us call this rule "distinguishing two cases," or "distinguishing cases" (DC). One name under which DC is known in the literature is "PB". "PB" stands for "principle of bivalence," which means that a logical formula can assume only one of two values.[3] Another name under which this rule is known in the literature is "dilemma". This name suggests a forced choice in one of two directions. In resolution-style theorem proving, the rule is known as "branch/merge," because the two distinguished cases must be merged back into one clause set before resolution may be continued. In daily speech, DC is also called "'the shotgun". This is with reason: the DC-rule is powerful and nice to have but may get us into trouble when used inappropriately. The resulting system is called *KE* (Table 2.4 on the preceding page). KE stands for "Klassical Eliminative" where "K" for "Classical" was a tribute to Gentzen's LK (moreover CE sounded awkward)[21, 22]. [4] Thus, KE = linear KE + DC.

In a moment, we will prove that DC makes refutation complete (Exercise 4 on the following page). Another nice property of DC is that it ensures that branches in refutation trees are mutually exclusive (i.e., not simultaneously satisfiable). This property can be read of from DC immediately. So far for the positive properties of DC. An inferior property of DC is that it can be applied unconditionally. At any point in the tree, and at any time, we (or the algorithm) can decide happily to apply it. A second inferior property of DC is that it can introduce *any* formula into the refutation, also formulas that are irrelevant to the original formula in question. Therefore, we must know when to apply DC and (better) when *not* to apply it. This is the subject of the next section.

PROBLEMS (Sec. 2.6)

1. Prove that it is impossible to fabricate a problem that is hard for *all* theorem provers. (Solution.)

2. (a) Show that linear KE is unsound, i.e., find a contingent formula (i.e., non-tautology) that can be proven in linear KE. (Solution.)
   (b) Prove that linear KE is complete. (Solution.)

3. (a) Formulate SAKE-rules for handling logical equivalence. (Solution.)
   (b) Are

$$p \equiv q \tag{2.6}$$

[2] Proof beyond the scope of this chapter. See [20].

[3] Indeed, linear KE as well as pure analytical refutation (Table 2.2 on page 31) are sound for some many-valued logics. E.g., it has been shown that the semantics of cut-free systems can be described in terms of a 3-valued logic.

[4] d'Agostino: *we never said what KE stands for. I think it stands for "Klassical Eliminative" where "K" for "Classical" was a tribute to Gentzen's LK (moreover CE sounded awkward).*

replace branching $\wedge$-right rule by linear rules

$$\frac{\ldots \vdash \ldots, q, \ldots}{\ldots, p, \ldots \vdash \ldots, p \wedge q, \ldots}\ \text{right}_1\text{-}\wedge \quad \text{and} \quad \frac{\ldots \vdash \ldots, p, \ldots}{\ldots, q, \ldots \vdash \ldots, p \wedge q, \ldots}\ \text{right}_2\text{-}\wedge$$

replace branching $\vee$-left rule by linear rules

$$\frac{\ldots, q, \ldots \vdash \ldots}{\ldots, p \vee q, \ldots \vdash \ldots, p, \ldots}\ \text{left}_1\text{-}\vee \quad \text{and} \quad \frac{\ldots, p, \ldots \vdash \ldots}{\ldots, p \vee q, \ldots \vdash \ldots, q, \ldots}\ \text{left}_2\text{-}\vee$$

replace branching $\supset$-left rule by linear rules

$$\frac{\ldots, q, \ldots \vdash \ldots}{\ldots, p, p \supset q, \ldots \vdash \ldots}\ \text{left}_1\text{-}\supset \quad \text{and} \quad \frac{\ldots \vdash \ldots, p, \ldots}{\ldots, p \supset q, \ldots \vdash \ldots, q, \ldots}\ \text{left}_2\text{-}\supset$$

left-$\neg$ and right-$\neg$ rules remain the same (they are already linear)

\+ new "distinguishing cases"-rule (aka "cut"):

$$\frac{\ldots \vdash \ldots, p, \ldots \quad \ldots, p, \ldots \vdash \ldots}{\ldots \vdash \ldots}\ DC(p)$$

Table 2.5: The system KE, presented as rules of inference in the cut-free Gentzen sequent calculus.

and

$$(q \vee \neg p) \wedge (\neg q \vee p) \tag{2.7}$$

logically equivalent? Proof your answer with the help of SAKE. (Solution.)

(c) Give an upper bound for the k-hardness of the above mentioned equivalence. (Solution.)

4. Consider the following system (Hilbert, 1862-1943):

Axioms (or, rather, axiom schema's)

- $\phi \supset (\psi \supset \phi)$
- $(\phi \supset (\psi \supset \chi)) \supset ((\phi \supset \psi) \supset (\phi \supset \chi))$
- $(\neg\phi \supset \neg\psi) \supset (\psi \supset \phi)$

where ϕ, ψ and χ may be any propositional formula;

Rule of inference (or, rather, inference scheme)

$$\frac{\phi \quad \phi \supset \psi}{\psi}\ \text{modus ponens}$$

where ϕ and ψ may be any propositional formula.

It is a well known fact that Hilbert's system is a sound and complete axiomatization of propositional logic. Although Hilbert's system is elegant and compact, it is not suited for automated theorem proving (Exercise 2e on page 27).

(a) Show that the three axioms can be proved in KE. [Hint: only the third axiom requires *DC*. Use $\neg p$ to distinguish between the case in which $\neg p$ and the case in which $\neg(\neg p)$.] (Solution.)

(b) Show that modus ponens can be simulated in KE. (Solution.)

(c) Argue that KE is complete. (Solution.)

5. Prove that every propositional tautology with k distinct proposition letters, can be proven by means of a KE-refutation of which the tree-size is polynomially expressible in the size of a truth-table for that tautology. (Solution.)

2.7 Restricted but complete versions of KE

KE is "wasteful," in the sense that the incorporation of *DC* establishes soundness at a high price. *DC* not only makes it possible to split every sequent, it also gives us the opportunity to insert spurious formulas at every node in the refutation tree. Linear KE (Exercise 2a), on the other hand is "economical," in the sense that it does not allow multiple reductions at one node, albeit at the price of being incomplete:

	Gentzen	*KE*	*linear KE*
Nr. of applicable rules (per sequent)	1	≥ 1	≤ 1
Sound	yes	yes	no
Complete	yes	yes	yes
Computationally tractible	no	this section	yes

(If you expected that linear KE is sound and incomplete (rather than unsound and complete), then see Problem 2 on page 39.)

The challenge is to control the power of *DC* in such a way that we obtain a system in which the number of possible reductions (per sequent) is minimized, without sacrificing soundness.

Let us first look at the *timing* of *DC*. One thing we could do is to defer its application as long as possible, until all linear rules have been applied in all possible ways. Putting off *DC* makes sense, since this make refutation trees fork at the latest possible moment.

After having applied all linear rules in all possible ways, there comes a point in which no linear rule is applicable. The only possibility—within KE—then, is to split a branch with *DC*. To control the effects of such an application of *DC*, it is wise to impose a number of restrictions on the *parametrization* of this rule, i.e., the formulas that we use to distinguish cases. To see which formulas are appropriate here, let us see for what possible reasons linear refutation may get stuck. One possible reason is when there is a disjunction $p \vee q$ on the LHS of a sequent, while none of its constituents (p or q) is on the RHS of the sequent. Another possible reason is when there is a conjunction on the RHS of sequent, while none of its constituents is on the LHS. Thus, possible causes are the inapplicability of either one of right$_1$-$\wedge$, right$_2$-$\wedge$, left$_1$-$\vee$, left$_2$-$\vee$, left$_1$-$\supset$, or left$_2$-$\supset$. For historical reasons, these rules are called *β-rules*, and the sequents which they operate on are called *β-sequents*. As opposed to other rules, β-rules cannot always be applied, because they sometimes miss the minor sub-formula (a constituent) at the other side of the sequent. So a solution is to resolve 'locked' β-sequents by introducing the missing sub-formulas with *DC*. This is always possible. The price we will have to pay for using *DC* is that it that we will have to create an additional branch for the negation of the minor sub-formula. In this way, all rules in KE are always applicable. Moreover, this limited use of *DC* would bring KE (again) within the realm of analytic systems. The resulting system is called a *strongly analytic* version of KE, abbreviated as *SAKE*. The adjective "strongly" is used because it is more restricted than KE.

Is the strongly analytic version of KE, SAKE, still complete? Fortunately, it is. To prove this, we use the notion of *saturated branch*. Informally, saturation means that all rules are applied in all possible and permitted ways.

Definition 2.2 (Saturation) 1. An open branch is *linearly saturated* if all linear KE-rules have been applied in all possible ways to every element of that branch.

2. An open branch is DC_β*-saturated* if DC is applied in all possible and permitted ways to every β-sequent on that branch. (The notion of β-sequent is explained above.)

3. An open branch is *saturated* if it is linearly saturated and DC_β-saturated.

Thus, in the beginning, every branch is open and may eventually become closed or saturated. Put differently: if a branch does not close it automatically becomes saturated in the long run. (Of course, the "automatically" has to be proven.)

Definition 2.3 (Hintikka set) A set of sequents S is *downwards saturated* if

1.	$\ldots, p \wedge q, \ldots \circ \ldots \in S$	implies	$\ldots, p, q, \ldots \circ \ldots \in S$	
2.	$\ldots \circ \ldots, p \vee q, \ldots \in S$	implies	$\ldots \circ \ldots, p, q, \ldots \in S$	
3.	$\ldots \circ \ldots, p \supset q, \ldots \in S$	implies	$\ldots, p, \ldots \circ \ldots, q, \ldots \in S$	
4.	$\ldots, \neg p, \ldots \circ \ldots \in S$	implies	$\ldots \circ \ldots, p, \ldots \in S$	
5.	$\ldots \circ \ldots, \neg p, \ldots \in S$	implies	$\ldots, p, \ldots \circ \ldots \in S$	
6.	$\ldots \circ \ldots, p \wedge q, \ldots \in S$	implies	$\ldots \circ \ldots, p, \ldots \in S$	or $\ldots \circ \ldots, q, \ldots \in S$
7.	$\ldots, p \vee q, \ldots \circ \ldots \in S$	implies	$\ldots, p, \ldots \circ \ldots \in S$	or $\ldots, q, \ldots \circ \ldots \in S$
8.	$\ldots, p \supset q, \ldots \circ \ldots \in S$	implies	$\ldots \circ \ldots, p, \ldots \in S$	or $\ldots, q, \ldots \circ \ldots \in S$

A set of sequents is *consistent*, or *conflict-free*, if does not have two elements $\ldots \circ \ldots, p \ldots$ and $\ldots, p, \ldots \circ \ldots$, for some formula p.

A *Hintikka set* is a downwards-saturated and conflict-free set of sequences.

With expressions such as "$\ldots, p \wedge q, \ldots \circ \ldots \in S$ implies $\ldots, p, q, \ldots \circ \ldots \in S$," the dots are not entirely arbitrary. To eliminate all ambiguity, we actually should have written something like

"$\phi_1, \ldots \phi_m, p \wedge q, \psi_1, \ldots \psi_n \circ \chi_1, \ldots \chi_k \in S$ implies $\phi_1, \ldots \phi_m, p, q, \psi_1, \ldots \psi_n \circ \chi_1, \ldots \chi_k \in S$".

I have not done this for the sake of brevity.

It is relatively easy to prove that the set of sequents on a saturated branch is downwards saturated. (Exercise 2 on the next page). Implications (1-5) easily follow from the fact that a saturated branch is in particular linearly saturated, while implications (6-8) follow from the fact that a saturated branch is in particular DC_β-saturated (Exercise 1 on the facing page).

Further, it is a well-known result of propositional logic (and of first-order logic) that every downwards saturated set of falsifiable sequents determines a model m that falsifies *all* sequents of that set. This result is known as *Hintikka's Lemma*. In particular the top-sequent is falsified by m, so that the top-sequent is invalid. This argument establishes the completeness of the strongly analytic version of KE. See Exercise 3 on the next page.

Heuristics within SAKE

Although SAKE is a restricted version of KE, it still permits a number of choices during a refutation. Thus, SAKE is *non-deterministic*. In particular, at points at which DC_β may be applied, there are many formulas that one may choose to distinguish cases. [For example, if linear rules do not apply on $\ldots \circ \ldots, p \wedge q, \ldots$, we may choose between $DC_\beta(p)$ and $DC_\beta(q)$.] Therefore, an important question that must be answered before SAKE can be implemented is: which sub-formulas do we use for DC_β? One suggestion is to select the least complex sub-formula of the most complex formula. This approach was taken by Jeremey Pitt and Jim Cunningham [84]. Another suggestion, made by D'Agostino, is to select the sub-formula that appears most often in the formulas on the branch [20]. However, the question precisely which sub-formula should be used has no definite answer and is currently subject to research.

PROBLEMS (Sec. 2.7)

1. Prove that Def. 2.3 (6-8) follows from the fact that a saturated branch is in particular DC_β-saturated. (Solution.)
2. Prove that the set of sequents on a saturated branch is a Hintikka set. (Solution.)
3. (Hintikka's lemma.) Prove that every Hintikka set of sequents possesses a countermodel that falsifies of its elements simultaneously. (Hint: consider axiomatic sequents first, and then use Definition 2.3 to prove that other sequents are falsified by this model as well.) (Solution.)
4. Argue that (2) and (3) together imply that the root of a saturated branch is falsifiable. (Solution.)

2.8 Polynomial fragments of SAKE

Let

$$\text{SAKE}_k =_{Def} \quad \text{the system obtained from SAKE by allowing at most } k \text{ } \textit{nested} \text{ applications of } DC.$$

In this section, we will show that, for every fixed k, SAKE_k has a polynomial time decision procedure. This result is encouraging, because experience has shown that formulas from industrial problem-domains, such as Correct VSLI Design and railway interlocking problems, tend to be decidable in SAKE_k, for $k \leq 1$. Thus, practical experience has shown that most formulas of a specific problem-domain can either be proven or refuted with KE-trees with at most two branches.

To prove that SAKE_k has a polynomial time decision procedure, we further elaborate on the notion of saturation given in Definition 2.2 on page 41. First, it is easy to see that branches can be linearly saturated in linear (!) time, since SAKE possess the subformula-property, and from Exercise 6 on page 29 we know that the number of subformulas of a formula is linearly related to the length of the original formula.

Thus, the complexity of a formula entirely depends on the number of DC-branchings which are required to complete the refutation tree.

Theorem 2.4 *For every fixed k, SAKE_k has a polynomial time decision procedure.*

Proof. Let $S =$ "$\phi_1, \ldots, \phi_l \circ \psi_1, \ldots, \psi_m$" be the initial sequent and let n be the length of S. We will have to prove that all SAKE_k-trees that arise out of S can be traversed in a time that polynomially depends on n. To this end, we observe that the total number of DC_β-applications in a SAKE_k-tree is at most 2^{k-1}, because the number of nested DC_β-applications in a SAKE_k-tree is at most k. Since k is a constant, so is 2^{k-1}, so let $c = 2^{k-1}$ be a new constant. If we write S as $(\phi_1 \wedge (\phi_2 \wedge (\ldots \wedge \phi_l) \ldots)) \supset (\psi_1 \vee (\psi_2 \vee (\ldots \vee \psi_m)))$ then, according to Exercise 6 on page 29, S has $3n$ subformulas. Since subformulas may occur more than once in a formula, there are at most $3n$ distinct subformulas of S and, therefore, at most $(3n)^c$ possible arrangements of DC. Since $(3n)^c$ polynomially depends on n, the desired result follows. □

Definition 2.5 A formula is called *k-hard* if it can be proven in SAKE_k, but not in SAKE_{k-1}.

For example, all formulas that can be proven without DC, are 0-hard.

In practice, it turns out that real industrial verification problems often correspond to formulas with hardness degree ≤ 1. Why is this? One reason could be that, within KE and its

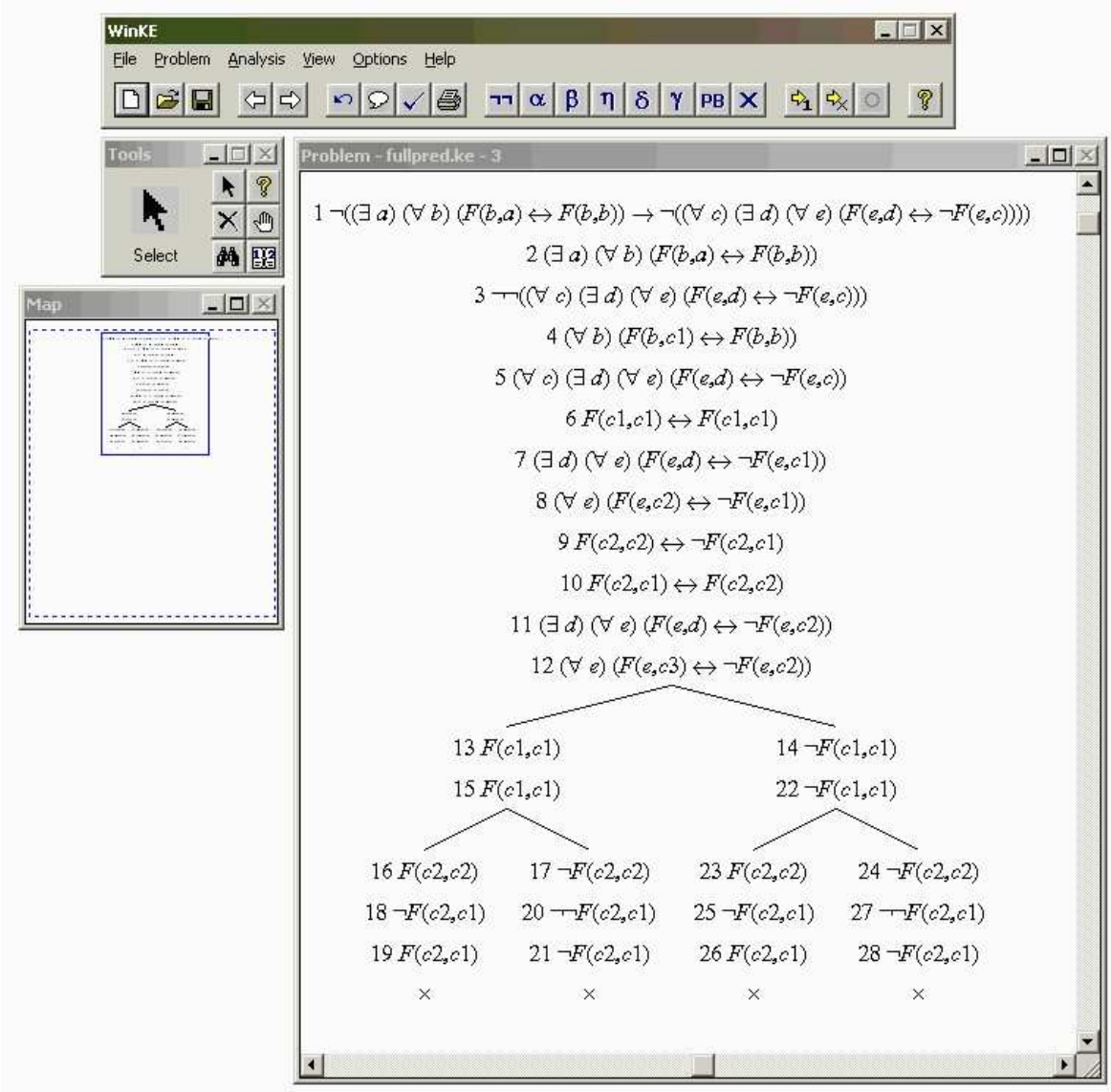


Figure 2.7: Screenshot of WinKE proof assistant and theorem prover.

specializations (SAKE and SAKE$_k$), branching is only done when necessary. Another reason could be the observation that most industrial verification systems can be be grasped by one person, in the sense that one person has designed the system, or at least can understand and formally argue for the correctness of that system. Such systems tend to correspond to easy formulas. In fact, on page 48 we will show that formulas that do not stem from practical problem domains but are generated artificially, tend to be hard. Similarly, systems in which components are assembled in a relatively uncontrolled way tend to correspond to hard formulas. But since most systems in real-life are assembled in a relatively organized way, they give rise to easy formulas.

Groote [38] and Sheeran *et al.* claim that almost all propositional formulas we encounter in practice are 0-hard (49). Also D'Agostino *et al.* [20] claim that quite large fragments of classical logic can be covered with a very limited number of applications of DC_β. They claim that, even with no applications of DC_β, most "textbook examples" can be proved.

PROBLEMS (Sec. 2.8)

1. WinKE is a proof assistant and theorem prover in the KE-calculus, written by Ulrich Endriss, King's College, London. (See Figure 2.7 on the facing page for a screenshot.)
   (a) Install WinKE. (Cf. Appendix A on page 253.)
   (b) Make yourself acquainted with it. (I.e., play with it, read the help, try to open an exercise set, try the exercises.)[5]
   (c) WinKE allows you to choose between one of three modes, which offer different levels of user support during rule applications, reachable in Options menu as Mode. Make yourself acquainted with the different modes, and make sure what they do. (I.e,. in Help menu, click Options, then Mode.)
   (d) Select pedagogue mode. Open in the File menu the set of exercises in propositional logic, called `PROPOSIT.KE`, and try to complete them.
   Note: it is always possible to let WinKE continue a refutation by selecting Analysis → Finish branch or by selecting Problem → Prove.
2. Below you will find ten of the first seventeen problems set by Pelletier for testing automatic theorem provers [81]. Each is a theorem of propositional logic. Give SAKE proofs of them.
   (a) $(p \supset q) \equiv (\neg q \supset \neg p)$
   (b) $\neg(p \supset q) \supset (q \supset p)$
   (c) $((p \vee q) \supset (p \vee r)) \supset (p \vee (q \supset r))$
   (d) $((p \vee q) \wedge (\neg p \vee q) \wedge (p \vee \neg q)) \supset \neg(\neg p \vee \neg q)$
   (e) $(q \supset r) \wedge (r \supset (p \wedge q)) \wedge (p \supset (q \vee r)) \supset (p \equiv q)$
   (f) $((p \equiv q) \equiv r) \equiv (p \equiv (q \equiv r))$
   (g) $(p \vee (q \wedge r)) \equiv ((p \vee q) \wedge (p \vee r))$
   (h) $(p \equiv q) \equiv ((q \vee \neg p) \wedge (\neg q \vee p))$
   (i) $(p \supset q) \equiv (\neg p \vee q)$
   (j) $((p \wedge (q \supset r)) \supset s) \equiv ((\neg p \vee q \vee s) \wedge (\neg p \vee \neg r \vee s))$
3. Find the hardness degree of the following formulas.
   (a) $(p \wedge (q \vee r)) \supset ((p \wedge q) \vee (p \wedge r))$
   (b) $p \supset (q \supset p)$
   (c) $\neg(p \wedge q) \supset (\neg p \vee \neg q)$
4. Write an algorithm that implements SAKE_k.

2.9 Benchmarks

Once you have built a theorem prover, you probably want to test it. You even might want to compare it with other implementations. Below, we will discuss completeness, correctness and performance.

[5] Unfortunately there is a bug in WinKE that allows you to select formulas P and $\neg P$ on two different branches, and close them as if they were on one branch. (They must be closed in succession.) With this technique, it is possible to prove $\vdash P \wedge \neg P$ (Fig. 2.8 on the following page). There is no tickbox or similar setting in the options menu that forbids this maneuver. The bug is known to the creator of WinKE but is left as-is until a next possible release.

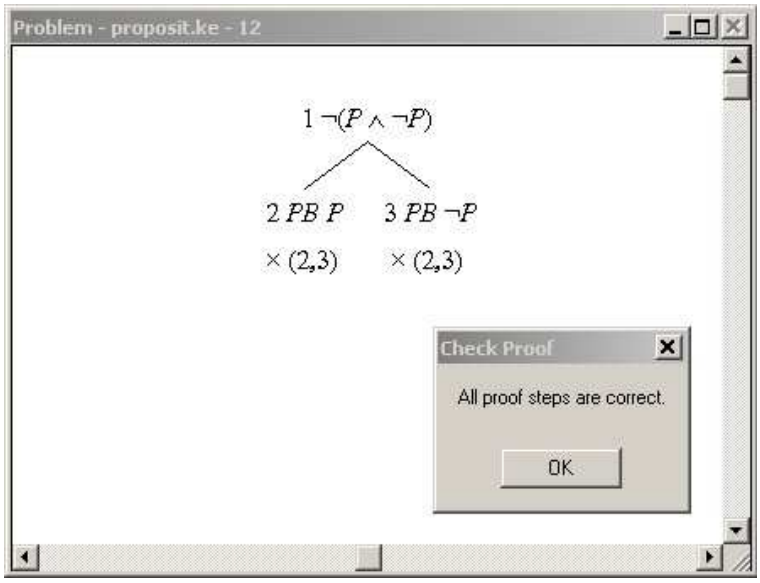


Figure 2.8: A bug in WinKE.

A theorem prover is *complete* if, in theory, every tautology has a proof in the format of the theorem prover. For propositional and first-order theorem provers, completeness is a minimal requirement. Completeness is not a matter of course: many theorem provers trade completeness for speed and efficiency, which is usually a good idea. (This trade is comparable to compromising on luggage for a world tour: being fully equipped for an exhaustive world tour gets you nowhere because you will collapse under your own baggage.) Further, there are even logics that trade completeness for decidability, (The class of *description logics* is such an example. These logics are decidable at the expense of completeness.)

Complete theorem provers are *not* complete in the sense that they are able to disprove all non-tautologies. For example, most first-theorem provers are complete but, due to the undecidability of first-order logic, none of them is able to to disprove all non-tautologies. On the other hand, all "serious" propositional theorem provers are complete both on tautologies as well as non-tautologies. Completeness is an aspect of your system that cannot be tested, but must follow from the design of you algorithm.

The first thing that you do once you have built a theorem prover, is to verify whether the answers are *correct*. Theoretical computer scientists demand that you actually prove the correctness of your program. But, as we all know, the reality is different. Often, you convince yourself of the correctness of the program by a careful inspection of the code. After you stared at the code long enough, you test it with a handful of examples. The combination of careful inspection and empirical testing should more or less convince you that the program works as expected. (Theoretical computer scientists who claim otherwise, have a problem.)
Correctness is equivalent to soundness: if you program halts with an answer, that answer must be correct. For tableaux-based theorem provers, for instance, correctness amounts to verifying that a tableaux closes *only* for tautologies. Correctness does *not* say that that all tautologies can be proved by means of closed tableaux. (This is completeness.) If a theorem prover is unable to give a correct answer, it should say so. In that case it is still correct.

A theorem prover that returns the right answers but takes ample time to produce them, is useless. So once you are convinced that your theorem prover returns the right answers, performance is an issue. Performance amounts to the time and memory a your program uses to prove or disprove formulas. Completeness aside, *theorem provers differ only on performance*

aspects.

Pelletier's problem set

A famous collection of problems that is even nowadays used to compare the performance of theorem provers, is *Pelletier's problem-set* (1986). This set consists of seventy-five problems and was proposed by Francis Pelletier (University of Alberta) [81].

One of the nice features of Pelletier's problem set is that it starts off gradually with a number of technically simple but historically important theorems. Pelletier: "Most (but not all) of my problems can be found in elementary logic textbooks—but they have been chosen either because logic students find them difficult or because they have some interesting connection with other areas of mathematics (such as set theory). The judgements of difficulty are my own, and are based primarily on my years of teaching elementary logic (so the judgments reflect how difficult beginning students find the problems). The scales are from 1 (easiest) to 10, and are relativized to the kind of problem involved. Thus; a '9' in the propositional logic section might in fact be easier than a '4' in the monadic logic section."

The list starts with seventeen problems in propositional logic:

1. $(p \supset q) \equiv (\neg q \supset \neg p)$ (2pts)

   A biconditional version of the 'most difficult' theorem proved by the original Logic Theorist of Newell *et al.*, 1957 [70].

2. $\neg\neg p \equiv p$ (2pts)

   A biconditional version of the 'most difficult' theorem proved by the new logic theorist of Newell *et al.*, 1972 [71].

3. $\neg(p \supset q) \supset (q \supset p)$ (1pt)

   The 'hardest' theorem proved by a breadth-first logic theorist; Siklóssy *et al.*, 1973 [103].

4. $(\neg p \supset q) \equiv (\neg q \supset p)$ (2pts)

   Judged by Siklóssy *et al.* [103] to be the 'hardest' of first 52 theorems of Whitehead and Russell's *Principia Mathematica* 1910, [128].

5. $((p \vee q) \supset (p \vee r)) \supset (p \vee (q \supset r))$ (4pts)

   Judged by Siklóssy *et al.* [103] to be the 'hardest' of first 67 theorems of Whitehead and Russell's *Principia Mathematica* 1910, [128].

6. $p \vee (\neg p)$ (2pts)

   The Law of Excluded Middle: can be quite difficult for systems such as natural deduction.

7. $p \vee (\neg(\neg(\neg p)))$ (3pts)

   Expanded Law of Excluded Middle. The strategies of the original Logic Theorist cannot prove this.

8. $((p \supset q) \supset p) \supset p$ (5pts)

   Pierce's Law. Unprovable by Logic Theorist, and tricky for systems such as natural deduction.

9. $[(p \vee q) \wedge (\neg p \vee q) \wedge (p \vee \neg q)] \supset \neg(\neg p \vee \neg q)$ (6pts)

   A problem not solvable by unit resolution nor by 'pure' input resolution.

10. $q \supset r, r \supset (p \wedge q), p \supset (q \vee r) \circ p \equiv q$ (4pts)

    A reasonable simple problem with premises, designed to see whether systems like natural deduction correctly manipulate premises.

11. $p \equiv p$ (1pt)

    A simple problem designed to see whether systems like natural dduction can do it efficiently, or whether they incorrectly try to prove the $\supset$ each way.

12. $[(p \equiv q) \equiv r] \equiv [p \equiv (q \equiv r)]$ (7pts)

    The 'hardest' propositional problem found in Kalish and Montague (1964), according to Peletier (1982)[80].

Distribution Laws can be very tricky for systems such as natural deduction. (They are *assumed* by resolution systems in the conversion to clause form.) The next few problems list some.

13. $[p \vee (q \wedge r)] \equiv [(p \vee q) \wedge (p \vee r)]$ (5pts)

14. $(p \equiv q) \equiv ((q \vee \neg p) \wedge (\neg q \vee p))$ (6pts)

15. $(p \supset q) \equiv (\neg p \vee q)$ (5pts)

16. $p \supset q) \vee (q \supset p)$ (4pts)

    A surprising theorem of propositional logic.

17. $((p \wedge (q \supset r)) \supset s) \equiv ((\neg p \vee q \vee s) \wedge (\neg p \vee \neg r \vee s))$ (6pts)

    A problem which appears not to be provable by Bledsoe *et al.* (1972). For details of why not, see Pelletier (1982) [80].

Other hard problems not mentioned in Pelletier's problem list include (2.3) on page 35 and (2.4) on page 36.

Artificial problems

An artificial problem is a problem that is generated by a computer program, and does not necessarily correspond to a practical problem.

One way to produce artificial problems, proposed by Groote *et al.* [38], is by proving or disproving that certain numbers are prime. This approach has a few advantages over other formula generating methods. Firstly, it is easy to generate arbitrarily large formulas that are either satisfiable or contradictory. Moreover, it can easily be predicted whether a formula is satisfiable or not; the numbers are so small that it is easy to check whether the numbers are prime using conventional means.

The idea is as follows. We assume that numbers are represented in binary notation. For instance, if $n = 11$ (in decimal notation), then $n = 8 + 2 + 1$, so that $n = 1011$ in binary notation. Numbers are also written as vectors, or arrays: $\bar{n} = (n_3, n_2, n_1, n_0) = (1, 0, 1, 1)$. A *factorization* of a number n consists of two integers p and q such that $n = pq$ and $p, q > 1$. A number n is *prime* if and only if it has no factorization.

Just as with decimal multiplication, numbers can be multiplied. The multiplication is given by introducing bits $s_{i,j}$ for *intermediate* results and *intermediate* carries $c_{i,j}$. All numbers < 7, for instance, can be multiplied according to Table 2.6 on the next page (left). Thus, per column $i + j$, the intermediate sum $s_{i+j,j}$ is determined by the previous sum $s_{i+j,j-1}$, the product $p_i q_j$, and whether there is a carry $c_{i+j-1,j-1}$ or not. The same holds for $c_{i+j,j}$. (If you feel like you are

			p_2	p_1	p_0
$\times$			q_2	q_1	q_0
			$s_{2,0}$	$s_{1,0}$	$s_{0,0}$
$+$		$s_{3,1}$	$s_{2,1}$	$s_{1,1}$	0
$+$	$s_{4,2}$	$s_{3,2}$	$s_{2,2}$	0	0

sum	*product*	*carry*	*new sum*	*new carry*
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
0	0	1	1	0
1	1	0	0	1
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

Table 2.6: Binary multiplication. (Arrow represents $c_{1,0}$.)

juggling with indices, it helps to do a plain binary multiplication yourself. Compute, e.g. 1011×100.) Looking at Table 2.6 (right), we see that

$$s_{i+j,j} \equiv (c_{i+j-1,j-1} \equiv (s_{i+j,j-1} \equiv (p_i \wedge q_j))) \tag{2.8}$$

$$c_{i+j,j} \equiv ((c_{i+j-1,j-1} \wedge s_{i+j-1,j-1}) \vee (c_{i+j-1,j-1} \wedge p_i \wedge q_j) \vee (s_{i+j,j-1} \wedge q_j)) \tag{2.9}$$

At the boundaries of the multiplication, these formulas are simpler, for instance because carries are known to be 0. The condition that $p, q > 1$ may be expressed as the conjunction of $\bar{p} \neq (0, \ldots, 0, 1)$ and $\bar{q} \neq (0, \ldots, 0, 1)$, i.e.,

$$(\neg p_0 \vee p_1 \vee p_2 \vee \ldots) \wedge (\neg q_0 \vee q_1 \vee q_2 \vee \ldots) \tag{2.10}$$

Further, the last row of $s_{i,j}$'s, i.e., the outcome of the multiplication is set to $\bar{n}$. If the conjunction of (2.8), (2.9) and (2.10) is satisfiable, a binary representation of the dividers is found in $p_k, p_{n-1} \ldots p_0$ and $q_k, q_{m-1} \ldots q_0$.

We may now write a program that reads a natural number and generates a propositional formula that is satisfiable if and only if its input *not* prime.

Example 2.6 For $n = 4$ the prime formula looks like

```
(s0_0 <-> (p0 & q0)) & ~c0_0 & (s1_0 <-> (p1 & q0)) & ~c1_0 & (s2_0
<-> (p2 & q0)) & ~c2_0 & ~c2_0 & (s1_1 <-> (c0_0 <-> (s1_0 <-> ((p0
& q1))))) & (c1_1 <-> ((c0_0 & s1_0) | (c0_0 & p0 & q1) | (s1_0 & p0
& q1))) & (s2_1 <-> (c1_0 <-> (s2_0 <-> ((p1 & q1))))) & (c2_1 <->
((c1_0 & s2_0) | (c1_0 & p1 & q1) | (s2_0 & p1 & q1))) & ~(p2 & q1)
& ~c2_1 & (s2_2 <-> (c1_1 <-> (s2_1 <-> ((p0 & q2))))) & (c2_2 <->
((c1_1 & s2_1) | (c1_1 & p0 & q2) | (s2_1 & p0 & q2))) & ~(p1 & q2)
& ~(p2 & q2) & ~c2_2 & ~s0_0 & ~s1_1 & s2_2 & ~q2 & ~p2
```

Since 4 is prime, the formula is satisfiable. A satisfying model is $\bar{p} = \bar{q} = \bar{2}$, i.e., $(p_1, p_0) = (q_1, q_0) = (1, 0)$.

It seems that artificial problems are harder than the ones that are taken form practical problem domains. Groote *et al.* [38] have shown that (formulas that correspond to) 112, 4711, 655381, 3476734, and 58697733 are 0-hard, 113 and 257 are 1-hard, 47161, 655379 and 3476734 are 2-hard, and that 58697731 is 3-hard.[6]

Randomly generated problems

A sub-class of the class of artificially generated problems, is formed by the class of *randomly generated problems*. An example of a class of randomly generated problems is Uniform

[6] For a definition of k-hardness, see on page 43.

Random-3SAT [45]. Uniform Random-3SAT is obtained by randomly generating 3CNF formulae in the following way: for a CNF with n variables and k disjunctions, each of the k disjunctions is constructed from three literals which are randomly drawn from the $2n$ possible literals (the n variables and their negations) such that each possible literal is selected with the same probability of $1/2n$. A disjunction is not accepted if it contains multiple copies of the same literal or if it contains a variable and its negation. Each choice of n and k thus induces a distribution of Random-3SAT instances. Uniform Random-3SAT is the union of these distributions over all n and k.

One particularly interesting property of uniform Random-3SAT is the occurrence of a phase transition phenomenon, i.e., a rapid change in solubility which can be observed when systematically increasing (or decreasing) the number k of disjunctions for fixed problem size n. The occurrence of a phase transition can be understood as follows. For small k, almost all formulae are underconstrained and therefore satisfiable; when reaching some critical k the probability of generating a satisfiable instances drops sharply to almost zero. Beyond k, almost all instances are overconstrained and thus unsatisfiable. For Random-3SAT , this phase transition occurs approximately at $k = 4.26n$ for large n; for smaller n, the critical clauses/variable ratio k/n is slightly higher. Further, for large n the transition is more abrupt.

Contemporary problem sets

Propositional formula checking has made an enormous progress. Accordingly, problem sets became larger and better organized. Some well known examples are TSPLIB (containing a variety of TSP and TSP-related instances), ORLIB (providing test instances for a variety of problems from Operations Research), TPTP (a collection of problem instances for theorem provers), and, more recently, CSPLIB (a benchmark library for constraints). SAT-oriented problems can be found in the benchmark collection from the Second DIMACS Challenge [119], the collection from the Beijing SAT Competition, and, lately, the SATLIB benchmark suite [45].

Most benchmark suites consist of three different types of problems: individual instances from various domains, such as the Pelletier problem list, test-sets sampled from random instance distributions, such as uniform Random-3SAT, and test-sets obtained by encoding instances from random instance distribution of SAT-encoded combinatorial problems such as Graph Colouring and planning problems; The SATLIB benchmark suite contains (or rather refers to) the DIMACS suite [119]. The DIMACS suite contains instances stemming from practical applications, as well as constructed hard instances, all in CNF. Most benchmark suites are available online and provide not just a bench- mark library, but rather an increasingly comprehensive resource for research including implementations, results from various algorithms, a list of people involved in research on propositional theorem proving, pointers to related web sites, events (such as workshops and conferences), and an annotated bibliography.

PROBLEMS (Sec. 2.9)

1. Someone claims that his (or her) theorem prover solves *all* seventy-five problems of the Pelletier problem list within 1 millisecond. What does that tell you? (Solution.)
2. Consider the method of generating prime formulas.
   (a) Verify (2.8), (2.9) and (2.10) on the preceding page.
   (b) Determine the prime formula of 2. This formula has thirteen conjuncts. (Solution.)
   (c) Write a program to produce prime formulas.
3. (From SAT review, at DIMACS.) Propositional formula checking is reputed to have many applications, but there is only a limited number of real instances available. Expert systems and other logic machines should be generating many, many instances.

(a) Do these instances "look" different from random problems?

(b) Are such instances easier or harder to solve in practice?

(c) Can aspects of such instances be embedded in random generators to allow for generating larger random instances?

(d) Are there applications that "typically" generate very hard instances?

Note: these are hard problems. Not to say that these are research problems.

Chapter 3

First-order theorem proving

This chapter assumes familiarity with the syntax and semantics of first-order logics. In particular, knowledge of the following concepts is assumed:

- well-formed formulas,
- scope of a quantifier,
- free and bound variables,
- closed well-formed formulas (sentences),
- fair substitutions,
- first-order domains,
- interpretation of constants, function symbols and predicate symbols,
- interpretation of well-formed formulas,
- variable assignments,
- first-order models,
- first-order countermodels

If you are not familiar with these concepts (or have forgotten them), you may wish to consult one of the many introductions that exist on first-order logic [17, 67, 29, 7]. Exercises 3-8 on the next page are meant to help to brush up your knowledge of the syntax and semantics of first-order logic.

3.1 Brief rehearsal

For completeness' sake, we recapitulate a number of important facts.

1. *Predicate logic.* Predicate logic is first-order logic without further restrictions on the semantics of the formulas. Thus, predicate logic is the most minimal form of first-order logic. Other first-order logics are designed to model specific structures, such as axioms for equality, arithmetic, Boolean algebras, binary trees, and stacks (to name a few typical examples).
2. *Completeness of predicate logic.* There exists an axiomatization of predicate logic (Gödel, 1930). Thus, there exists a collection of axioms and syntactical proof rules, with the help of which precisely all semantically valid formulas of predicate logic can be proven. A proof of this fact is of medium difficulty [67, 106, 29, 27]. An axiomatization of predicate logic is called a *predicate calculus*. There are different predicate calculi.
3. *Church's thesis.* Our (intuitive) notion of computability corresponds to computability on a whole class of provably equivalent symbol manipulating mechanisms. Elements of this class include the Turing machine (Turing, 1936), the Von Neumann register machine (Von Neumann, 1946), the λ-calculus (Rosser and others, 1943), the concept of recursive functions (Kleene and others, 1936), canonical systems (Post, 1943), and normal algorithms (Markov and Nagorny, 1988).
   Church's thesis is a proposal to connect the term algorithm to this class of symbol manipulating mechanisms.

4. *Undecidability of predicate logic.* (Church, 1936). There exists no Turing machine that computes for every sentence in the predicate calculus whether it is valid or invalid. A proof of this fact is involved, because we have to show that, for example, no Turing machine is able to stop on every first-order formula with a correct answer.
   If we accept Church's thesis, we may say that there exists no algorithm that computes for every sentence in the predicate calculus whether it is valid or invalid.
5. *Semi-decidability of predicate logic.* There exist algorithms that prove precisely all formulas that are valid in the predicate logic. This fact is the basis of ATP, and will be made plausible in this chapter. (A rigorous mathematical proof falls beyond the scope of this book, but see [27].)
   If $\mathcal{A}$ is an algorithm proving precisely all valid formulas, and ϕ is valid, then $\mathcal{A}$ will prove ϕ. If $\mathcal{A}$ is given an invalid formula, however, then, by the undecidability of predicate logic, no one can guarantee that $\mathcal{A}$ will ever disprove ϕ. Thus, as long as $\mathcal{A}$ is running and keeps on running, we do not know whether it is busy with a proof of ϕ, a refutation of ϕ, or with a failing attempt to refute of ϕ. If we hit `<ctrl-C>`, we never know whether $\mathcal{A}$ would have answered within the next minute, had `<ctrl-C>` not been pressed.
6. *Incompleteness of arithmetic.* Any first-order logic that is expressive as arithmetic cannot be axiomatized (Gödel, 1931). In other words, if we extend the predicate calculus with an axiomatization of arithmetic, there are valid but unprovable formulas. A proof of this fact is rather difficult.

The adjective "first-order" comes from the fact that, in a first-order language, quantifiers range over variables. By contrast, a second-order language allows one to quantify over predicate variables. An example of a second-order formula is

$$(\forall x)[(\forall y)((\forall z)(x(y,z) \supset \neg x(z,y))) \supset (\forall y)(\neg x(y,y))] \tag{3.1}$$

(Exercise 1). There are several reasons why second and higher-order logics are more difficult to manage. Firstly, most higher-order theories can be embedded in, or translated into, first-order theories [67]. Another reason is that first-order logic suffices for the expression of most mathematical theories. A final, and perhaps more practical reason to postpone the treatment of higher-order logics is that dealing with first-order languages is complicated enough. Indeed, most efforts of the automated reasoning community have gone into the proof theory of first-order logic.[1]

PROBLEMS (Sec. 3.1)

1. Give a possible interpretation of Formula 3.1. What could it mean? Are there interpretations for which this formula could be wrong? (Solution.)

2. What is a formula? What is a well-formed formula (WFF)? What is a sentence? (Solution.)

3. Which of the following expressions can reasonably be exchanged between two persons without danger of misinterpretation? Which of them can be feed to a theorem prover without parse errors or danger of misinterpretation? Which formulas are sentences?

   (a) $(\forall x)(Px, y)$
   (b) $(\forall x)((\forall y)(Px, y))$
   (c) $(\forall x)(\forall y)(Px, y)$
   (d) $(\forall x, \forall y)(Px, y)$

[1] This is not to the deny the important work that has been done on higher-order logic (HOL) and HOL ATP techniques. I just wanted to indicate that there is already a lot to do in the field of first-order ATP.

(e) $(\forall x, y)(Px, y))$

(f) $(\forall z)(Pf(x), a)$

(Solution.)

4. A variable x is *bound* if it occurs in the scope of an $(\forall x)$ or $(\exists x)$ quantifier. E.g., $(\forall x)(\ldots x \ldots)$. List the free and bound variables of the following formulas.

   (a) Px, y

   (b) $(\forall x)(Px, y)$

   (c) $(\forall y)(Px, y)$

   (d) $(\forall x)(Py) \wedge (\forall y)(Px, y)$

   (Solution.)

5. The expression $\phi[t/x]$ represents ϕ, with zero or more free occurrences of the variable x replaced by the term t. Pronounce: "t for x in ϕ". Let

$$\phi = (\forall x)(Px, y \supset (\forall w)(Pw, x \vee (\exists y)(Py, z))).$$

   (a) Determine $\phi[f(b, g(a, w))/x]$. To obtain a unique answer, let us assume that all free occurrences are replaced.

   (b) Determine $\phi[f(b, g(a, w))/y]$. (Same assumption as with previous item.)

   (c) Determine $\phi[f(b, g(a, w))/z]$. (Same assumption.)

   (d) Determine $\phi[f(g(a, d), z)/v]$. (Same assumption.)

   (Solution.)

6. A term t is called *free for x in ϕ*, if we may safely substitute free occurrences of x by t in ϕ. "Safe," here, means that no variable in t will accidentally be bound when $\phi[t/x]$ is formed. Thus, t is free for x in ϕ iff no free occurrence of x in ϕ lies within the scope of quantifiers that have a variables in t.

   Consider $\phi = (\exists x)(Rx, y)$.

   (a) Give a term t_1 that is not free for x in ϕ.

   (b) Give a term t_2 that is not free for y in ϕ.

   (c) Show what goes wrong if one or more free occurrences of y in ϕ are replaced by t_2.

   (Solution.)

7. Let $\phi = (\forall x)(Px, y \supset (\forall w)(Pw, x \vee (\exists y)(Py, z)))$.

   (a) Is $f(b, g(a, w))$ free for x in ϕ?

   (b) Is $f(b, g(a, w))$ free for y in ϕ?

   (c) Is $f(b, g(a, w))$ free for z in ϕ?

   (d) Is $f(g(a, d), z)$ free for v in ϕ?

   (Solution.)

8. Give a model M and a variable assignment s such that the combination (M, s) satisfies each of the following sets of formulas.

(a) The set of formulas

$$(\forall x)(\neg Px, x)$$
$$(\forall x, y, z)((Px, y \wedge Py, z) \supset Px, z)$$
$$(\forall x, y, z)(Px, y \supset \neg Py, x)$$

(Solution.)

(b) Formulas listed at (8a), extended with $(\forall x, y)(Px, y \supset (\exists z)(Px, z \wedge Pz, y))$. (Solution.)

(c) Formulas listed at (8a), extended with $(\forall x)(Px, y)$. (Solution.)

(d) The set of formulas

$$(\forall x, y)(Ef(x, y), f(y, x))$$
$$(\forall x)((Ef(x, a), x)$$
$$(\forall x, y)((\exists y)(Ef(x, y), a)$$
$$(\forall z)((Ef(x, z), z)$$

(Solution.)

9. In a first-order model M with domain D, constants c are interpreted as elements $c_M \in D$, n-ary function symbols f are interpreted as functions $f_M : D^n \to D$, and n-ary predicate symbols P are interpreted as subsets $P_M \subseteq D^n$. (It follows that D must be non-empty.) A *valuation* is a function s that maps every variable to an element in D. If t is a term, then the image of t in D, written t_s is defined inductively as $[f(t_1, \ldots, t_n)]_s =_{Def} [f_M((t_1)_s, \ldots, (t_n)_s)]$, $a_s =_{Def} a_M$, and $x_s =_{Def} s(x)$.

   Prove the so-called *substitution lemma for terms*:

$$(t'[t/x])_s = t'_{s[t_s/x]}$$

   for any two terms t and t'. {Rather than object-oriented, the above equation may also be written in function notation: $s(t'[t/x]) = s[s(t)/x](t')$, or mixed: $s(t'[t/x]) = s[t_s/x](t')$.} Hint: prove with induction on the composition of t'. (Solution.)

10. The so-called *substitution lemma for formulas* says that $(M, s) \models \phi[t/x]$ iff $(M, s[t_s/x]) \models \phi$, provided t is free for x in ϕ.

    (a) Show with the help of an example that the requirement that t is free for x in ϕ, is essential. Hint: compare $(\forall y)(R(x, y))$ and $(\forall y)(R(f(y), y))$. (Solution.)

    (b) Prove the substitution lemma. [If you are able to prove the substitution lemma, then you have shown to have enough understanding of the syntax and semantics of first-order logic to proceed to the next section, so give it a try.] (Solution.)

    The substitution lemma connects syntax and semantics. It is therefore extremely helpful in proving completeness of first-order proof methods, such as the tableaux method and resolution.

3.2 Formula-generators

Table 3.1 on the facing page extends the propositional proof table (Exercise 2.2 on page 28) with rules that describe how quantified formulas can be made true or false.

The idea behind making a universal statement $(\forall x)(P)$ true is that we make all instances true. Not all at once (which is impossible in a step-by-step refutation), but one at a time:

$$\begin{aligned}
(\forall x)(P) &\equiv (\forall x)(P) \wedge P[a/x] && \text{(spawn formula with } t = a\text{)}\\
&\equiv (\forall x)(P) \wedge P[a/x] \wedge P[b/x] && \text{(spawn with } t = b\text{)}\\
&\equiv (\forall x)(P) \wedge P[a/x] \wedge P[b/x] \wedge P[f(a)/x] && \text{(spawn with } t = f(a)\text{)}\\
&\equiv (\forall x)(P) \wedge P[a/x] \wedge P[b/x] \wedge P[f(a)/x] \wedge P[f(b)/x] && \text{(spawn with } t = f(b)\text{)}\\
&\vdots && \vdots
\end{aligned}$$

To this end, we use the fact that a universal sentence is logically equivalent to itself conjoined with an arbitrary instance of that sentence. The acronym GNU is a good example of this principle. The GNU Project was launched in 1984 to develop a complete Unix-like operating system which is free software: the GNU system. GNU means: GNU's Not Unix. So GNU means GNU's Not Unix's Not Unix, and so forth. Thus, the acronym GNU is an inexhaustible generator of an infinite number of sentences. Here is where the analogy ends, by the way, because the difference with "left-$\forall$" is that a $(\forall x)p$ on the LHS may generate a potentially infinite number of *different* terms.

A *ground term* is a term without variables. The set of all possible ground terms that can be made with constants and function symbols that occur in a formula, is called the *Herbrand domain* of that formula. If a term has no constants, then a fresh constant c_0 is used to prevent the Herbrand domain from being empty.

The notion may be generalized to sets of formulas and terms.

Example 3.1 (Herbrand domain)

Set of formulas and terms	*Constants, and function symbols*	*Herbrand domain*
$\{(\forall x)(p\,x, a)\}$	$\{a\}$	$\{a\}$
$\{(\forall x)(pf(x), a)\}$	$\{a, f\}$	$\{a, f(a), f(f(a)), \ldots\}$
$\{(\forall x)(pf(x))\}$	$\{f\}$	$\{c_0, f(c_0), f(f(c_0)), \ldots\}$
$\{(\forall x)(pg(x, y))\}$	$\{g\}$	$\{c_0, g(c_0, c_0), g(g(c_0, c_0), c_0), \ldots\}$
$\{(\forall x)(pg(x, y)),\ qf(c_3)\}$	$\{f, g, c_3\}$	$\{c_3,\ f(c_3),\ g(c_3, c_3),\ g(f(c_3), c_3),$ $g(c_3, f(c_3)),\ f(g(c_3, c_3)),$ $g(g(c_3, c_3), c_3),\ \ldots\}$

Reduction rules for building a refutation tree	
- If $(\forall x)\phi$, then $\phi[t/x]$ for all terms t.	- If $(\forall x)\phi$ is false, then $\phi[c/x]$ is false for some constant c.
- If $(\exists x)\phi$, then $\phi[c/x]$ for some constant c.	- If $(\exists x)\phi$ is false, then $\phi[t/x]$ is false for all terms t.

Table 3.1: Extension of reduction table (Exercise 2.2 on page 28.)

.

The reduction rules of proposition logic have the pleasant property that they reduce the complexity of the formula (Table in Exercise 2.2 on page 28). This is known as the *sub-formula property* (page 29). The rules "left-$\forall$" and "right-$\exists$" lack this property, in the sense that they do not reduce the complexity of the formula they are operating on (Exercise 3 on page 33). Reductions may go on forever so that branches may grow indefinitely, which is inherent to the undecidability of predicate logic. On the other hand, the possibility that reductions go on forever

Rules from Table 2.2 on page 31, plus

$$\begin{array}{ll} (\forall x)\phi \circ & \circ\,(\forall x)\phi \\ \quad\Big|\ \text{left-}\forall & \quad\Big|\ \text{right-}\forall \\ \quad\Big|\ \text{Term } t \text{ free for } x \text{ in } \phi & \quad\Big|\ y \text{ does not occur in the current tree} \\ \phi[t/x] \circ & \circ\,\phi[y/x] \end{array}$$

$$\begin{array}{ll} (\exists x)\phi \circ & \circ\,(\exists x)\phi \\ \quad\Big|\ \text{left-}\exists & \quad\Big|\ \text{right-}\textit{exists} \\ \quad\Big|\ y \text{ does not occur in the current tree} & \quad\Big|\ \text{Term } t \text{ free for } x \text{ in } \phi \\ p[y/x] \circ & \circ\,\phi[t/x] \end{array}$$

Table 3.2: Analytic refutation rules for first-order logic.

Rules from Table 2.3 on page 34, plus

$$\text{left-}\forall: \quad \frac{\ldots, (\forall x)\phi, \phi[t/x], \ldots \vdash \ldots}{\ldots, (\forall x)\phi, \ldots \vdash \ldots} \qquad \text{right-}\forall: \quad \frac{\ldots \vdash \ldots, \phi[c/x], \ldots}{\ldots \vdash \ldots, (\forall x)\phi, \ldots}$$

$$\text{left-}\exists: \quad \frac{\ldots, \phi[c/x], \ldots \vdash \ldots}{\ldots, (\exists x)\phi, \ldots \vdash \ldots} \qquad \text{right-}\exists: \quad \frac{\ldots \vdash \ldots, (\exists x)\phi, \phi[t/x], \ldots}{\ldots \vdash \ldots, (\exists x)\phi, \ldots}$$

Table 3.3: Gentzen system for first-order logic.

is a worst-case scenario. In many cases, we are able to guess which substitutions must be made to steer the refutation to an end.

PROBLEMS (Sec. 3.2)

1. Determine the Herbrand domain of the following formulas.
   (a) Px, a (Solution.)
   (b) $(\forall x)(Px \supset Qf(x), a)$ (Solution.)
   (c) $Rg(f(x), y), z$ (Solution.)

3.3 No functions and no equality

If a sentence contains no function and no equality symbols, its Herbrand domain is a finite but non-empty set of constants. During the refutation process, this set may be extended with fresh constants due to the application of a "left-$\exists$" or "right-$\forall$" rule, but it never grows excessively, since new constants are only introduced by left-$\exists$ or right-$\forall$. Thus, without function and equality symbols, it is no problem to substitute all variables and constants that we have encountered in the refutation thus far.

Example 3.2 Let us begin analyzing a simple case, namely $(\forall x)(p\,x) \vdash pa$. The only way to reduce this formula, is by means of the left-$\forall$ rule. To apply this rule, we will have to decide which term t we are going to substitute for x in $(\forall x)(p\,x)$. Since the Herbrand domain of $(\forall x)(p\,x)$, pa is $\{a\}$, we have not much choice but $t = a$. Substituting a for x in $p\,x$ immediately produces a formula on the LHS that mirrors a sentence pa on the RHS, so we are done.

$$\begin{array}{c} (\forall x)(p\,x) \circ pa \\ \text{left-}\forall \\ \Big| \\ (\forall x)(p\,x); \mathbf{pa} \circ \mathbf{pa} \\ \times \end{array}$$

The universal formula on the left works as a formula generator that spawn ground formulas that may match with formulas on the RHS. *Which* formulas the universal formula produces is determined by the order in which we select terms from the Herbrand domain of that formula.

Example 3.3 Consider $p \vdash (\forall x)(qx)$. The only rule applicable to $p \circ (\forall x)(qx)$ is "right-$\forall$", with a fresh constant c_1:

$$\begin{array}{c} p \circ (\forall x)(qx) \\ \text{right-}\forall \\ \Big| \\ p \circ qc_1 \\ \not\downarrow \end{array}$$

$\mathrm{LHS} \cap \mathrm{RHS} = \varnothing$, so that we have found a counterexample model M with domain $D = \{1\}$, such that c_1 and all other constants are mapped to 1, and

Predicate	*Extension*
p	true
q	$\varnothing$

Notice how in the process of constructing a counterexample the domain grows by demand. First, it is empty, and then a constant is added to realize a counterexample.

We move on to a somewhat more complicated case.

Example 3.4 Consider the (valid) sequent

$$(\forall x)(p\,x) \vdash (\exists x)(p\,x). \tag{3.2}$$

There are two reductions possible: one by means of "left-$\forall$," and one by means of "right-$\exists$". However, to apply either one of these rules, we need to choose a term based on the Herbrand domain of (3.2) to perform a substitution. However, there is no such term, since the Herbrand domain of (3.2) is empty. Therefore, we use an arbitrary constant c_0, say, to kick off the refutation.

$$\begin{array}{c} (\forall x)(p\,x) \circ (\exists x)(p\,x) \\ \text{left-}\forall \\ \Big| \\ (\forall x)(p\,x); pc_0 \circ (\exists x)(p\,x) \\ \text{right-}\exists \\ \Big| \\ (\forall x)(p\,x); \mathbf{pc_0} \circ (\exists x)(p\,x); \mathbf{pc_0} \\ \times \end{array}$$

Note that, in the second step, it is no longer necessary to introduce a new term because c_0 then already exists.

Example 3.5 Here is a refutation of the claim that "exists" implies "for all":

$$(\exists x)(p\,x) \circ (\forall x)(p\,x)$$
left-$\exists$
$$pc_1 \circ (\forall x)(p\,x)$$
right-$\forall$
$$pc_1 \circ pc_2$$
$\lightning$

First, pc_1 is put forward as a witness for $(\exists x)(p\,x)$. Since any term may be substituted for x in $(\forall x)(p\,x)$ on the RHS, we substitute a term different from c_1, viz. c_2, to falsify pc_1 on the LHS.

We end up with a counterexample model M with

Predicate	*Extension*
p	$\{c_1\}$

and domain $D = \{1, 2\}$, such that c_1 and c_2 are interpreted as 1 and 2. Then $M \models p(c_1)$ but $M \not\models p(c_2)$.

Example 3.6 The sequent $(\exists x)(p\,x) \wedge (\exists x)(qx) \vdash (\exists x)(p\,x \wedge qx)$ is invalid. (The existence of pipe-smoking persons, and the existence of babies does not imply the existence of pipe-smoking babies.) Here is a refutation:

$$(\exists x)(p\,x) \wedge (\exists x)(qx) \circ (\exists x)(p\,x \wedge qx)$$
left-$\wedge$
$$(\exists x)(p\,x); (\exists x)(qx) \circ$$
left-$\exists$
$$pc_1 \circ$$
left-$\exists$
$$qc_2 \circ$$
right$^+$-$\exists$ with $\{c_1, c_2\}$
$$qc_2 \circ pc_1 \wedge qc_1; pc_2 \wedge qc2$$
right-$\wedge$

$qc_2 \circ pc_1 \wedge qc_1; pc_2$ right-$\wedge$ → $\mathbf{pc_1}; qc_2 \circ \mathbf{pc_1}; pc_2$ ×; $qc_2 \circ qc_1; pc_2$ $\lightning$

$qc_2 \circ pc_1 \wedge qc_1; qc2$ ⋮

Countermodel:

Predicate	*Extension*
p	$\{c_1\}$
q	$\{c_2\}$

An essential property of this countermodel is that $c_1 \neq c_2$.

Consider:

i. $(\forall x)((\exists y)(p\,x, y))$ Everybody likes a music genre.
ii. $(\exists x)((\forall y)(p\,x, y))$ There is someone who likes all music genres. (Musical omnivore?)
iii. $(\forall y)((\exists x)(p\,x, y))$ Every music genre is liked by somebody. (The creator of that genre?)
iv. $(\exists y)((\forall x)(p\,x, y))$ There is a music genre that is liked by everybody. (Muzak?)

Clearly, (ii) $\rightarrow$ (iii) and (iv) $\rightarrow$ (i), while all other $4 \times 3 - 2 = 10$ implications are false.

Instead of using "left-$\forall$" and "right-$\exists$," let us use the rules displayed in Table 3.4. The rule "left$^+$-$\forall$" means: one or more applications of "left-$\forall$," and similarly for "right$^+$-$\exists$". Both rules do not enable reductions that would otherwise be impossible. Instead, they are meant to reduce the size of the refutation trees (and, hence, the size of the proof trees).

In the proofs that follow we apply all rules of Table 3.4, to all terms that occur in the Herbrand domain of the original formula.

$$\begin{array}{cc} (\forall x)\phi \circ & \circ\,(\exists x)\phi \\ \Big| \ \text{left}^+\text{-}\forall & \Big| \ \text{right}^+\text{-}\exists \\ (\forall x)\phi, \phi[t_1/x], \ldots, \phi[t_n/x] \circ & \circ\,(\exists x)\phi, \phi[t_1/x], \ldots, \phi[t_n/x] \end{array}$$

$$\begin{array}{ll} \text{left}^+\text{-}\forall: & \text{right}^+\text{-}\exists: \\ \dfrac{\ldots, (\forall x)\phi, \ldots, \phi[t_1/x], \ldots, \phi[t_n/x], \ldots \vdash \ldots}{\ldots, (\forall x)\phi, \ldots \vdash \ldots} & \dfrac{\ldots \vdash \ldots, (\exists x)\phi, \ldots, \phi[t_1/x], \ldots, \phi[t_n/x], \ldots}{\ldots \vdash \ldots, (\exists x)\phi, \ldots} \end{array}$$

Table 3.4: Many-on-one variants of left-$\forall$ and right-$\exists$.

Example 3.7 A failed refutation (= proof) of (ii) $\rightarrow$ (iii):

$$\begin{array}{c} (\exists x)((\forall y)(p\,x, y)) \circ (\forall y)((\exists x)(p\,x, y)) \\ \text{left-}\exists \\ \Big| \\ (\forall y)(pc_1, y) \circ (\forall y)((\exists x)(p\,x, y)) \\ \text{left}^+\text{-}\forall \\ \Big| \\ (\forall y)(pc_1, y); pc_1, c_1 \circ (\forall y)((\exists x)(p\,x, y)) \\ \text{right-}\forall \\ \Big| \\ (\forall y)(pc_1, y); pc_1, c_1 \circ (\exists x)(p\,x, c_2) \\ \text{left}^+\text{-}\forall \\ \Big| \\ (\forall y)(pc_1, y); pc_1, c_2; pc_1, c_1 \circ (\exists x)(p\,x, c_2) \\ \text{right}^+\text{-}\exists \\ \Big| \\ (\forall y)(pc_1, y); \mathbf{pc_1, c_2}; pc_1, c_1 \circ (\exists x)(p\,x, c_2); \mathbf{pc_1, c_2}; pc_2, c_2 \\ \times \end{array}$$

Notice that with "left-$\exists$" the domain of the intended counterexample grows from $\varnothing$ to $\{c_1\}$. With "right-$\forall$" the domain of the intended counterexample grows from $\{c_1\}$ to $\{c_1, c_2\}$.

The following example shows what might happen if rules that *produce* constants (such as left-$\exists$ and right-$\forall$) are mixed with rules that *use* constants.

Example 3.8 If everybody likes some music genre, this does not mean that there is someone who likes all music genres: (i) $\nrightarrow$ (ii) [page 61]. Here is a non-terminating refutation (only with new formulas left and right):

$$(\forall x \exists y)(p\,x, y) \circ (\exists y \forall x)(p\,x, y)$$

c_0 is used to "kick-off" the refutation, then applying left$^+$-$\forall$ with c_0

$$\Big|$$

$$(\exists y)(p\,c_0, y) \circ$$

right$^+$-$\exists$ with c_0

$$\Big|$$

$$\circ\, (\forall x)(px, c_0)$$

left-$\exists$, introducing c_1

$$\Big|$$

$$p\,c_0, c_1 \circ$$

left$^+$-$\forall$ with c_1

$$\Big|$$

$$(\exists y)(p\,c_1, y) \circ$$

right$^+$-$\exists$ with c_1

$$\Big|$$

$$\circ\, (\forall x)(p\,x, c_1)$$

left-$\exists$, introducing c_2

$$\Big|$$

$$p\,c_1, c_2 \circ$$

left$^+$-$\forall$ with c_2

$$\Big|$$

$$(\exists y)(p\,c_2, y) \circ$$

$$\vdots$$

What happens here is that the fresh constants produced by "left-$\exists$" and "right-$\forall$" yields new material for "left$^+$-$\forall$" and "right$^+$-$\exists$". The last two rules on their turn produce new existential formulas on the left and new universal formulas on the right that produce new constants. This interaction between term-building and term-using rules lasts forever.

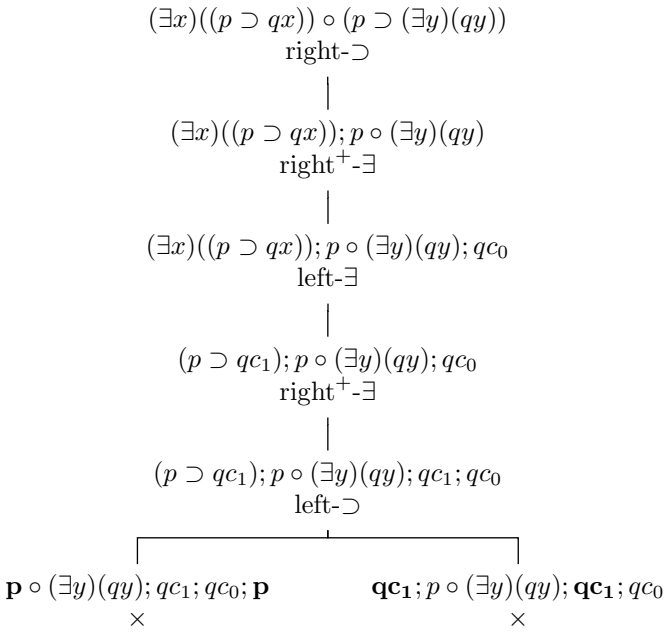
It is a question whether it is possible to extend your ATP with a device that recognizes such cases. Cf. Exercise 2.

PROBLEMS (Sec. 3.3)

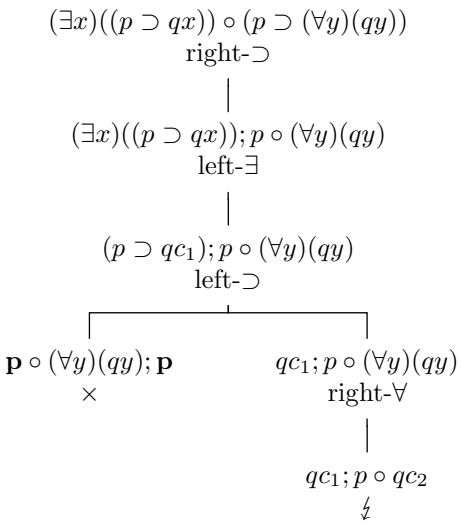
1. Try to refute $(\forall x \exists y)(p\,x, y) \circ (\exists y \forall x)(p\,x, y)$ yourself, but with a different proof strategy. Is it possible to escape from the endless loop? Explain your answer. (Solution.)

2. Is it possible to change the tableaux method in such a way that it recognizes cases such as displayed in Example 3.8? How? (Solution.)

Example 3.9 A failed refutation (= proof):



Example 3.10 A refutation:



Counterexample: model M with domain $D = \{1, 2\}$, with c_1 interpreted as 1 and c_2 interpreted as 2, and with

Predicate	*Extension*
q	$\{c_1\}$
p	true

Note that it is important that $1 \neq 2$ in this counterexample.

Sound- and completeness of first-order refutation trees

- Soundness: if a sequent is falsifiable, then all refutation trees for that sequent have at least one branche that cannot be closed.

- Completeness: if a sequent is valid (i.e., not falsifiable), then every refutation tree closes.

Sound- and completeness: a sequent is valid if and only if all refutations close.

Proving completeness within the framework of first-order logic is considerable more work than the propositional case, but amounts to the same ideas. The only problem here is that a falsifying branch may be infinite. In such cases, there is no end-node that spells out a counter-model for us. This is as far as the difficulty goes, however, because it can be proven that every infinite first-order refutation branch gives rise to a first-order model M that falsifies all sequents on that branch. M can be constructed inductively. (It is not hard to imagine how this could be done, since the formula-complexity along an infinite branch decreases monotonically when we move farther from the root.) The result is that every infinite first-order refutation branch gives rise to a first-order model M that falsifies all sequents on that branch. This result is known as *Hintikka's lemma* [106, 27].

Definition 3.11 (Hintikka set, first-order logic) A set of first-order nodes S is a *Hintikka set* if it is a Hintikka set in the propositional sense of the word (cf. page 42) and additionally satisfies the following properties:

9.	$\ldots, (\forall x)p, \ldots \circ \ldots \in S$	implies	$\ldots, (\forall x)p, p\,[t/x], \ldots \circ \ldots \in S$, for all ground terms t
10.	$\ldots, (\exists x)p, \ldots \circ \ldots \in S$	implies	$\ldots, p\,[t/x], \ldots \circ \ldots \in S$, for some ground term t
11.	$\ldots \circ \ldots, (\forall x)p, \ldots \in S$	implies	$\ldots \circ \ldots, p\,[t/x], \ldots \in S$, for some ground term t
12.	$\ldots \circ \ldots, (\exists x)p, \ldots \in S$	implies	$\ldots \circ \ldots, (\exists x)p, p\,[t/x], \ldots \in S$, for all ground terms t

PROBLEMS (Sec. 3.3)

1. Prove that the set of sequents on a saturated branch is a Hintikka set. (Solution.)

2. (Hintikka's lemma.) Prove that every Hintikka set of sequents possesses a countermodel that falsifies of its elements simultaneously. (Solution.)

3. Argue that (1) and (2) together imply that the root of a saturated branch is falsifiable. (Solution.)

3.4 Functions (no equality)

Function symbols complicate theorem proving, because it is possible to produce many terms with the help of only a few function symbols:

$$\text{HerbrandDomain}(pf(a)) = \{a, f(a), f(f(a)), f(f(f(a))), \ldots\}$$

All terms thus generated could, in principle, be used by left-$\forall$ or right-$\exists$ as long as at least one branch remains open.

Let us study a concrete case to get an estimate of the problems that may arise.

Example 3.12 Suppose we are considering a situation in which the only relevant constants and function symbols are the two constants a and b, and a one-place function symbol f. In such a

situation the formula $(\forall x)(p\,x)$ may be "unfolded," as follows:

$$\begin{aligned}(\forall x)(p\,x) &\equiv (\forall x)(p\,x)\\ &\equiv (\forall x)(p\,x) \wedge pa\\ &\equiv (\forall x)(p\,x) \wedge pb \wedge pa\\ &\equiv (\forall x)(p\,x) \wedge pf(a) \wedge pb \wedge pa\\ &\equiv (\forall x)(p\,x) \wedge pf(b) \wedge pf(a) \wedge pb \wedge pa\\ &\equiv (\forall x)(p\,x) \wedge pf(f(a)) \wedge pf(b) \wedge pf(a) \wedge pb \wedge pa\\ &\equiv (\forall x)(p\,x) \wedge \ldots \wedge pf(f(a)) \wedge pf(b) \wedge pf(a) \wedge pb \wedge pa\end{aligned}$$

Think about how to produce this within a text-editor with cut and paste. The same principle applies when writing a subroutine to unfold universally quantified formulas.

The number of formulas to be generated soon runs out of hand in case there are more constants and more function symbols, especially if the functions have two or more arguments. The fact that "left$^+$-$\forall$" and "right$^+$-$\exists$" do not reduce the formulas they operate on is a problem, because, usually, a first-order language $\mathcal{L}$ consists of a large number of constant symbols $c_1, c_2, \ldots$, a large number of variables $x_1, x_2, \ldots$, a large number of one-place function symbols $f_1^1, f_2^1, \ldots$, a large number of two-place function symbols $f_1^2, f_2^2, \ldots$, and so forth. This would make for an even larger number of candidate terms to substitute in a $\forall$-sentence on the LHS (or a $\exists$-sentence on the RHS), namely:

$$c_1, x_1, f_1^1(c_1), f_1^1(x_1), c_2, x_2, f_1^1(c_2), f_1^1(x_2), f_2^1(c_1), f_2^1(x_1), \ldots \tag{3.3}$$

This sequence is still countably infinite (since the Cartesian product of two or more countably infinite sets is countably infinite), but it will not come as a surpise that it will take a while until a proof algorithm encounters the right terms to close a refutation tree (if it closes at all).

Not all terms in (3.3) have to be used in a substitution. Counterexamples need only be constructed from terms that can be made from free variables, function symbols and constants that occur in the original formula. The set that consists of these terms is called the *Herbrand domain* of that formula.

PROBLEMS (Sec. 3.4)

1. The goal of this exercise is see how the Herbrand domain of a (set of) first-order formula(s) can be constructed systematically. It is an important exercise, since it gives you an impression of the relative difficulty of enumerating a Herbrand domain.

   Let $P(g(f(a), b))$ be a sentence.

   (a) Find the set of free variables, function symbols and constants that occur in the above sentence.

   (b) What is the arity of the function symbols?

   (c) Let $T_1 = \{a, b\}$. If f is applied to T_1 once, we obtain $T_2 = \{a, b, f(a), f(b)\}$. Describe the set T_2 that is formed by applying g to all terms in T_1 once. How many new terms are added?

   (d) Describe the set T_3 that is formed by applying f to all terms in T_2 once, without producing terms that are already present. How many terms are added? (Hint: to fill a square of n^2 tiles in which a corner of k^2 tiles is already filled, you have to add $n^2 - k^2 = n(n-k) + (n-k)(n-k)$ tiles: first one large side, and then then the remaining side from which one part of $(n-k)^2$ tiles has just been filled. In this way you add each tile exactly once. Similarly, to fill a cube of n^3 blocks in which a corner of k^3 blocks is already filled, you have to add $n^3 - k^3 = nn(n-k) + n(n-k)(n-k) + (n-k)(n-k)(n-k)$ blocks.)

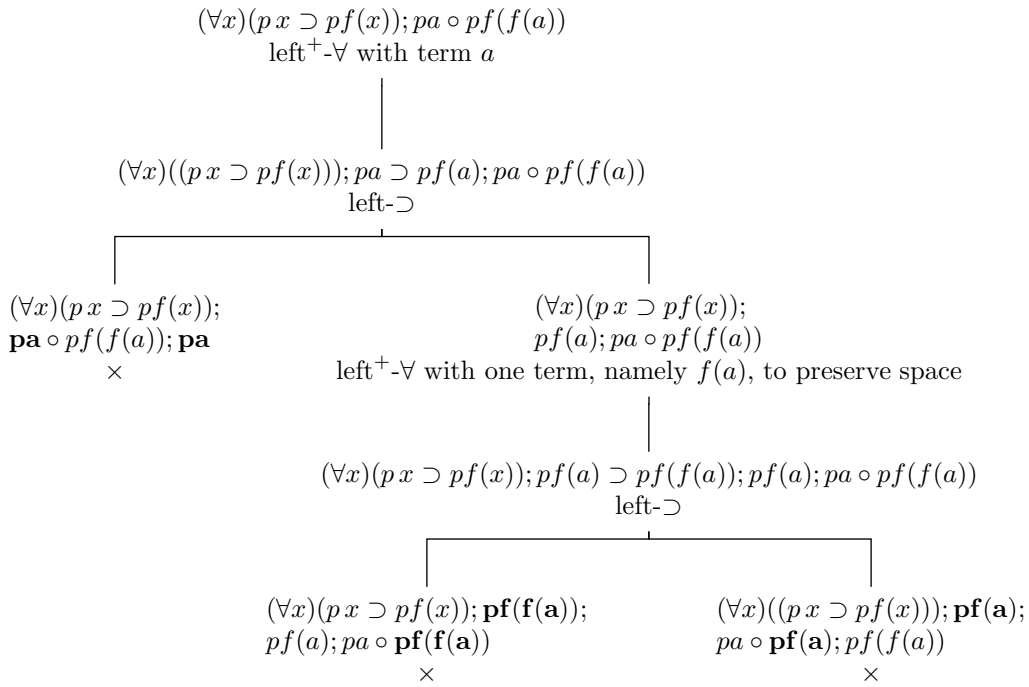
(e) Calculate how many terms are added in the transition from T_3 to T_4 and the transition from T_4 to T_5, again without producing terms that are already present. (Beware: don't spell out the sets themselves!)

(f) Formulate a procedure to generate terms from the free variables, function symbols and constants that occur in an arbitrary sentence.

Example 3.13 Let us try to refute the valid sequent $(\forall x)(p\,x \supset pf(x)); pa \vdash pf(f(a))$. The Herbrand domain of this sequent is equal to

$$\text{HerbrandDomain}\{a, f(a), f(f(a)), \ldots\} \tag{3.4}$$

which is infinite. Therefore, we will have to take care when we apply left$^+$-$\forall$ or right$^+$-$\exists$, because it is impossible to substitute all terms of the Herbrand domain at once:



Notice that there happens something new here: the proof "requires" a term, namely $f(f(a))$, that is not present in the original sequent. Instead, it must be generated explicitly, in the course of the proof. Luckily, it was $f(f(a))$ that we needed, and not, e.g., $f^{10,000}(a)$.

PROBLEMS (Sec. 3.4)

1. Let $\{x, f/1, g/2, a, b\}$ be the free variables, function symbols and constants that occur in a sequent (where $f/1$ means that f is a one-place function symbol).

   (a) What is the Herbrand domain of the sequent in question?

   (b) Suppose we try to refute the sequent in question. Give an enumeration of terms that may be used for left$^+$-$\forall$ and right$^+$-$\exists$.

   (c) Suppose that c_1 and c_2 are fresh constants that are introduced in the course of the refutation, for example as a result of two times applying right-$\forall$. Same question as with (1b).

   (d) Describe how the list of terms that may be used for left$^+$-$\forall$ and right$^+$-$\exists$ can be enumerated if the arsenal of free variables, function symbols and constants that occurs in the refutation tree is equal to a finite subset of $\{x, f/1, g/2, a, b, c_1, c_2, c_3, \ldots\}$.

2. Construct a refutation tree of $(\forall x)((\forall y)p\,x, y) \vdash (\forall x)((\forall y)pf(x), g(f(x), y))$. Maintain a correct bookkeeping of fresh variables and new terms. (Expect the refutation tree to have ≈ 14 nodes.)

3. Determine the size of a refutation tree for $(\forall x)(p\,x \supset pf(x)); pa \vdash pf^{10,000}(a))$ (See Example 3.13 on the preceding page.) If there are more refutation trees, then give the size of the smallest one possible.

3.5 First-order proofs in KE

The following set of exercises is taken from Ulrich Endriss' course on automated reasoning. (Cf. the set of exercises on propositional KE on page 45.) It requires the program WinKE. (Cf. Appendix A on page 253.)

PROBLEMS (Sec. 3.5)

1. Work on the (three) problems in the file `cs3aur-1.ke`, which you can download from the course web site.

2. An article on computational complexity in the *New York Times* from 13 July 1999 starts like this:

   > *"Anyone trying to cast a play or plan a social event has come face-to-face with what scientists call a satisfiability problem. Suppose that a theatrical director feels obligated to cast either his ingénue, Actress Alvarez, or his nephew, Actor Cohen, in a production. But Miss Alvarez won't be in a play with Cohen (her former lover), and she demands that the cast include her new flame, Actor Davenport. The producer, with her own favors to repay, insists that Actor Branislavsky have a part. But Branislavsky won't be in any play with Miss Alvarez or Davenport. [. . .]"*

   Is there a possible casting (and if there is, who will play)?

   Formalize the problem and enter it into WinKE. Then saturate the corresponding KE proof tree. If there is an open branch left it will correspond to a model (a possible casting). You can use the *Countermodel* option from the *Analysis* menu to generate it. (WinKE uses the term 'countermodel', because a model constitutes a counterexample for an attempted KE proof.)

3. (a) Try to prove

   $$(\forall x)(p\,x \vee qx) \models (\forall x)(p\,x) \vee (\forall x)(qx). \tag{3.5}$$

   (b) Proving (3.5) is not possible (because it is false). Try to get WinKE to generate a countermodel for you. To do this you need to expand one of the branches (in case there are several) until it is 'obvious' that further rule applications would never succeed in closing it. (In the WinKE help system, under useful instantiation, you can read how WinKE can decide in some cases whether further rule applications would be futile; but note that there can be no general strategy of this kind as FOL is undecidable.)

   (c) WinKE's description of models is somewhat informal. Write down the model suggested by WinKE in the formal way introduced during classes.

4. Use WinKE to generate as many models as you can for the following set of sentences:

   $\{$ *loves(Arabella, Carlo)*,
   loves(Carlo, Bianca) $\rightarrow$ *loves(Dino, Bianca)*,
   loves(Eduardo, Arabella) $\vee$ *loves(Eduardo, Bianca)* $\}$

   Note that for this particular type of example, WinKE can provide a visualization of the generated model. Try it.

5. Use WinKE to prove the following statements. Part of the exercise is about understanding what exactly these statements mean (and how to enter them into WinKE).

   (a) $(\exists x)(p\,x) \rightarrow (\forall x)(p\,x); pa \models Pb$

   (b) The set $\{p \vee q,\ p \vee \neg q,\ \neg p \vee q,\ \neg p \vee \neg q\}$ is not satisfiable.

   (c) $(\forall x)(p\,x \vee qx), \neg(\exists x)(p\,x), (\exists x)(\neg qx) \models \bot$

   (d) $(\exists x)(p\,x \vee qx) \equiv (\exists x)(p\,x) \vee (\exists x)(qx)$ is a theorem.

   (e) $\models (\exists x)(\forall y, z)[(p\,y \supset qz) \supset (p\,x \supset qx)]$.

6. The objective of this question is to prove that any binary relation that is symmetric, transitive, and serial, must also be reflexive. (A relation is called serial if for any element we have an element to which the former is related.) Proceed as follows. Given the following abbreviations:

$$\begin{array}{rcl} \textit{reflexive} & = & (\forall x)(rxx) \\ \textit{symmetric} & = & (\forall x, y)(rxy \supset ryx) \\ \textit{transitive} & = & (\forall x, y, z)(rxy \wedge ryz \supset rxz) \\ \textit{serial} & = & (\forall x \exists y)(rxy) \end{array}$$

   Use WinKE to show:

   $\textit{symmetric} \wedge \textit{transitive} \wedge \textit{serial} \models \textit{reflexive}.$

3.6 Free-variable substitutions

When you implement a theorem-prover on the basis of Table 3.2 on page 58 and the notion of reducts (page 57), you will be able to prove most "textbook examples". You will probably also be able to generate countermodels for most (but certainly not all) invalid formulas. This is in fact the way in which almost all Gentzen-style proofs and reduction trees in this book were produced.[2]

Still, proofs on the basis of standard analytic refutation rules soon become very inefficient.[3] This inefficiency is caused by the rules left$^+$-$\forall$ and right$^+$-$\exists$ which may, in principle, fill a branch with all elements from the Herbrand domain. On page 57 we have seen that reducts are generally very, V E R Y, large. This is usually no problem, because when we do a refutation by hand, it is often obvious to us which terms must be picked from the Herbrand domain to close a branch. If, however, our algorithm is not equipped with the same sort of "intelligence," we are in trouble, as is shown in the following example.

Example 3.14 (Naive refutation) Suppose our naive algorithm tries to prove the invalid sequent $(\forall x)(Pf(x)) \vdash Pa$ with conventional means. For us, it is obvious that Pa cannot follow

[2] By means of a Perl script, refutation trees as well as cut-free Gentzen-style proof were written as LaTeX onto a file called `proof.tex`, after which parts of `proof.tex` were pasted into the present document. For invalid sequents, the script wrote a countermodel and a partially finished proof tree, also in LaTeX.

[3] Try, for instance, $Pa, b, c \wedge (\forall x)[Px, y, z \supset Pf(y, z), g(x, z), h(x, y)] \vdash Pb, c, a$.

from $(\forall x)(Pf(x))$, because a is not of the form $f(t)$, for some t. However, if our algorithm lacks the machinery to recognize this situation, then it just tries out all applications of left-$\forall$, with the "intention" to produce Pa:[4]

$$
\begin{array}{c}
(\forall x)(Pf(x)) \circ Pa \\
\text{left-}\forall \\
| \\
(\forall x)(Pf(x)), Pf(a) \circ Pa \\
\text{left-}\forall \\
| \\
(\forall x)(Pf(x)), Pf(a), Pf(f(a)) \circ Pa \\
\text{left-}\forall \\
| \\
\vdots \\
\text{left-}\forall \\
| \\
(\forall x)(Pf(x)), Pf(a), Pf(f(a)), \ldots, Pf^n(a) \circ Pa \\
\vdots
\end{array}
$$

Obviously, this leads to nowhere.

What do we know what the naive algorithm doesn't? I.e., how do we see that most term-substitutions do not close branches, and, more importantly, how do we know which terms *do* close branches? How do we select our substitutions?

What we probably do when constructing refutation trees ourselves, is to postpone term-substitutions until we see an opportunity to close a branch. When encountering a left-$\forall$, for instance, we do not substitute a specific term, but instead mark it with a *place-holder*, i.e., a fresh variable v_i, for the time being, with the intention to replace v_i by a "real" term as soon as we see an opportunity to close a branch. An advantage of this approach is that it also rules out left$^+$-$\forall$ and right$^+$-$\exists$ as term-generators, because as long as v_i is not replaced by a specific ground term, it may represent any such term.

There are two problems with this approach. The first problem is that is unclear whether this approach works when branches split, since the v_i above splits are used to serve different branches with possibly different variable substitutions. We will discuss this problem later in Example 3.17 on page 72. The second problem is that left-$\exists$ and right-$\forall$ now become problematic, since the constants that we use in these rules must be fresh. But how do we know that they are fresh if the set of constants on a branch which we are working at, is undefined? The problem is that the set of constants that occur on a branch with pending variables $v_1, \ldots, v_n$ is undefined, since we do not now which constants occur in the terms that substitute $v_1, \ldots, v_n$ eventually.

A solution to this problem is to indicate that constants on a branch with postponed substitutions $v_1, \ldots, v_n$ depend on the value of $v_1, \ldots, v_n$. For example, if a branch contains the constants a, d, e and pending variables $v_1, \ldots, v_n$, we do not introduce a fresh constant c, but a fresh *function* ς, and indicate that ς depends on $v_1, \ldots, v_n$ by writing $\varsigma(v_1, \ldots, v_n)$. Thus we now run proofs with fresh variables v_i and fresh function symbols ς_i (Table 3.5 on the following page). These functions are called *Skolem functions*, for reasons that will become clear later.

Soundness of Table 3.5 on the next page is not self-evident, because the rules are rather liberal. Soundness was proven only in 1994 by Hähnle *et al.* [41, 42]. See also Fitting's work [27].

The following example is from Fitting:

[4] Obviously, algorithms like the one we discuss have no intentions, but the terminology is used for the sake of illustration.

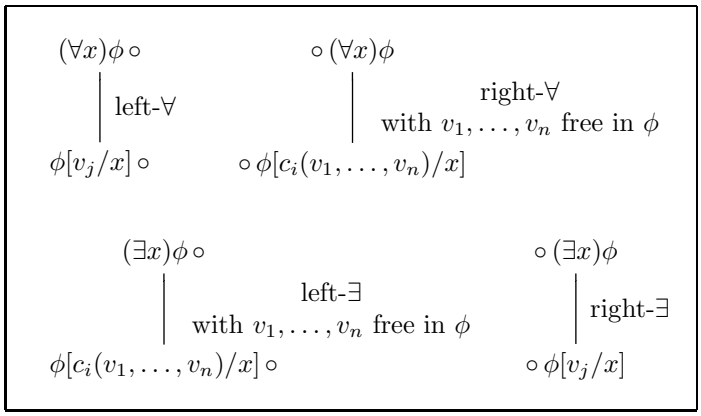


Table 3.5: Analytic refutation rules for first-order logic with postponed substitutions.

Example 3.15 (Refutation with postponed substitutions) Here is an example of a failing refutation (= proof) of the valid sequent $(\exists w, \forall x)(Rx, w, f(x, w)) \vdash (\exists w, \forall x, \exists y)(Rx, w, y)$. All substitutions in the refutation are postponed.

$$(\exists w, \forall x)(Rx, w, f(x, w)) \circ (\exists w, \forall x, \exists y)(Rx, w, y)$$
left-$\exists$, introducing the constant ς_1 (no pending variables)
$$\Big|$$
$$(\forall x)(Rx, \varsigma_1, f(x, \varsigma_1)) \circ (\exists w, \forall x, \exists y)(Rx, w, y)$$
right-$\exists$, substituting v_1 for w
$$\Big|$$
$$(\forall x)(Rx, \varsigma_1, f(x, \varsigma_1)) \circ (\forall x, \exists y)(Rx, v_1, y)$$
right-$\forall$, introducing the function ς_2 with argument v_1
$$\Big|$$
$$(\forall x)(Rx, \varsigma_1, f(x, \varsigma_1)) \circ (\exists y)(R\varsigma_2(v_1), v_1, y)$$
left-$\forall$, substituting v_2 for x
$$\Big|$$
$$Rv_2, \varsigma_1, f(v_2, \varsigma_1) \circ (\exists y)(R\varsigma_2(v_1), v_1, y)$$
right-$\exists$, substituting v_3 for y
$$\Big|$$
$$Rv_2, \varsigma_1, f(v_2, \varsigma_1) \circ R\varsigma_2(v_1), v_1, v_3$$

The last node of the refutation tree has the predicate R on the left and on the right, and the branch would close if these predicates were equal, i.e, if $v_2 = \varsigma_2(v_1)$, $\varsigma_1 = v_1$, and $f(v_2, \varsigma_1) = v_3$. This can be realized with the substitution ς_1/v_1, $\varsigma_2(\varsigma_1)/v_2$, $f(\varsigma_2(\varsigma_1), \varsigma_1)/v_3$:

$$\vdots$$
$$\Big|$$
$$Rv_2, \varsigma_1, f(v_2, \varsigma_1) \circ R\varsigma_2(v_1), v_1, v_3$$
$$/v_3$$
$$\times, \text{ with } \varsigma_1/v_1, \varsigma_2(\varsigma_1)/v_2, f(\varsigma_2(\varsigma_1)), \varsigma_1$$

Hence, the tableau closes.

Now an example that shows that "left-$\forall$" and "right-$\exists$" type of formulas must be kept, because otherwise tableaux may not close for valid sequents.

Example 3.16 The sequent

$$\vdash (\forall x)(\exists y)(\forall z)(\exists w)[rx, y \vee \neg rw, z] \tag{3.6}$$

is valid, so that a refutation should fail. We could start as follows:

$$\begin{array}{c}
\circ\ (\forall x)(\exists y)(\forall z)(\exists w)[rx, y \vee \neg rw, z] \\
\text{right-}\forall \\
| \\
\circ\ (\exists y)(\forall z)(\exists w)[r\varsigma_1, y \vee \neg rw, z] \\
\text{right-}\exists\text{, creating } v_1 \\
| \\
\circ\ \phi, (\forall z)(\exists w)[r\varsigma_1, v_1 \vee \neg rw, z] \\
\text{right-}\forall \text{ with } v_1 \text{ free in RHS} \\
| \\
\circ\ \phi, (\exists w)[r\varsigma_1, v_1 \vee \neg rw, \varsigma_2(v_1)] \\
\text{right-}\exists\text{, creating } v_2 \\
| \\
\circ\ \phi, \psi, r\varsigma_1, v_1 \vee \neg rv_2, \varsigma_2(v_1) \\
\text{right-}\vee \\
| \\
\circ\ \phi, \psi, r\varsigma_1, v_1,\ \neg rv_2, \varsigma_2(v_1) \\
\text{right-}\neg \\
| \\
rv_2, \varsigma_2(v_1) \circ \phi, \psi, r\varsigma_1, v_1
\end{array}$$

where $(\exists y)(\forall z)(\exists w)[r\varsigma_1, y \vee \neg rw, z]$ and $(\exists w)[r\varsigma_1, v_1 \vee \neg rw, \varsigma_2(v_1)]$ are "taken along to the bottom" as ϕ en ψ.

Now, the question is: how to proceed? At this point, $rv_2, \varsigma_2(v_1)$ cannot be unified with $r\varsigma_1, v_1$, because v_1 occurs in $\varsigma_2(v_1)$. This is a problem, because (3.6) is valid, so that the tableaux must close.

At this point it becomes clear why it was useful "to take along" ϕ to the bottom: in this way we can instantiate ϕ with "right-$\exists$" for a second time. We proceed:

$$\begin{array}{c}
rv_2, \varsigma_2(v_1) \circ \phi, \psi, r\varsigma_1, v_1 \\
\text{right-}\exists\text{, creating } v_3 \\
| \\
rv_2, \varsigma_2(v_1) \circ \phi, \psi, (\forall z)(\exists w)[r\varsigma_1, v_3 \vee \neg rw, z], r\varsigma_1, v_1 \\
\text{right-}\forall \\
| \\
\vdots \\
| \\
rv_2, \varsigma_2(v_1), rv_4, \varsigma_3(v_3) \circ \phi, \psi, r\varsigma_1, v_1, r\varsigma_1, v_3 \\
\times \text{ with } \sigma = \{\varsigma_1/v_1, \varsigma_1/v_2, \varsigma_2(\varsigma_1)/v_3\}
\end{array}$$

Now the tableaux *does* close, because $rv_2, \varsigma_2(v_1)$ is unifiable with $r\varsigma_1, v_3$. □

The following example shows how to deal with different branches in which the same pending variable occurs.

Example 3.17 (Refutation with postponed substitutions) Consider the sequent

$$(\forall x)(Nx \supset Ns(x)) \vdash N0 \supset Ns^{100}(0).$$

A possible interpretation of this sequent is to let $N(x)$ be true if and only if x is a positive integer, and to let $s(x)$ be the successor of x, i.e., $s(x) = x + 1$. Then

$$\begin{array}{c} (\forall x)(Nx \supset Ns(x)) \circ N0 \supset Ns^{100}(0) \\ \text{left-}\forall\text{, substituting } v_1 \text{ for } x \\ \Big| \\ Nv_1 \supset Ns(v_1) \circ N0 \supset Ns^{100}(0) \\ \text{right-}\supset \\ \Big| \\ Nv_1 \supset Ns(v_1), N0 \circ Ns^{100}(0) \\ \text{left-}\supset \\ \begin{array}{cc} \ulcorner & \urcorner \end{array} \\ \begin{array}{cc} Ns(v_1), N0 \circ Ns^{100}(0) & N0 \circ Nv_1, Ns^{100}(0) \\ \times, \text{ with } s^{99}(0)/v_1 & \times, \text{ with } 0/v_1 \end{array} \end{array}$$

Thus, variables that correspond to postponed substitutions may represent different substitutions in different branches. This is allowed, since different branches correspond to different ways to falsify a formula.

The above example is of course not a *proof* that different branches permit different substitutions, let alone that it is a proof that the postponed substitution method is complete. We merely have made it plausible that every refutation of a valid sequent results in a closed tree, but we have not really demonstrated it by means of a rigorous mathematical proof. Such a proof falls beyond the scope of this book, but can be found in, e.g., [27, 19].

PROBLEMS (Sec. 3.6)

1. The last sequent of the refutation of Example 3.16 on the preceding page can also be closed with another substitution. True or false? (Solution.)

2. Prove the following sequents with semantic tableaux with postponed substitutions.

   (a) $(\forall x)(p\,x) \vdash (\forall x)(p\,x)$ (Solution.)

   (b) $(\exists x)[p\,x \supset (\forall x)(p\,x)]$ (Solution.)

   (c) $(\forall x, y)(p\,x \wedge p\,y) \vdash (\exists x, y)(p\,x \vee p\,y)$ (Solution.)

   (d) $(\forall x, y)(p\,x \wedge p\,y) \vdash (\forall x, y)(p\,x \vee p\,y)$ (Solution.)

3.7 Unification

With free-variable substitutions, we need an algorithm that computes for us if two terms can be made equal, and if so, how. The process of trying to make two terms equal is called *unification*. Unification is an important process in ATP because it also plays an important role in resolution.

Definition 3.18 (Unification) A unification of terms t_1 and t_2 is a substitution σ that makes $t_1\sigma$ and $t_2\sigma$ syntactically equal.

The aim of this section is to formulate a *unification algorithm*.

Unification-like procedures were already proposed around 1930 by Herbrand [43]. The term unification itself was introduced by Robinson [96], who also indicated its importance for automated theorem proving. The algorithm that will be given here is due to Robinson (Table 1). To understand the algorithm, we must first know how all the differences between two terms are determined. The *differences* between two terms t_1 and t_2 can be expressed by a set

$$D = \{(s_1^1, s_2^1), \ldots, (s_1^n, s_2^n)\} \tag{3.7}$$

of pairs of terms, such that

- $s_1^i \neq s_2^i$
- Every s_1^i is a sub-term of t_1 and every s_2^i is a sub-term of t_2.
- The terms t_1' and t_2' that are created when all s_j^i's are replaced by a similar token are syntactically equal.

  (A token here is a meaningless plug to stop the hole that came to existence when all the s_j^i's are removed from t_1 and t_2. It functions as a place-holder to denote that a new term must be plugged in at that particular point in the syntactic tree.)
- All the s_j^i's are as large as possible.

Algorithm 1 Unification

Input: two terms t_1, t_2

1: - let $\sigma = \varnothing$
2: **while** $t_1\sigma \neq t_2\sigma$ **do**
3: - choose pair of terms (s_1, s_2) that are different in $(t_1\sigma, t_2\sigma)$
4: - **return** `undef` **if** neither s_1 nor s_2 are variables
5: - swap s_1 and s_2 **if** s_1 is not a variable
6: - **return** `undef` **if** variable s_1 occurs in s_2
7: - $\sigma = \sigma \cup \{s_2/s_1\}$
8: - **return** σ

Here is an example.

Example 3.19 Let

$$t_1 = g(f(h(b), y), h(c)),$$
$$t_2 = g(f(z, d), h(h(x)))$$

(Figure 3.1 on the following page.) Thus, f and g are two-place (binary) function symbols, while h is a one-place (unary) function symbol.

The differences between t_1 and t_2 are

$$s_1^1, s_2^1 = h(b), z$$
$$s_1^2, s_2^2 = y, d$$
$$s_1^3, s_2^3 = c, h(x)$$

Note that the s_j^i's comply to the restrictions above.

If we would like to unify t_1 and t_2, then $h(b)$ should be equal to z, y should be equal to d, and c should be equal to $h(x)$. The first two requirements can be fulfilled by the (partial) substitution $z/h(b)$ and y/d. The third requirement cannot be fulfilled, however, because s_1^3 is a constant while s_2^3 starts with a function symbol.

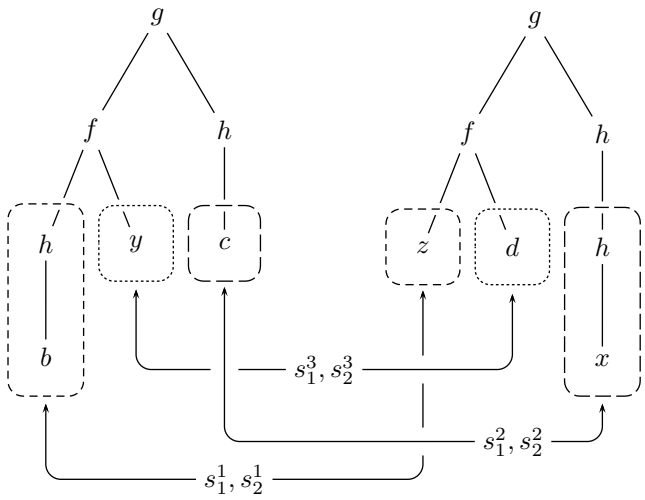


Figure 3.1: Differences between two terms.

To prove that Algorithm 1 on the previous page is correct, we will have to show two things: (i) if (t_1, t_2) can be unified, then Algorithm 1 will find a unification (completeness); (ii) Algorithm 1 will not unify terms that cannot be unified (soundness). A proof of the correctness of Algorithm 1 can be found in, e.g., [27].

PROBLEMS (Sec. 3.7)

1. We say that σ *subsumes* τ if there is a third substitution, μ, such that $\tau = \sigma\mu$ (i.e. τ is identical to first applying σ and then μ). A *most general unifier* (MGU) of two terms t_1 and t_2 is a unifier σ such that every other unifier σ' of t_1 and t_2 is subsumed by σ. A MGU resembles the largest common divisor of two integers. I hasten to add that the analogy only goes halfway, since a LCD always exists, while a MGU need not always exist.

   Determine a MGU for the following pairs of terms, if one exists:

   (a) $t_1 = f(x, a)$, $t_2 = f(b, y)$ (Solution.)

   (b) $t_1 = g(x, a)$, $t_2 = g(b, x)$ (Solution.)

   (c) $t_1 = h(x, a, w, y)$, $t_2 = h(b, a, z, c)$ (Solution.)

   (d) $t_1 = f(x, w, w, x)$, $t_2 = f(w, z, d, a)$ (Solution.)

   (e) $t_1 = g(x, y, w, z)$, $t_2 = g(w, f(z), y, z)$ (Solution.)

   (f) $t_1 = h(x, y, y, z)$, $t_2 = h(x, y, z, y)$ (Solution.)

2. Explain why the unification algorithm always halts. (Hint: consider the total number of variables in t_1 and t_2.)

3. Suppose t_1 and t_2 are unifiable.

   (a) Write down formally the fact that t_1 and t_2 are unifiable.

   (b) Let s_1, s_2 be one of the differences between t_1 and t_2. Explain that one of the two must be a variable that does not occur in the other term.

   (c) Suppose s_1 is a variable. Let σ be a unifier of t_1 and t_2. Show that $\sigma = \sigma \cup \{s_2/s_1\}$.

(d) Prove that the unification algorithm given in Table 1 on page 73 terminates without unification, if its input is not unifiable.

(e) Suppose t_1 and t_2 are unifiable. Let τ be a unifier of t_1 and t_2.

i. Let σ as defined in the unification algorithm. Prove that $\tau = \sigma\tau$ at the start of the algorithm.

ii. Prove that the equation $\tau = \sigma\tau$ is an invariant of the for-loop. (I.e., show that the equation remains valid during the for-loop.)

iii. Prove that the unification algorithm returns a MGU of t_1 and t_2 if both terms are unifiable.

iv. A substitution is *idempotent* if $\sigma = \sigma\sigma$. Prove that the unification algorithm returns an idempotent unifier if its input is unifiable.

3.8 Functions and equality

Almost all realistic problems involve equalities, so ATP becomes interesting when equality comes into play. Unfortunately, the introduction of equality is also the point where automated theorem proving suddenly starts to become rather complicated. In fact, equational reasoning is a vast sub-field of automated theorem proving.

Table 3.6 on the next page, introduces additional rules for dealing with equality. The first and most important thing to note is that left and right-replacement rules are *directional*. They only permit a left-to-right use of equalities. For instance, if $a = b$ we may rewrite pa into pb, but we may not rewrite pb into pa. Further, the predicate P can be the equality predicate itself, so that we may use equalities to rewrite other equalities.

It can be proven that the rules given in Table 3.6 are sound and complete for first-order predicate logic with equality. A completeness proof for first-order predicate logic with equality is long, but doable [67, 29, 7, 27]. Because it is a well-known result and doesn't help us very much in getting to grips with the problems of first-order ATP, we don't go into the details of such a completeness proof. (The ideas are similar to those given on page 63. The difference is that we must now work with equivalence classes of terms, where two terms are considered equivalent if they can be unified.)[5]

Figure 3.2 on page 77 shows an example of a proof of transitivity of $=$: Figure 3.3 on page 78 shows a proof of $(\forall x)((\exists y)(y = f(x) \wedge (\forall z)(z = f(x) \supset y = z)))$.

PROBLEMS (Sec. 3.8)

1. Prove the following formulas by refutation. (Function and predicate substitution are not needed.)

   (a) Symmetry: $(\forall x)((\forall y)(x = y \supset y = x))$ (Solution.)

   (b) Existential reflexivity: $(\forall x)((\exists y)(x = y))$ (Solution.)

2. Prove the following formulas by refutation.

   (a) $(\forall u)((\forall v)((\forall w)((\forall x)((u = v \wedge w = x) \supset (Pu, w \supset Pv, x)))))$ (Solution.)

3. Prove the following formulas by refutation, with postponed substitutions.

   (a) $(\forall x)(\exists y)(\exists z)(y = f(x) \wedge z = g(y))$ (Solution.)

   (b) $(\forall x)(\forall y)(x = y \supset f(x) = f(y))$ (Solution.)

[5] Unification is explained on page 72.

Rules from Table 3.2 on page 58, plus

$$\dfrac{\circ}{s = s \circ}\ \text{left-=} \qquad \dfrac{\circ}{f(t_1,\ldots,t_n) = f(t_1,\ldots,t_n) \circ}\ \text{left-=, for functions}$$

$$\dfrac{s = t, P(\ldots,s,\ldots) \circ}{s = t, P(\ldots,s,\ldots), P(\ldots,t,\ldots) \circ}\ \text{left replacement} \qquad \dfrac{s = t \circ P(\ldots,s,\ldots)}{s = t \circ P(\ldots,s,\ldots), P(\ldots,t,\ldots)}\ \text{right replacement}$$

Table 3.6: Analytic refutation rules for first-order logic with equality.

(c) $(\forall x)(\forall y)(\forall z)(y = f(x) \wedge z = f(x) \supset y = z)$ (Solution.)

4. Show that, given, Table 3.6, the following rules are valid:

$$\dfrac{s = t \circ}{t = s \circ}\ \text{swap} \qquad \dfrac{s = t \circ P(\ldots,t,\ldots)}{s = t \circ P(\ldots,t,\ldots), P(\ldots,s,\ldots)}\ \text{right}_2\ \text{replacement}$$

(Solution.)

3.9 Heuristics

By moving from propositional to first-order logic, we gained expressive power at the price of proof complexity. Adding equality has the same effect, much intensified. With equality rules present, theorem-provers have so many paths to explore that they can generally succeed only with simple and rather uninteresting theorems.

To help theorem-provers prioritize their schedule of logic operations, they must be equipped with a heuristic. A heuristic consists of guidelines that help a theorem proving algorithm to steer the search process in the right direction. (What is "right" depends on the theorem we would like to prove.) In the case of first-order logic, a heuristic typically consists of guidelines on the basis of which we can decide which branches must be explored first, which sequents on those branches must be analyzed first and, most importantly, which term-substitutions with those sequents must be made first. The problem to determine which branch, sequent, or substitution is most promising is up to the (designer of the) heuristic.

PROBLEMS (Sec. 3.9)

1. A minimum requirement on a heuristic, is that it must be be *fair*. A heuristic is fair, if it ensures that all candidate-branches, all candidate-sequents on a branch, all

$$\circ(\forall x)((\forall y)((\forall z)(x = y \wedge y = z \supset x = z)))$$
right-$\forall$
$$\circ(\forall y)((\forall z)(\varsigma_1 = y \wedge y = z \supset \varsigma_1 = z))$$
right-$\forall$
$$\circ(\forall z)(\varsigma_1 = \varsigma_2 \wedge \varsigma_2 = z \supset \varsigma_1 = z)$$
right-$\forall$
$$\circ\varsigma_1 = \varsigma_2 \wedge \varsigma_2 = \varsigma_3 \supset \varsigma_1 = \varsigma_3$$
right-$\supset$
$$\varsigma_1 = \varsigma_2 \wedge \varsigma_2 = \varsigma_3 \circ \varsigma_1 = \varsigma_3$$
left-$\wedge$
$$\varsigma_1 = \varsigma_2, \varsigma_2 = \varsigma_3 \circ \varsigma_1 = \varsigma_3$$
using $\varsigma_1 = \varsigma_2$ on the left to rewrite $\varsigma_1 = \varsigma_3$ on the right into $\varsigma_2 = \varsigma_3$
$$\varsigma_1 = \varsigma_2, \varsigma_2 = \varsigma_3 \circ \varsigma_1 = \varsigma_3, \varsigma_2 = \varsigma_3$$
$\times$

Figure 3.2: Proof of transitivity of $=$

candidate-terms in a sequent, and all candidate-substitutions of a term are tried eventually, as far as time and space permit.

Give an example of a heuristic that is unfair with respect to the selection of ...

(a) branches.

(b) sequents.

(c) terms.

(d) substitutions.

2. Propose a prioritization strategy with respect to equality substitution.

3.10 First-order benchmarks

Pelletier's problem set

Problems 1-17 of Pelletier's problem set are formulated in the language of propositional logic (page 47). All other problems are first-order problems: one-place predicates occur from Problem 18 onwards, n-place predicates occur from Problem 36 onwards, identity occurs from Problem 48 onwards, and function symbols occurs from Problem 56 onwards.

A random selection:

48. $a = b \vee c = d, a = c \vee b = d \vdash a = d \vee b = c$ (3pts)

    'A problem to test identity ATPs (Piotr Rudnicki, University of Alberta).

49. $(\exists x)(\exists y)(\forall z)(z = x \vee z = y), pa \wedge pb, a \neq b \vdash (\forall x)(px)$ (5pts)

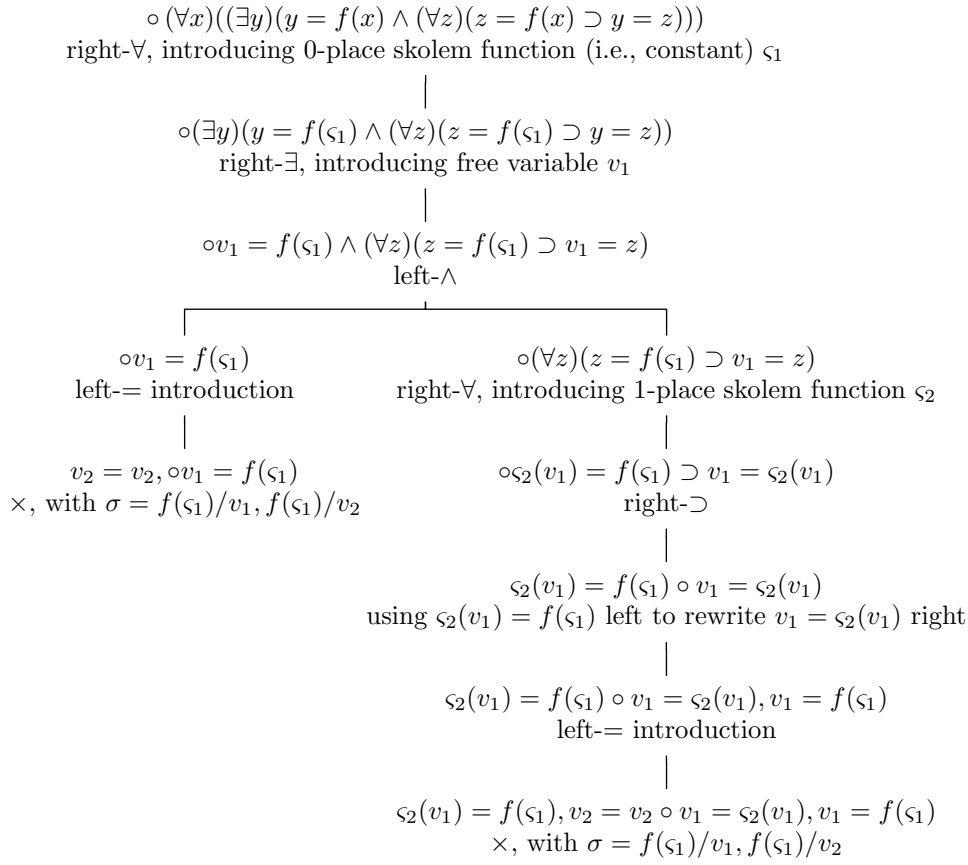


Figure 3.3: A proof of $(\forall x)((\exists y)(y = f(x) \wedge (\forall z)(z = f(x) \supset y = z)))$.

50. $(\forall x)[pax \vee (\forall y)(p\,xy)] \supset (\exists x)(\forall y)(p\,xy)$ (4pts)

Two appealing problems are Schubert's Steamroller and the Dreadsbury Mansion Murder Mistery.[6] Schubert's Steamroller is a problem that can be formulated in first-order logic, and is difficult for first-order automated theorem provers. It reads like this:

47. "Wolves, foxes, birds, caterpillars, and snails are animals, and there are some of each of them. Also, there are some grains, and grains are plants. Every animal either likes to eat all plants or all animals much smaller than itself that like to eat some plants. Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which in turn are much smaller than wolves. Wolves do not like to eat foxes or grains, while birds like to eat caterpillars but not snails. Caterpillars and snails like to eat some plants.

    Is there an animal that likes to eat a grain-eating animal?" (10pts)

    A possible translation to the language of first-order logic is:

$(\exists x)\,Wolf(x)$	$(\forall x)(Wolf(x) \rightarrow Animal(x))$
$(\exists x)Fox(x)$	$(\forall x)(Fox(x) \rightarrow Animal(x))$
$(\exists x)Bird(x)$	$(\forall x)(Bird(x) \rightarrow Animal(x))$
$(\exists x)Caterpillar(x)$	$(\forall x)(Caterpillar(x) \rightarrow Animal(x))$
$(\exists x)Snail(x)$	$(\forall x)(Snail(x) \rightarrow Animal(x))$
$(\exists x)Grain(x)$	$(\forall x)(Grain(x) \rightarrow Plant(x))$

[6] No, not the German composer Franz Schubert (1797-1828).

$(\forall x, y)(Caterpillar(x) \wedge Bird(y) \rightarrow Smaller(x, y))$
$(\forall x, y)(Snail(x) \wedge Bird(y) \rightarrow Smaller(x, y))$
$(\forall x, y)(Bird(x) \wedge Fox(y) \rightarrow Smaller(x, y))$
$(\forall x, y)(Fox(x) \wedge Wolf(y) \rightarrow Smaller(x, y))$

$(\forall x, y)(Bird(x) \wedge Caterpillar(y) \rightarrow Eats(x, y))$
$(\forall x, y)(Wolf(x) \wedge Fox(y) \rightarrow \neg Eats(x, y))$
$(\forall x, y)(Wolf(x) \wedge Grain(y) \rightarrow \neg Eats(x, y))$
$(\forall x, y)(Bird(x) \wedge Snail(y) \rightarrow \neg Eats(x, y))$

$(\forall x)(Caterpillar(x) \rightarrow ((\exists y)(Plant(y) \wedge Eats(x, y))))$
$(\forall x)(Snail(x) \rightarrow ((\exists y)(Plant(y) \wedge Eats(x, y))))$

All animals either eat all plants or eat all smaller animals that eat some plants:

$$(\forall x)(Animal(x) \rightarrow (\forall y)(Plant(y) \rightarrow Eats(x, y)) \vee (\forall z)(Animal(z) \wedge Smaller(z, x) \wedge (\exists u)(Plant(u) \wedge Eats(z, u)) \rightarrow Eats(x, z)))$$

Question:

$(\exists x, y)(Animal(x) \wedge Animal(y) \wedge (\exists z)[Eats(x, y) \wedge Eats(y, z) \wedge Grain(z)])$?

55. Someone who lives in Dreadsbury Mansion killed Aunt Agatha. Agatha, the butler, and Charles live in Dreadsbury Mansion, and are the only people who live therein. A killer always hates his victim, and is never richer than his victim. Charles hates no one that Aunt Agatha hates. Agatha hates everyone except the butler. The butler hates everyone not richer than Aunt Agatha. The butler hates everyone Agatha hates. No one hates everyone. Agatha is not the butler. Therefore, Agatha killed herself. (8pts)

$(\exists x)(Lives(x) \wedge Killed(x, agatha))$
$Lives(agatha) \wedge Lives(butler) \wedge Lives(charles)$
$(\forall x)(Lives(x) \rightarrow (x = agatha \vee x = butler \vee x = charles))$
$(\forall x, y)(Killed(x, y) \rightarrow Hates(x, y))$
$(\forall x, y)(Killed(x, y) \rightarrow \neg Richer(x, y))$
$(\forall x)(Hates(agatha, x) \rightarrow \neg Hates(charles, x))$
$(\forall x)(\neg(x = butler) \rightarrow Hates(agatha, x))$
$(\forall x)(\neg Richer(x, agatha) \rightarrow Hates(butler, x))$
$(\forall x)(Hates(agatha, x) \rightarrow Hates(butler, x))$
$(\forall x)(\exists y)(\neg Hates(x, y))$
$\neg(agatha = butler)$

Björn:	$Killed(charles, agatha)$?
Reidar:	$Killed(agatha, agatha)$?
Olaf:	$\neg Killed(butler, agatha)$?

Chapter 4

Clause sets

Tableaux and sequent calculi are probably the most intuitive automated theorem proving techniques. Yet, when it comes to automated proof search, tableaux-based theorem provers are often slower than other theorem provers, and often even slower than model-finders such as GSAT. The current state of affairs in ATP-land is that resolution is still the fastest and most powerful technique available. State-of-the-art theorem provers such as OTTER for example, use hyperresolution as their main inference rule. Also, most theorem provers that participate in the CADE ATP System Competition are based on resolution principles and/or the manipulation of clause sets. Finally, influential logic programming languages, such as Prolog, use SLD-resolution, or other variants of linear resolution, as their main inference rule. Thus, resolution is a practical and therefore important ATP technique.

Resolution-based theorem provers operate on clauses and on clauses only. Therefore, a firm knowledge of clause sets is necessary to understand the essentials of resolution-based theorem provers.

4.1 Normal forms

Clause sets are set-representations of conjunctive normal forms. Although you are probably familiar with normal forms, I hope (and actually expect!) that the following section still shows you a number of new things.

Disjunctive normal forms

A *disjunctive normal form* is a disjunction of conjunctions of literals. Examples:

1. $(p \wedge \neg q) \vee (\neg p \wedge r \wedge s) \vee (\neg r \wedge \neg s)$
2. $(p \wedge \neg q) \vee (\neg p \wedge r \wedge s) \vee (\neg r \wedge r)$
3. $(p \wedge \neg q) \vee (\neg p \wedge r \wedge s) \vee \neg r$
4. $(p \wedge \neg q) \vee (\neg p \wedge r \wedge s)$
5. $p \vee \neg r$
6. $p \wedge \neg q$

DNF (1), (2) and (3) consist of three terms, DNF (4) and (5) consist of two terms, and DNF (6) consists of one term.

You have probably noted that (5) and (6) are peculiar DNFs. If we continue to use parenthesis as we did in (1-4), then (5) would be written $(p) \vee (\neg r)$, while (6) would be written $(p \wedge \neg q)$.

i. Another example of a peculiar DNF is the *empty* DNF, which is a DNF with zero terms.

ii. Because a DNF consists of disjunctions, it can be made true if and only if one of its terms can be made true. I.e., a DNF is satisfiable if and only if one of its terms satisfiable.

iii. By (i) and (ii), the empty DNF is false.

iv. Because every term of a DNF is a conjunction, a term of a DNF is satisfiable if and only if it does not contain two complementary literals.

v. By (ii) and (iv), a DNF is satisfiable if and only if it has a term without complementary literals. Hence, it is easy to check whether a DNF is satisfiable. (More precisely, DNFs can be checked for satisfiability in polynomial time.)

Every proposition can be written in DNF. There exist several methods. Some are intuitive but not very efficient, and some are efficient but not very intuitive. Probably the most intuitive and well-known method is the one that makes use of truth-tables.

Constructing a DNF with truth-tables

Suppose we want to write $\phi = (p \wedge q) \supset \neg(q \vee \neg p)$ in DNF. From ϕ's truth-table, then,

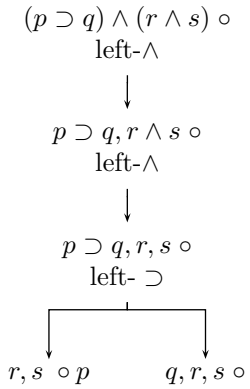
(	p	$\wedge$	q	)	$\supset$	$\neg$	(	q	$\vee$	$\neg$	p	)
	0	0	0		1	1		0	1	1	0	
	0	0	1		1	0		1	1	1	0	
	1	0	0		1	1		0	0	0	1	
	1	1	1		0	0		1	1	0	1	

we see that the first three rows make ϕ true, while the last row makes ϕ false. Hence, $(\neg p \wedge \neg q) \vee (\neg p \wedge q) \vee (p \wedge \neg q)$ is a DNF of ϕ. Probably not the shortest one, but the point here is to show that we can construct a DNF in a systematic fashion. Finding a shortest DNF is quite another matter. In this case, a shorter DNF would be $\neg p \vee (p \wedge \neg q)$.

Constructing a DNF with analytic tableaux

Working with truth-tables is of course unattractive, not to say boring. Another (and perhaps more attractive) method to compute DNFs, is by means of analytic tableaux (Chapter 2, Table 2.2 on page 31).

Example 4.1 Suppose that we would like to compute the DNF of $\phi = (p \supset q) \wedge (r \wedge s)$. This can be done by by putting ϕ on the LHS of the "$\circ$" on the top of an analytic tableaux. Thus, we try to make ϕ true. To investigate which atoms make ϕ true, we perform the standard tableaux procedure all the way down to the bare atoms:



Now ϕ is true if and only if one of the leaves is made true. Thus, ϕ is true if and only if $r \wedge s \wedge \neg p$ is true, or $q \wedge r \wedge s$ is true. Hence, $\phi \equiv (r \wedge s \wedge \neg p) \vee (q \wedge r \wedge s)$. This method only works on the proviso that we complete all branches.

Constructing a DNF by factorization

Yet another method for generating DNFs is eliminating all "$\supset$," "$\equiv$," and then eliminating all parentheses. In more detail:

1. Eliminate "$\supset$" and "$\equiv$" by applying the rewrite rules $A \supset B \to \neg A \vee B$ and $A \equiv B \to A \wedge B \vee \neg A \wedge \neg B$. (No parenthesis needed for the last one, so that's easy.)
2. Eliminate parenthesis by repeatedly applying distributive laws for operators with higher precedence:

$$\begin{aligned} A \wedge (B \vee C) &\to A \wedge B \vee A \wedge C \\ (A \vee B) \wedge C &\to A \wedge C \vee B \wedge C \\ \neg(A \wedge B) &\to \neg A \vee \neg B \\ \neg(A \vee B) &\to \neg A \wedge \neg B \\ \neg\neg A &\to A \end{aligned}$$

Eventually, you end up with a an expression such as for example

$$a_1 \wedge \neg a_2 \wedge \neg a_3 \vee a_4 \wedge a_5 \vee \neg a_6 \wedge a_7 \vee \ldots .$$

This expression is in DNF, because "$\wedge$" has priority over "$\vee$," which becomes more clear if we write it thus:

$$(a_1 \wedge \neg a_2 \wedge \neg a_3) \vee (a_4 \wedge a_5) \vee (\neg a_6 \wedge a_7) \vee \ldots .$$

Note that this procedure can be automated. (Which is the whole point, after all.)

Conjunctive normal forms

A *conjunctive normal form* (CNF) is a conjunction of disjunctions of literals.

i. A particular example of a CNF is the *empty* CNF, which is a one with zero terms.

ii. Because a CNF consists of conjunctions, a CNF is true if and only if all its terms are true.

iii. By (i) and (ii), the empty CNF is vacuously true.

iv. Because every term of a CNF is a disjunction, a term of a CNF is true if and only if it contains two complementary literals.

v. By (ii) and (iv), a CNF is a tautology if and only all terms possess complementary literals. It is therefore easy to check whether a CNF is a tautology. (More precisely, for CNFs a tautology check can be performed in polynomial time.)

To see that every proposition can be written in CNF, let ϕ be an arbitrary proposition. We already know that every proposition can be written in DNF, so in particular we can write $\neg\phi$ in DNF:

$$\neg\phi = (a_1 \wedge a_2 \wedge \ldots) \vee (b_1 \wedge b_2 \wedge \ldots) \vee \ldots$$

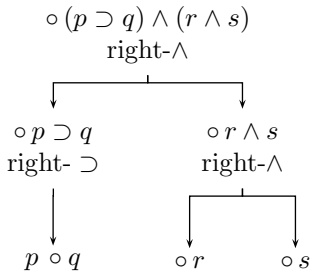
By means of The Morgan's laws [which are: $\neg(a \wedge b) \equiv \neg a \vee \neg b$ and $\neg(a \vee b) \equiv \neg a \wedge \neg b$]:

$$\begin{aligned} \phi &\equiv \neg[\neg\phi] \\ &\equiv \neg[(a_1 \wedge a_2 \wedge \ldots) \vee (b_1 \wedge b_2 \wedge \ldots) \vee \ldots] \\ &\equiv \neg(a_1 \wedge a_2 \wedge \ldots) \wedge \neg(b_1 \wedge b_2 \wedge \ldots) \wedge \neg(\ldots) \wedge \ldots \\ &\equiv (\neg a_1 \vee \neg a_2 \vee \ldots) \wedge (\neg b_1 \vee \neg b_2 \vee \ldots) \wedge \ldots \end{aligned} \tag{4.1}$$

Removing double negations from (4.1) yields ϕ in CNF.

In Example 4.1 on page 82 it was demonstrated how a DNF can be computed with the help of analytic tableaux. CNFs can be computed by means of analytic tableaux as well.

Example 4.2 (Constructing a CNF with analytic tableaux) Suppose that we would like to compute the CNF of $\phi = (p \supset q) \wedge (r \wedge s)$. This can be done by by putting ϕ on the RHS of the "$\circ$" on the top of an analytic tableaux. Thus, we try to falsify ϕ.



All leaves that represent counterexamples must be denied to prevent ϕ from being false. Thus, $(\neg p \vee q) \wedge r \wedge s$ is a CNF for ϕ. Note that there may be more CNFs, and there may even be *shorter* CNFs.

Again, this method only works on the proviso that we complete all branches.

Clause sets

It is advantageous to write CNFs as *clause sets*. In sets, literals do not have an order and multiple occurrences of literals "melt" together into a single literal. In particular we do not need to apply explicit laws of associativity, commutativity or idempotency to simply logical expressions. We all get this for free with the clause set notation.

For example, the clause set on the RHS of

$$\begin{aligned}(p \vee \neg q) \wedge (\neg p \vee q \vee \neg r) &\equiv \{p \vee \neg q, \neg p \vee q \vee \neg r\} \\ &\equiv \{\{p, \neg q\}, \{\neg p, q, \neg r\}\}\end{aligned}$$

represents the conjunction of two clauses, viz. $\{p, \neg q\}$ and $\{\neg p, q, \neg r\}$. Each clause represents a disjunction of literals.

The *empty clause* is denoted by $\Box$. Since a clause represents a disjunction, and since a disjunction is true if and only if one of its components is true, we have $\Box \equiv$ false. Similarly, if $\varnothing$ is regarded as an empty clause set, we have $\varnothing \equiv$ true.

Complexity of rewriting into normal form

Rewriting formulas in normal form is conceptually simple, but computationally hard. Thus, there exist simple algorithms that describe how to compute CNFs, but the actual execution of such algorithms may take exponential time in the length of the formula. Whether an exponential blow-up indeed occurs, depends on the structure of the formula. (For DNFs, see Exercise 4 on page 87; for CNFs, see Exercise 5 on page 88.)

There is another way to see why rewriting formulas in normal form is computationally hard. Take DNF, for instance. The existence of a simple DNF algorithm (read: a DNF-algorithm polynomial in the length of its input), would imply the existence of a simple theorem proving algorithm, as follows. To check whether $\phi_1, \ldots, \phi_n \vdash \phi$:

1. Form $(\phi_1 \wedge \ldots \wedge \phi_n) \wedge \neg\phi$. (Because $\phi_1, \ldots, \phi_n \vdash \phi$ is true iff $(\phi_1 \wedge \ldots \wedge \phi_n) \wedge \neg\phi$ is not satisfiable.)
2. Rewrite $(\phi_1 \wedge \ldots \wedge \phi_n) \wedge \neg\phi$ in DNF.
3. If all conjunctions in the DNF contain complementary literals, then the DNF is unsatisfiable and $\phi_1, \ldots, \phi_n \vdash \phi$. Else, the DNF is satisfiable and $\phi_1, \ldots, \phi_n \nvdash \phi$.

This would suggest a polynomial-time procedure to test the validity of propositional inferences. However, by an argument that involves the NP-completeness of testing the satisfiability of 3SAT-clause sets, it is generally believed that such a procedure does not exist. See further point (v) on page 82.

Conversion to $\leq$ 3CNF in linear time

It is impossible to rewrite an arbitrary formula into an equivalent CNF in linear time (Exercise 5 on page 88). Hence, it is impossible to linearly rewrite a formula into an equivalent clause set. This fact is hard to live with, because resolution works with clause sets, and for most formulas we may not assume that they are already in the form of a clause set.

However, there is a solution. This solution suggests itself if we observe that the essential invariant of resolution is satisfiability, rather than logical equivalence. With refutation methods, we would like to know whether the original formula ϕ, say, can be falsified—not whether it can be proven equivalent. This way of looking at clause sets turns out to be the right one, because in a moment we will see that it is possible to produce a clause set in linear time that is satisfiable if and only if ϕ is satisfiable. Computing such a clause set is done by computing the so-called *Tseitin-derivative* of ϕ. The Tseitin-derivative consists of a conjunction of equivalences, where the LHS of each equivalence is a proposition letter that corresponds to a sub-formula and the RHS of each equivalence is a conjunction, negation, implication, or disjunction of other sub-formulas representing proposition letters, conjunctions in the Tseitin-derivate correspond to conjunctions in ϕ, negations in the Tseitin-derivate correspond to negations in ϕ, and so forth.

Example 4.3 (Tseitin-derivative) Let

$$\phi = (\neg q) \wedge (p \supset ((\neg q)) \vee p\,).$$

First we make a table of sub-formulas, in which every compound subformula is associated with a fresh proposition letter. The table is built up in a top-down fashion (largest formulas first):

Subformula	*New proposition letter*
ϕ	r
$\neg q$	s
$p \supset ((\neg q)) \vee p\,)$	t
$(\neg q) \vee p$	u
$\neg q$	v

The sub-formula $\neg q$ occurs twice, so that it corresponds to two different variables, viz. s and v. We could take these two variables together, but two is okay for now. The Tseitin-derivative of ϕ is now

$$TS(\phi) = r \wedge (r \equiv s \wedge t) \wedge (s \equiv \neg q) \wedge (t \equiv p \supset u) \wedge (u \equiv v \vee p\,) \wedge (v \equiv \neg q). \tag{4.2}$$

If v is identified with s (which is not compulsory), the Tseitin-derivative can be shortened into

$$r \wedge (r \equiv s \wedge t) \wedge (s \equiv \neg q) \wedge (t \equiv p \supset u) \wedge (u \equiv s \vee p\,).$$

If the Tseitin-derivative is computed in a purely mechanical fashion, without further simplifications, it is unique.

Remarks:

1. Do not make the mistake to suppose that the Tseitin-derivative is an equivalent rewrite of ϕ in $\leq$3CNF. This is not the case. Actually such a rewrite is impossible, since it can be shown that it is impossible to rewrite a formula to CNF (*a fortiori*, in $\leq$3CNF) in linear time while maintaining validity. (This is demonstrated in [30].) We have only produced a $\leq$ 3CNF that is satisfiable if and only if ϕ is satisfiable.
2. The size of the set of new proposition letters depends linearly on the size of ϕ. (Exercise 6 on page 29.) This fact is important because it ensures that producing a Tseitin-derivation is a process that linearly depends on the size of the original formula.
3. If a sub-formula occurs positively in ϕ, i.e., in the scope of an even number of negations, then, maintaining satisfiability, the $\equiv$ in the formula above can be replaced by $\supset$. If a sub-formula occurs negatively, i.e., in the scope of an odd number of negations, then $\equiv$ can be replaced by $\subset$ (implication from right to left). Experience has shown that replacing equivalences by implications (sometimes) leads to a considerable gain in efficiency in the theorem proving procedure that follows, especially when the formula is satisfiable.

Subformula	*CNF (in clause set notation)*
$r \equiv s \wedge t$	$\{\{r, \neg s, \neg t\}, \{\neg r, s\}, \{\neg r, t\}\}$
$s \equiv \neg q$	$\{\{s, q\}, \{\neg s, \neg q\}\}$
$t \equiv p \supset u$	$\{\{p, t\}, \{t, \neg u\}, \{\neg p, \neg t, u\}\}$
$u \equiv v \vee p$	$\{\{u, \neg v\}, \{u, \neg p\}, \{\neg u, v, p\}\}$
$v \equiv \neg q$	$\{\{v, q\}, \{\neg v, \neg q\}\}$

Table 4.1: Reducing equivalences to $\leq$3CNF clause sets

The next step is to transform each equivalence to a CNF using Table 4.1. Collecting the separate CNFs from (4.2) into one big CNF yields

$$\begin{aligned} TS(\phi) = \{\{r\}\} \cup \{&\{r, \neg s, \neg t\}, \{\neg r, s\}, \{\neg r, t\}, \\ &\{s, q\}, \{\neg s, \neg q\}, \\ &\{p, t\}, \{t, \neg u\}, \{\neg p, \neg t, u\}, \\ &\{u, \neg v\}, \{u, \neg p\}, \{\neg u, v, p\}\}, \\ &\{v, q\}, \{\neg v, \neg q\}\} \end{aligned}$$

This CNF has exactly the same satisfying assignments as the original formula ϕ, provided assignments are compared with respect to the variables that occur in both formulas. (Which is reasonable, if you come to think of it: it is not hard to prove that the truth value of variables that do not occur in ϕ are irrelevant to the truth value of ϕ.)

The rest of this chapter is concerned with resolution and, more generally, the manipulation of clause sets. Resolution is an inference process on clause sets that takes a number of clauses (from the clause set) to infer a new clause; if the inferred clause is new, interesting, and/or useful, it is kept (official terminology: *retained*). Other techniques that manipulate clause sets are not really resolution techniques, but are often meant to keep clause sets small and non-redundant.

The idea behind resolution is to prove a theorem by proving that the negation of that theorem is inconsistent with the assumptions on the basis of which that theorem is proven.

Example 4.4 (The idea behind resolution) Suppose that we would like to know whether

$$\neg p,\ (\neg q \wedge r) \supset p,\ \neg q \vdash \neg r. \tag{4.3}$$

Thus, we would like to prove $\neg r$ on the basis of $\neg p$, $(\neg q \wedge r) \supset p$, and $\neg q$. This is the same as proving the inconsistency of

$$\{\neg p,\ (\neg q \wedge r) \supset p,\ \neg q\} \cup \{r\},$$

which amounts to proving the inconsistency of

$$(\neg p) \wedge ((\neg q \wedge r) \supset p) \wedge (\neg q) \wedge r. \tag{4.4}$$

Rewriting (4.4) in CNF yields

$$(\neg p) \wedge (p \vee q \vee \neg r) \wedge (\neg q) \wedge r,$$

which is equivalent with the clause set

$$S = \{\{\neg p\}, \{p, q, \neg r\}, \{\neg q\}, \{r\}\}. \tag{4.5}$$

The objective is to show that (4.5) is inconsistent (or unsatisfiable, which is the same for that matter). This is done by adding clauses to S and deleting clauses from S *such that satisfiability of S remains invariant*. If, somewhere in the process, the empty clause $\Box$ is added to S, we know that S must have been unsatisfiable, so that (4.3) follows.

PROBLEMS (Sec. 4.1)

1. Prove that the truth value of variables that do not occur in ϕ are irrelevant to ϕ's truth value.

2. Use the refutation method to compute a DNF and CNF for each of the following formulas. The refutation method is explained in Example 4.1 on page 82 (DNF) and Example 4.2 on page 84 (CNF).

   (a) $p \supset (q \supset p)$
   (b) $a \wedge (b \wedge (c \vee d))$
   (c) $\neg(\neg(m))$
   (d) $p \vee \neg p$
   (e) $p \wedge \neg p$

3. Construct refutation trees for the formulas listed at (2).

   (a) Give a DNF and CNF for every formula that is not refutable. (Easy!)
   (b) If a formula is refutable, construct a DNF and a CNF out of the non-axiom leaves of the refutation tree.
   (c) Give an algorithm based on refutation trees to construct a DNF (or CNF).
   (d) Explain why the algorithm works and always produces the right answers.

4. Consider the formula

   $$(a_1 \vee a_2) \wedge \cdots \wedge (z_1 \vee z_2). \tag{4.6}$$

   This formula has length $26c$, for some fixed constant c. Strictly speaking, the letter c is now overloaded, since it is supposed to represent a proposition letter as well as a positive integer. (But if you've come this far, I guess you are able to deal with this much ambiguity as well.)

   (a) Determine c.
   (b) Determine a shortest DNF of (4.6). (Hint: first try to construct a DNF for $(a_1 \vee a_2) \wedge (b_1 \vee b_2)$, and then try to generalize.) How do you know it is the shortest DNF?
   (c) What is the length of the formula found at (4b)?

(d) Generalize your answer if the number of basic literals is n, rather than 26.

(e) What does the result tell you about the conversion from CNF to DNF?

(f) Draw a general conclusion from the previous answer.

5. Consider the formula

$$(a \wedge \ldots \wedge z) \vee (\neg a \wedge \ldots \wedge \neg z) \tag{4.7}$$

(a) Same questions as with (4).

(b) Try to produce a combinatorial argument that explains why the conversion from (4) to CNF is exponential. (Solution.)

6. Describe a polynomial time algorithm for determining whether a DNF is satisfiable.

7. Describe a polynomial time algorithm for determining whether a CNF is a tautology.

8. Someone claims that propositional satisfiability can be verified in polynomial time, by a conversion to DNF, and then verifying the satisfiability of the DNF in polynomial time. What is wrong with this argument?

9. Which of the following are clauses? Which are clause sets? Give a corresponding CNF if possible.

(a)	$\{\{p\}\}$	(e)	$\varnothing$
(b)	$\{\Box\}$	(f)	$\{\{\Box\}\}$
(c)	$\{p, \{\neg q\}\}$	(g)	$\Box$
(d)	$\{\Box, \{, \neg q\}\}$	(h)	$\{\{\neg p, p\}, \{\neg p, p, q\}, \Box\}$

(Solution.)

10. Consider

$$\begin{array}{l} p \wedge \neg r \wedge s \rightarrow t \\ \neg t \rightarrow p \\ \neg r \wedge q \rightarrow \neg s \\ q \rightarrow \neg p \end{array}$$

where "$\rightarrow$" represents "$\supset$". Rewrite these rules into a clause set. (Solution.)

11. Rewrite the following formulas as $\leq$3CNF clause sets.

(a) $a \wedge b \wedge c \wedge d \wedge e$ (Solution.)

(b) $a \vee b \vee c \vee d \vee e$ (Solution.)

12. Describe how the equivalence of (2.6) and (2.7) on page 39 can automatically be proven by applying a Tseitin transformation and resolution. What is the number of clauses in the Tseitin transformation? Note that you don't actually have to spell out the Tseitin transformation to answer this question. (Solution.)

13. Someone claims that computing a Tseitin-derivative makes no sense. His (or her) argument is that, although the conversion itself can be done in linear time, it introduces proposition letters that linearly depend on the length of the original formula. The problem is that every additional propositional letter doubles the search space of the resolution algorithm. What is your comment? (Solution.)

4.2 Techniques to clean up and simplify clause sets

It is perhaps a bit odd to put a section like this up front in a chapter on resolution: what is the point of knowing how to simplify clause sets if we don't know how to make new clause sets in the first place? Still, there is a point. Usually the initial clause set can be simplified without applying any resolution rule of some sort. Further, resolution-based theorem provers produce a lot of dirt (i.e., useless clauses) that must be removed between resolution steps proper (Figure 4.1.) By nature, resolution is a process in which a large number of clauses is produced. If

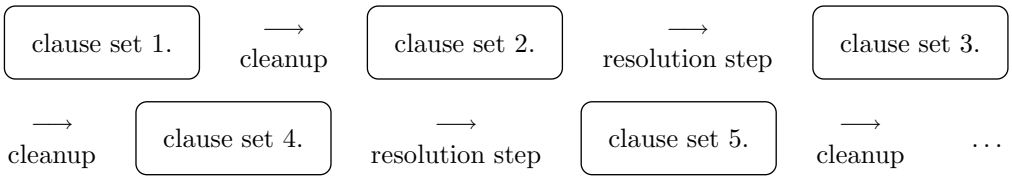


Figure 4.1: Main loop of a resolution-based theorem prover.

we do nothing between intermediate steps to reduce the size of the clause set, a theorem prover would within no time inspecting hundreds of thousands of redundant clauses. Therefore, a good theorem prover spends time to distinguish important from unimportant (or redundant) clauses. Thus, cleaning up a clause set makes sense even before the resolution process itself begins. Hence the place of this section. Below, a number of simplification techniques are discussed.

The one-literal rule

The by far most important technique to simplify clause sets is the *one-literal rule* (OLR). This rule (not to be confused with unit resolution, p. on page 101) can be applied as soon as there are unit clauses.

To appreciate the benefits of the OLR, consider

$$F = p \wedge (p \vee \phi) \wedge (\neg p \vee \psi) \wedge \chi \tag{4.8}$$

where ϕ, ψ and χ are $\{p, \neg p\}$-free. Obviously, F corresponds to a clause set of the form $\{\{p\}, \{p\} \cup A, \{\neg p\} \cup B, C\}$.

It is not difficult to show that (4.8) is satisfiable if and only if the formula $\psi \wedge \chi$ is satisfiable: if m is a model such that $m \models F$, then $m \models p$, $m \models \neg p \vee \psi$ and $m \models \chi$, so that $m \models \psi \wedge \chi$. Conversely, if ψ and χ are $\{p, \neg p\}$-free and $m \models \psi \wedge \chi$, then also $m_p \models \psi \wedge \chi$, where m_p is the model that is identical to m except possibly for p, where m_p makes p true. This is possible because ψ and χ are $\{p, \neg p\}$-free. Clearly, $m_p \models p \wedge (p \vee \phi) \wedge (\neg p \vee \psi) \wedge \chi$.

Thus, every clause set of the form

$$\{\{p\}, \{p\} \cup A, \{\neg p\} \cup B, C\} \tag{4.9}$$

may be reduced to

$$\{B, C\}. \tag{4.10}$$

The transition from (4.9) to (4.10) does not respect logical equivalence, but is satisfiability-equivalent.

The general case amounts to

$$F = p \wedge (p \vee \phi_1) \wedge \ldots \wedge (p \vee \phi_m) \wedge (\neg p \vee \psi_1) \wedge \ldots \wedge (\neg p \vee \psi_n) \wedge \chi_1 \wedge \ldots \wedge \chi_r \tag{4.11}$$

where all the ϕ_i's, ψ_j's and χ_k's are $\{p, \neg p\}$-free.
As above, it can be shown that

F is satisfiable if and only if $\psi_1 \wedge \ldots \wedge \psi_n \wedge \chi_1 \wedge \ldots \wedge \chi_r$ is satisfiable.

Thus, every clause set of the form

$$\{\{p\}, \{p\} \cup A_1, \ldots, \{p\} \cup A_m, \{\neg p\} \cup B_1, \ldots, \{\neg p\} \cup B_n, C_1, \ldots, C_r\} \tag{4.12}$$

may be reduced to

$$\{B_1, \ldots, B_n, C_1, \ldots, C_r\}. \tag{4.13}$$

respecting satisfiability-equivalence.

Example 4.5 The clause set of Example 4.5 on page 87 can be solved entirely by repeatedly applying the OLR. This is done on $\neg p$, $\neg q$ and r, respectively:

$\{\{\neg p\}, \{p, q, \neg r\}, \{\neg q\}, \{r\}\}$	OLR with $\neg p$
$\{\{q, \neg r\}, \{\neg q\}, \{r\}\}$	OLR with $\neg q$
$\{\{\neg r\}, \{r\}\}$	OLR with r
$\{\square\}$	

Notice that the empty clause $\square$ stems from $\{\neg r\}$. If the one-literal rule was applied with $\neg r$, the empty clause would stem from $\{r\}$.

The OLR is extremely valuable, because it simplifies clause sets. Another valuable property of the OLR is that it may introduce other unit clauses which on their turn may further simplify the clause set.

Monotone variable fixing

Another simple but valuable technique is to verify whether every literal of every clause is complemented by a literal in some other clause. If not, the entire clause is useless and can be removed from the clause set. It is of course possible that more clauses can be deleted this way.

The validity of monotone variable fixing rests on the fact that

$\phi_1 \wedge \ldots \wedge \phi_m \wedge (\psi_1 \vee p) \wedge \ldots \wedge (\psi_n \vee p)$ is satisfiable iff $\phi_1 \wedge \ldots \wedge \phi_m$ is satisfiable,

provided that $\phi_1, \ldots, \phi_n$ are $\neg p$-free. This maneuver is known as *monotone variable fixing*. (I leave it up to you to determine which variable is fixed.) The adjective "monotone" refers to the irreversibility of the process: once a variable is fixed, its truth value is determined and does not change in the rest of the process. Monotone variable fixing, also known as the *pure literal rule* (PLR) [13], is often not included in current state-of-the-art implementations, because it slows down the algorithm [8, 39].

Example 4.6 Consider

$$S = \{\{p, \neg q, t\}, \{\neg q, r, \neg s\}, \{\neg t, \neg u, \}, \{r, \neg s, \neg q\}, \{p, u, \neg v\}\}$$

The literals p, $\neg q$, r, $\neg s$, and $\neg v$ occur "uncomplemented" in S, and can therefore be removed by monotone variable fixing. If p is set to true and q to false, we obtain

$\{\underline{\{p, \neg q, t\}}, \{\neg q, r, \neg s\}, \{\neg t, \neg u, \}, \{r, \neg s, \neg q\}, \underline{\{p, u, \neg v\}}\}$	set $p = 1$
$\{\underline{\{\neg q, r, \neg s\}}, \{\neg t, \neg u, \}, \underline{\{r, \neg s, \neg q\}}\}$	set $q = 0$
$\{\underline{\{\neg t, \neg u, \}}\}$	set $u = 0$
$\varnothing$.	

We see that in the process of deleting clauses, other literals (in this case $\neg u$) may become uncomplemented, so that the last clause may also be deleted by setting u to false.

Monotone variable fixing preserves satisfiability-equivalence but not necessarily logical equivalence.

Tautology rule

Based on the equivalence $(p \vee \neg p \vee \phi) \wedge \psi \equiv \psi$, clauses with complementary literals, i.e., tautologies, may be removed from the clause set without compromising logical equivalence.

Subsumption

If C_i and C_j are two clauses in a clause set $\{C_1, \ldots, C_n\}$ such that $C_i \subset C_j$, we say that C_i *subsumes* C_j, or that C_j is *subsumed by* C_i. The dependent clause C_j may be deleted on the basis of the fact that $C_i \vdash C_j$, so that

$$C_1 \wedge \ldots \wedge C_n \equiv C_1 \wedge \ldots \wedge C_{j-1} \wedge C_{j+1} \wedge \ldots \wedge C_n.$$

With clause sets, subsumption is equivalent to the subset relation. But as every programmer knows, computers do not excel at set manipulation. The most common technique to represent sets is by means of bit vectors. A bit vector is a string of bits, where each bit says whether a particular variable is an element of that set. Subset relations between sets can then be computed by masking both vectors by bitwise logical "and" and then see whether one of the vectors remains invariant. (Based on the equivalence $A \subset B$ iff $A \cap B = A$ and $A \neq B$.) For example, $v_1 =$ "00101" represents a proper subset of the set that is represented by $v_2 =$ "01111", since $v_1 \& v_2 = v_1 \neq v_2$. Still, testing clause sets on dependent sets is computationally expensive, so that theorem provers in are typically conservative with the use subsumption checks.

4.3 The Davis-Putnam/Logemann-Loveland algorithm

Monotone variable fixing, subsumption, and other simplification techniques are of great help in cleaning up clause sets, but in itself these techniques are incomplete.

One particularly easy straightforward way to enforce completeness is to pick an arbitrary proposition variable, and to apply the one-literal rule to this variable in both directions. This maneuver is know as the *splitting rule* [13]. With the splitting rule, we pick an arbitrary proposition, p, and distinguishes two cases: one in which p is true, and one in which $\neg p$ is true. Such a distinction is necessary, because we don't know whether p is true or false. (Otherwise, we would have applied the OLR with either p or $\neg p$.) The splitting rule rests on the equivalence $\phi \equiv (\phi \wedge p\,) \vee (\phi \wedge \neg p\,)$ and amounts to the following:

1. *Selection.* Select a literal p.
2. *Distinguishing cases.* Split the current clause set $C = \phi$ into a clause set $C_1 = \phi \cup \{p\}$ and the clause set $C_2 = \phi \cup \{\neg p\}$.
3. *Recursion.* Proceed with C_1. If C_1 is satisfiable, then C is satisfiable with $p = 1$ and halt. Else, proceed with C_2. If C_2 is satisfiable then C is satisfiable too, with $p = 0$. Else C is unsatisfiable.

The splitting rule may be applied repeatedly (recursively, if you wish), thus producing a binary tree of clause sets. One level in such a tree corresponds to a disjunction of clause sets (a

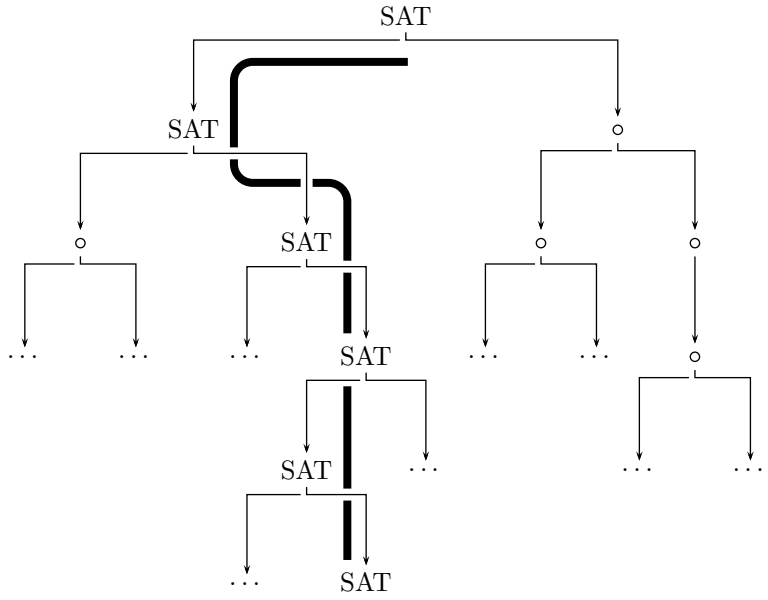


Figure 4.2: The splitting rule preserves satisfiability in at least one branch

disjunction of conjunctions of disjunctions!) and is sometimes called a *block*. Some logic textbooks works with blocks, but I think this is not a very intuitive approach.

Applying the splitting rule repeatedly is called the *DPLL algorithm*. This algorithm always works, which means that it always leads to the empty clause if a clause set is unsatisfiable, even without the use of simplification techniques. The price to be paid for completeness is that the DPLL algorithm splits one clause set into two clause sets and may thus lead to a combinatorial explosion of clause sets.

Example 4.7 (DPLL algorithm) Suppose we would like to investigate the satisfiability of

$$\{\{t\}, \{\neg t, \neg r\}, \{p, \neg r, s, \neg t\}, \{\neg p, q, r\}, \{\neg s, u, \neg u\}, \{p, \neg q, \neg t\}, \{p, t\}\}.$$

We might start thus:

$$\begin{array}{ll}
\{\{t\}, \{\neg t, \neg r\}, \{p, \neg r, s, \neg t\}, \ldots & \\
\qquad \ldots \{\neg p, q, r\}, \{\neg s, u, \neg u\}, \{p, \neg q, \neg t\}, \{p, t\}\} & \text{tautology rule with } \neg u \\
\{\{t\}, \{\neg t, \neg r\}, \{p, \neg r, s, \neg t\}, \{\neg p, q, r\}, \{p, \neg q, \neg t\}, \{p, t\}\} & \text{OLR with } t \\
\{\{\neg r\}, \{p, \neg r, s\}, \{\neg p, q, r\}, \{p, \neg q\}\} & \text{OLR with } \neg r \\
\{\{\neg p, q\}, \{p, \neg q\}\}. &
\end{array}$$

At this point we get stuck, because there is no unit clause that we may use to apply the OLR. So at this point we perform a split. It does not matter much whether we split with p or q, since both resulting clause sets are equally complex. Let us choose p. Then $C = \{\{\neg p, q\}, \{p, \neg q\}\}$ splits in to $C_1 = \{\{p\}, \{\neg p, q\}, \{p, \neg q\}\}$ and $C_2 = \{\{\neg p\}, \{\neg p, q\}, \{p, \neg q\}\}$. Proceeding with C_1 yields

$$\begin{array}{ll}
\{\{p\}, \{\neg p, q\}, \{p, \neg q\}\} & \text{OLR with } p \\
\{\{q\}\} &
\end{array}$$

so that we conclude that C_1 is satisfiable with $(p, q) = (1, 1)$. Hence, C is satisfiable with $(p, q) = (1, 1)$. Since we know that C is satisfiable, there is no further need to investigate C_2.

To see that the DPLL algorithm is sound and complete, look at Fig. 4.2 on the preceding page. It can easily be verified that, if a clause set is satisfiable, and the splitting rule is applied to it, then at least one of its children is also satisfiable. Conversely, if at least one of the children is satisfiable, then the original clause set was satisfiable as well. Further, every time the DPPL rule is applied, a variable is eliminated, so that the splitting rule may be applied at most n times, where n is the number of variables in the clause set. The clause sets that cannot be further reduced are either the empty set (which is satisfiable), or the clause set consisting of the empty clause (which is unsatisfiable). Soundness now follows from the fact that a satisfiable clause always has at least one satisfiable child. Completeness follows from the fact that the DPLL algorithm is finite and that all children of an unsatisfiable clause are unsatisfiable.

The splitting rule is widely regarded as an effective method for deciding on the satisfiability of a set of propositional clauses. It is not really a resolution rule but it does operate on clause sets. A disadvantage of the DPLL algorithm is that we might be forced to investigate the satisfiability of a number of different clause sets. In the worst case, we might be forced to investigate the satisfiability of 2^n different clause sets, where n is the number of different proposition letters.

4.4 The classic Davis-Putnam procedure

The Davis-Putnam procedure (DPP) is a predecessor of the splitting rule. It works like DPLL except that, if we commence a split, the two resulting clause sets are immediately merged back into one (usually big) clause set with the help of the distributive rules for $\wedge$ and $\vee$. The DPLL algorithm was presented as an improvement of the DPP [13, 57].

The DPP works as follows. First, clean up the clause set (using the one-literal rule, monotone variable fixing and possibly other techniques). Then select a literal p, say, for which we know that p and $\neg p$ occur hardly ever in the clause set. (Note this is a heuristic. See also section "Choice of branching variables," p. 94.) Write

$$\phi = \{\{p, C_1\}, \ldots, \{p, C_k\},\ \{\neg p, D_1\}, \ldots, \{\neg p, D_l\},\ E_1, \ldots, E_m\} \tag{4.14}$$

where $C_1, \ldots, C_k,\ D_1, \ldots, D_l$ are sequences of literals without p or $\neg p$, and $E_1, \ldots, E_m$ are clauses without p or $\neg p$. In fact, it is is recommended to select p such, that the product kl is as small as possible. We now distinguish two cases: one in which p is true and one in which $\neg p$ is true:

$$\{\{p, C_1\}, \ldots, \{p, C_k\},\ \{\neg p, D_1\}, \ldots, \{\neg p, D_l\},\ E_1, \ldots, E_m\}$$

$$p \qquad\qquad \neg p$$

$$\{D_1, \ldots, D_l, E_1, \ldots, E_m\} \qquad \{C_1, \ldots, C_k, E_1, \ldots, E_m\}$$

Thus, (4.14) is satisfiable if and only if the disjunction of the two children is satisfiable:

$$\begin{aligned} \psi &\equiv \{D_1, \ldots, D_l, E_1, \ldots, E_m\} \ \vee\ \{C_1, \ldots, C_k, E_1, \ldots, E_m\} \\ &\equiv ((D_1 \wedge \ldots \wedge D_l) \ \vee\ (C_1, \wedge \ldots \wedge C_k)) \ \wedge\ (E_1, \wedge \ldots \wedge E_m). \end{aligned} \tag{4.15}$$

Note that ϕ is generally not equivalent with ψ (for the same reason as why $(p \vee A) \wedge (\neg p \vee B)$ and $A \wedge B$ are not equivalent). With DPP, we merge the two clause sets immediately back into one clause set. This can be done by using the distributive laws of $\wedge$ and $\vee$. The disjunction (4.15) can then be rewritten into

$$= \begin{array}{ccc} \{\{C_1, D_1\}, & \ldots, & \{C_1, D_l\}, \\ \vdots & & \vdots \\ \{C_k, D_1\}, & \ldots, & \{C_k, D_l\}, E_1, \ldots, E_m\}. \end{array} \tag{4.16}$$

At this point, all occurrences of p and $\neg\, p$ have been eliminated, and the idea is that the clause set has been mixed up to such an extent that a number of simplification and deletion techniques can now be applied. (See page 89 and further.)

The DPP has a number of interesting properties.

i. It does away with proposition letters one at a time.

ii. The DPP, even without the usual simplification techniques, is complete. This can be seen by looking at the sound and completeness proof of the DPLL algorithm.

iii. The DPP changes a clause set with $k + l + m$ clauses into a clause set with $kl + m$ clauses. Therefore, it is often carried out only when the formula reduces in size, or there is a limited growth.

iv. If we start with $\leq$3CNF, the DPP often results in a $\leq$4CNF. In such cases, auxiliary proposition letters must be introduced to restore $\leq$3CNF. For example, if some C_i contains two literals and $\neg p$ occurs in a clause with three literals, then we will have to introduce an extra letter. Since p often occurs both positively and negatively, the resulting $\leq$3CNF is often considerably larger than the original, so we must be careful when to apply the DPP.

The classic Davis Putnam rule must not be confused with the "modern" variant of Davis, Longemann and Loveland [23]. The latter is currently far more popular but as we have seen, the splitting rule is essentially a mixture of the one-literal rule, monotone variable fixing and splitting.

Choice of branching variables

An important step of the DPLL algorithm is the choice of the variable on which we split. If good branching variables are chosen, the search tree is kept relatively small. Several studies are concerned with finding good branching rules.

A popular and cheap branching heuristic is MOM's heuristic. This picks the literal that occurs most often in the smallest clauses. (Ties are broken randomly or with a static ordering.) Sometimes, this is even considered too expensive and we simply branch on the first literal of a smallest clause.

The Jeroslow-Wang heuristic is another interesting strategy [47]. It estimates the contribution each literal is likely to make to satisfying the clause set. The estimated contribution of a literal l is computed by looking at the length $|C|$ of each clause C that l appears in. For each such clause, $2^{-|C|}$ is added to the estimated contribution. The split rule is then applied to the literal with the highest estimated contribution to satisfiability.

Most proposals can be divided in the following steps:

1. *Restrict*. Determine a set B of candidate-branching variables.
2. *Estimate*. For each $b \in B$, compute $f(b)$ and $f(\neg b)$, where f is some heuristic function that estimates the quality of branching on b. (For example, the size of the resulting clause sets.)

3. *Balance*. Compare, or balance the two values $f(b)$ and $f(\neg b)$, by means of some other heuristic function $g : R^2 \to R$.
4. *Choose*. Take $b \in B$ such that $g(f(b), f(\neg b))$ is maximal. Break ties if necessary.

Enhancements of the DPLL-algorithm include *lookahead unit resolution*, *intelligent backtracking* and the use of *elliptic approximations*. See further [126].

PROBLEMS (Sec. 4.4)

1. Suppose ϕ is a (cleaned up) clause set in which literals occur with the following frequency:

Literal	*frequency*	*literal*	*frequency*
p:	1	$\neg p$:	7
q:	3	$\neg q$:	5
r:	4	$\neg r$:	4
s:	6	$\neg s$:	2

   Which pair of literals would you, on the basis of this information, select to distinguish cases in the DPP? (Solution.)

2. Write the following disjunction of clause sets into one clause set. First, try to predict the answer. Then rewrite (4.17) into CNF (On page 83 it is described how this can be done.) Finally, argue that your answer might be considered as an informal verification of (4.16).

$$\{\{a, b\}, \{c, d\}\} \ \vee \ \{\{e, f\}, \{g, h\}\} \tag{4.17}$$

   (Solution.)

3. Let

$$\begin{aligned} S = \{&\{\neg r, \neg s\}, \{p, r\}, \{\neg s, t\}, \\ &\{\neg p, \neg r\}, \{s, r\}, \{q, s\}, \\ &\{\neg q, \neg t\}, \{p, t\}, \{\neg p, q\}\}. \end{aligned}$$

   (a) Show by means of classic Davis-Putnam that S is unsatisfiable. (Solution.)

   (b) Prove with DPLL that S is unsatisfiable. (Solution.)

4. Apply the DPP to the following clause sets:

   (a) $\{\{\neg a, b\}, \{\neg b, c\}, \{\neg c, d\}, \{\neg d\}\}$ (Solution.)

   (b) $\{\{a\}, \{\neg a, b\}, \{\neg b, \neg c, d\}, \{\neg d, e\}, \{e\}\}$ (Solution.)

   (c) $\{\{a, b, c\}, \{a, b, \neg c\}, \{a, \neg b, c\}, \{a, \neg b, \neg c\}\} \cup$
   $\{\{\neg a, b, c\}, \{\neg a, b, \neg c\}, \{\neg a, \neg b, c\}, \{\neg a, \neg b, \neg c\}\}$ (Solution.)

   (d) $\{\{\neg a, \neg b, c\}, \{\neg a, \neg c, d\}, \{a, b, d\}, \{\neg b, \neg d\}, \{\neg a, b, d\}\}$ (Solution.)

5. A special case is when a literal, say p, occurs only once in a clause set of length two, say $\{p, q\}$, while its complement may occur arbitrarily often. Show with DPP that we may substitute all occurrences of $\neg p$ by q, thus reducing the number of proposition letters by one, and reducing the length of the formula. (Solution.)

6. In the literature an alternative formulation of the DPP is known:

   Given a clause set

$$\phi = \{\{p, L_1\}, \ldots, \{p, L_n\}, C_1, \ldots, C_m\} \tag{4.18}$$

where L_i are sub-clauses (finite sequences of literals) and $C_1, \ldots, C_m$ are p-free (but not necessarily $\neg p$-free), then (4.18) can be reduced to

$$\{C_1[\neg p/L], \ldots, C_m[\neg p/L]\}$$

where L is the clause set $L_1 \wedge \ldots \wedge L_n$.

Show that his reduction is indeed an alternative formulation of the DPP. (Solution.)

Chapter 5

Resolution

An sympathetic property of theorem provers (and AR-systems in general) would be that single deduction steps are simple enough to be apprehended as correct by human beings in a single intellectual act. In this way, each single step of a deduction would be verifiable, even though the deduction as a whole may consist of a long chain of such steps. The previous chapters have shown that such theorem provers exist. Unfortunately, a downside of using small inference steps is that it causes the search space to combinatorially explode. A case in point are the DPP and DPLL-procedure which in 1950/1960 were state-of-the art theorem-proving techniques. A major breakthrough was made in 1965 by John Alan Robinson, who approached the matter from a more combinatorial point of view [96]. Robinson came to see how several ideas of Davis, Putnam and others could be combined into a new and far more efficient ATP for first-order predicate logic, now known as resolution.

Figure 5.1: Alan Robinson (1998).

As opposed to earlier approaches, resolution is an explicitly machine-oriented (rather than human-oriented) form of inference so that single inferences (resolution steps) are often beyond the ability of human comprehension and are therefore opaque to human beings. According to Robinson, this did not matter because resolution was designed to be implemented on a computer in the first place.

Since Robinson's seminal paper numerous improvements and refinements to resolution haven been proposed. And the rest, as they say, is history.[1]

[1]Born in Yorkshire in 1930, Robinson came to the United States in 1952 with a classics degree from Cambridge University. He studied philosophy at the University of Oregon before moving to Princeton where he received his PhD in philosophy in 1956. Temporarily "disillusioned with philosophy," he went to work as an operations research analyst for Du Pont, where he learnt programming and taught himself mathematics. Robinson moved to Rice University in 1961, spending his summers as a visiting researcher at the Argonne National Laboratory's Applied Mathematics Division. Its then Director, William F. Miller, pointed Robinson in the direction of theorem proving. Miller "was thinking a great deal about 'what could be automated' including data analysis, control of experiments, aids to theory development, design automation, and aids to programming. I was much influenced by

5.1 Binary resolution

Binary resolution is the most elementary form of resolution that is complete. With binary resolution, precisely two clauses from the clause set are used to infer a new clause. Hence, the name.

Binary resolution rests on the equivalence

$$(a \vee p) \wedge (b \vee \neg p) \equiv (a \vee p) \wedge (b \vee \neg p) \wedge (a \vee b) \tag{5.1}$$

On the basis of this equivalence, we may extend clause sets with other (hopefully simpler) clauses while preserving equivalence. For example, with (5.1) the clause set

$$\{C_1, \ldots, C_m, \{a, p\}, \{b, \neg p\}\}$$

may be extended with the clause $\{a, b\}$, resulting in the logically equivalent clause set

$$\{C_1, \ldots, C_m, \{a, p\}, \{b, \neg p\}, \{a, b\}\}.$$

The new clause is obtained by joining the two parent clauses such that a complementary pair of literals—in this case p, $\neg p$—is deleted.

The clause $\{a, b\}$ is called a *resolvent* of the clauses $\{a, p\}$ and $\{b, \neg p\}$ and write

$$\{a, p\}, \{b, \neg p\} \rightsquigarrow \{a, b\}.$$

We speak of *a* resolvent, because two clauses can have multiple resolvents. For complementary literals that disappear in the process of forming a resolvent, we sometimes say that they *clash*.

Resolvents are not produced for their own sake. If produced and added repeatedly, we may or may not arrive at the empty clause $\square$. If we do, we have shown that the original clause set was inconsistent. Using logic programming terminology, we would say that we produced a *refutation* of the original clause set. Accordingly, a resolution (a sequence of elementary resolution steps that ends in with the empty clause is called a *resolution refutation*.

Example 5.1 (Binary resolution) Suppose we would like to prove $\phi = [(p \supset r) \wedge (q \supset r) \wedge (p \vee q)] \supset r$, with refutation by resolution. To this end, we form the negation of ϕ, and convert it to a clause set:

$$\begin{aligned}
\neg\phi &\equiv \neg[[(p \supset r) \wedge (q \supset r) \wedge (p \vee q)] \supset r] \\
&\equiv [(p \supset r) \wedge (q \supset r) \wedge (p \vee q)] \wedge \neg r \\
&\equiv (\neg p \vee r) \wedge (\neg q \vee r) \wedge (p \vee q) \wedge \neg r \\
&\equiv \{\{\neg p, r\}, \{\neg q, r\}, \{p, q\}, \{\neg r\}\}. \qquad (5.2)
\end{aligned}$$

We will now extend (5.2) repeatedly by adding resolvents:

$$\begin{aligned}
&\equiv \{\{\neg p, r\}, \{\neg q, r\}, \{p, q\}, \{\neg r\}\} && \{\neg q, r\}, \{p, q\} \rightsquigarrow \{p, r\} \\
&\equiv \{\{\neg p, r\}, \{\neg q, r\}, \{p, q\}, \{\neg r\}, \{p, r\}\} && \{\neg p, r\}, \{p, r\} \rightsquigarrow \\
&\equiv \{\{\neg p, r\}, \{\neg q, r\}, \{p, q\}, \{\neg r\}, \{p, r\}, \{r\}\}\{r\} && \{\neg r\}, \{r\} \rightsquigarrow \square \\
&\equiv \{\{\neg p, r\}, \{\neg q, r\}, \{p, q\}, \{\neg r\}, \{p, r\}, \{r\}, \square\}
\end{aligned}$$

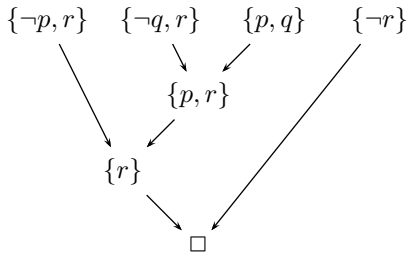
Allen Newell and had many discussions with him." Miller showed Robinson a 1960 paper by Martin Davis and Hilary Putnam (coincidentally, the latter had been Robinson's PhD supervisor) proposing a predicate-calculus proof procedure that seemed potentially superior to Gilmore's, but which they had not yet turned into a practical computer program. Miller suggested that Robinson use his programming skills to implement Davis and Putnam's procedure on the Argonne IBM 704. Robinson quickly found that their procedure remained very inefficient. However, while implementing a different procedure also suggested in 1960 by Dag Prawitz, Robinson came to see how the two sets of ideas could be combined into a new, far more efficient, automated proof procedure for first-order predicate logic, now known as resolution. (Taken and slightly adapted from [62].)

We end up with an empty clause $\square$, which means that $\neg\phi$ is false, so that ϕ is true. Hence the proof of ϕ is completed.

The entire resolution process can be depicted linearly:

1.	$\{\neg p, r\}$	premise
2.	$\{\neg q, r\}$	premise
3.	$\{p, q\}$	premise
4.	$\{\neg r\}$	premise
5.	$\{p, r\}$	(resolvent of 2 and 3)
6.	$\{r\}$	(resolvent of 1 and 5)
7.	$\square$	(resolvent of 4 and 6),

or as a directed acyclic graph (DAG):



The advantage of the DAG representation is that all clauses appear at most once. Unfortunately, the present example uses all clauses at most once, so that the example is not very illustrative.

We will now give a completeness proof of binary resolution. This proof is given because completeness proofs of many forms of resolution run along the same lines.

Theorem 5.2 *Binary resolution is sound and complete for propositional logic.*

Proof. Let $S = \{C_1, \ldots, C_n\}$ be a non-empty clause set, and let R be the set that is formed by starting with S and adding resolvents.
- Soundness: If $\square \in R$, then R is unsatisfiable. Since adding resolvents does not change satisfiability (Eq. 5.1 on the facing page), this means that S must have been unsatisfiable.
- Completeness: The approach here is to prove the implication "S is unsatisfiable $\Rightarrow \square \in R$". We do this with induction on the number of different proposition letters in S. Let this number be n.

i. If $n = 0$, then S is either $\varnothing$ or $\{\square\}$. Since the empty set is satisfiable while S is not, S must be $\{\square\}$, so that $\square$ is a resolvent of S (in zero steps).

ii. For the induction step, we assume that S uses n proposition letters. Select a letter that occurs in S, say p. Split S with p and $\neg p$ to obtain two clause sets $S_1 = S|_{p=1}$ and $S_2 = S|_{p=0}$, respectively. S_1 is not satisfiable, because if it were, then a model m for S_1 would be a model for S if we make $m(P) = 1$. Similarly, S_2 is unsatisfiable. Further, both S_1 and S_2 use less than n different proposition letters. Hence, we may use the induction hypothesis to conclude that there exist resolution refutations from S_1 and from S_2 that produce $\square$. Add to every clause in the refutation of S_1 the literal p, and add to every clause in the refutation of S_2 the literal $\neg p$. It can be verified that the "diluted" refutations consist of admissible resolvent-steps with clauses that stem from S. If one of these two diluted refutations end in the empty clause, we are done. Else, both "pseudo refutations" must end in clause sets that contain the unit-clauses $\{p\}$ and $\{\neg p\}$, respectively. These two pseudo refutations may be combined to produce a refutation of S proper.

□

The completeness of binary resolution is an important result: we now know of at least one form of resolution that is complete. A less attractive aspect of binary resolution, however, is that in its pure and unrestricted form, it is a non-deterministic process with many degrees of freedom. It describes what *may* be done,—not what *must* be done. Thus, binary resolution in its pure and unrestricted form is an ineffient theorem proving algorithm that produce many irrelevant resolvents in many different ways.

To make resolution somewhat more efficient, there are a number of things we could do. One such thing is to restrict ourselves to special forms of binary resolution. In this manner, we reduce the number of choices that the theorem prover can make. By restricting ourselves to special forms of binary resolution, however, we may forfeit completeness. Restricted but *complete* versions of binary resolution include p-resolution, n-resolution, linear resolution, the set-of-support strategy (SOS), and SOS combined with linear resolution. These forms of resolution are explained below. Incomplete (but very efficient) forms of binary resolution are unit resolution and input resolution. Also these forms of resolution are explained below.

PROBLEMS (Sec. 5.1)

1. Which of the following expressions are clause sets: $\{\{p\}\}$, $\{\Box\}$, $\{p, \{\neg q\}\}$, $\{\Box, \{\neg q\}\}$, $\varnothing$, $\{\varnothing\}$, $\{\Box, \{p\}\}$, $\{\Box, \Box\}$? (A definition of clause sets can be found on page 84.)

2. Verify that Eq. 5.1 on page 98 is a tautology. (You may use the truth-table method, a semantic tableau, or similar means.)

3. Suppose $C_1, C_2 \rightsquigarrow C_3$. I.e., clause C_3 is a resolvent of C_1 and C_2.

   (a) Argue why $\{C_1, C_2\} \equiv \{C_1, C_2, C_3\}$. (Or prove it, if necessary.)

   (b) Show with an example that two clauses may have more than one resolvent.

   (c) Show with an example that a resolvent may not replace its parents without possibly violating the logical equivalence.

4. (a) Use (binary) resolution to prove that the following propositions are unsatisfiable. Convert to clause sets first.

      i. $(w \vee \neg u \vee \neg v) \wedge \neg w \wedge u \wedge (\neg u \vee v)$

      ii. $(r \vee \neg u) \wedge (\neg r \vee s) \wedge (u \vee s) \wedge (\neg s \vee v) \wedge (\neg v \vee \neg s)$

   (b) Use (binary) resolution refutation to prove the following propositions. Convert to clause sets first.

      i. $\neg[(w \vee \neg u \vee \neg v) \wedge \neg w \wedge u \wedge (\neg u \vee v)]$

      ii. $\neg[(r \vee \neg u) \wedge (\neg r \vee s) \wedge (u \vee s) \wedge (\neg s \vee v) \wedge (\neg v \vee \neg s)]$

5. Let S be in $\leq$2CNF-form, which means that every clause consists of at most two literals.

   (a) Let R be the set that is formed by starting with S and adding resolvents. How large can R be at its most? (Solution.)

   (b) Explain that there exists a polynomial time algorithm to test the satisfiability of $\leq$2CNF-clauses. (Solution.)

5.2 Linear resolution

Linear resolution is important because it lies at the basis of many theorem provers. A resolution is linear if every resolvent is used immediately to produce another resolvent. More precisely:

Definition 5.3 (Linear resolution) Given a set S of clauses, a *linear resolution* of $C_0, \ldots, C_n$ from S is a resolution where $C_0 \in S$ and, for each $1 \leq i \leq n$, C_i is a resolvent of C_{i-1} and B where B is some previous clause (that may be in S but must differ from C_i).

Thus, with linear resolution we have an immediate parent, also called the *center clause*, and a second parent called the *side clause*. The clause $C_1 \in S$ is called the *top clause*.

What is interesting to linear resolution is its simple structure. Further, linear resolution is sound and complete with respect to the semantics of propositional logic. A proof of this fact is comparable to the proof of Theorem 5.2 on page 99. Also here we could use the number of different proposition letters in a clause set as an induction measure. This number can be reduced by applying the splitting rule (p. 91) on an arbitrary literal l. If we are able to prove that the reduced clause set(s) is (are) also unsatisfiable, then we can apply the induction hypothesis to conclude that the reduced clause set(s) can be refuted by means of linear resolution(s). The desired resolution of the original clause set, is then obtained by "lifting" the linear resolution(s) of the reduced clause sets to fragments in which l and $\neg l$ again occur. Finally, it is possible to put the "lifted" fragments together and to complete the result into a linear resolution of the original clause set. A complete proof can be found in, e.g., [13] and [97].

Another attractive property of linear resolution is that it remains complete if it is combined with the set-of-support method (Section 5.3 on page 105).

Unit resolution and input resolution

A disadvantage of linear resolution (and of many other types of resolution, by the way) is that the resolvents are often larger than their parent clauses. Unit resolution does not suffer from this shortcoming.

Definition 5.4 (Unit resolution) With *unit resolution* at least one of the parent clauses must be a unit clause, i.e., a clause consisting of one (positive or negative) literal.

From two clauses of length 1 and n, respectively, unit resolution produces a resolvent of length $n-1$ (propositional logic) or length at most $n-1$ (first-order logic). Another advantage of unit resolution, is that it is easy to implement on the computer.

The one-literal rule (OLR) and unit resolution are very much alike. The difference between the two is that unit resolution adds one clause and preserves equivalence, while the OLR removes one or more clauses and does not preserve equivalence. The OLR, however, is satisfiability-invariant, which is sufficient for resolution.

Example 5.5 If we apply the OLR on

$$\{\{p, \neg q\}, \{\neg p, q, r\}, \{p, u\}, \{\neg p, r\}, \{p\}\} \tag{5.3}$$

and p, we get

$$\{\underline{\{p, \neg q\}}, \{\underline{\neg p}, q, r\}, \underline{\{p, u\}}, \{\underline{\neg p}, r\}, \underline{\{p\}}\}$$
$$\{\{q, r\}, \{r\}\}$$

If we apply unit resolution with $\{p\}$ on (5.3), we get

$$\begin{aligned} &\{\{p, \neg q\}, \{\neg p, q, r\}, \{p, u\}, \{\neg p, r\}, \{p\}\} \\ \equiv &\{\{p, \neg q\}, \{\neg p, q, r\}, \{p, u\}, \{\neg p, r\}, \{p\}, \{r\}\} && \text{by } \{\neg p, r\}, \{p\} \rightsquigarrow \{r\} \end{aligned}$$

and then

$$\equiv \{\{p, \neg q\}, \{\neg p, q, r\}, \{p, u\}, \{\neg p, r\}, \{p\}, \{r\}, \{q, r\}\} \qquad \text{by } \{\neg p, q, r\}, \{p\} \rightsquigarrow \{q, r\}$$

and so forth.

Another restricted form of resolution is input resolution.

Definition 5.6 (Input resolution) Let S be an input set. An *input resolution* is a resolution where each resolvent has a parent in S.

It can be proven that input resolution is equivalent to unit resolution. This means that input resolution is incomplete as well, except if all clauses are Horn clauses.

PROBLEMS (Sec. 5.2)

1. P-resolution is binary resolution where one of the parents is a positive clause (i.e., a clause with positive literals only). Similarly for N-resolution. Show that a combining the restrictions of P-resolution and N-resolution is incomplete. (Solution.)

2. (Linear resolution.) It is a well-known fact that input resolution is linear. But what about

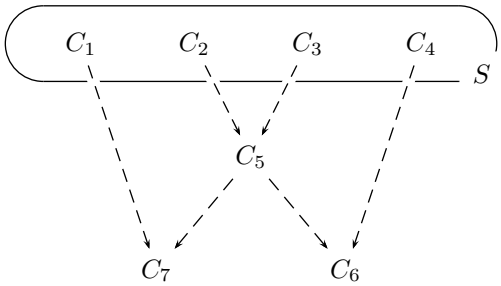


Figure 5.2: Input resolution with input C_1, C_2, C_3, C_4.

the resolution depicted in Figure 5.2, where C_1, C_2, C_3, C_4 are input and the order of derivation is C_5, C_6, C_7?

Hint: it may help to make a distinction between a resolution step, a resolution, and a resolution refutation. (Consult the index for definitions if necessary.) (Solution.)

5.3 Semantic resolution

Semantic resolution is a powerful specialization of binary resolution, and is used in many theorem provers, such as OTTER. With semantic resolution, we pick a model m (hence the name) and divide the clause set S into two groups: one group S_1 that is made true by m, i.e., $S_1 =_{Def} \{ C \mid m \models C\}$ and the rest $S_2 =_{Def} S \backslash S_1$. We then require that parent clauses must come from different groups. This obviously cuts down the number of possible resolutions, and therefore semantic resolution is more deterministic than plain binary resolution. It can be proven that semantic resolution is complete [13, 97].

Semantic resolution is often combined with an ordering on proposition letters or predicate symbols. For example, if the set of proposition letters available is $\{ a, \ldots, z\}$, we may put $a < \cdots < z$. The constraint with ordered resolution, then, is the following.

Definition 5.7 (Ordered resolution) Only the $<$-smallest element of the m-falsified clause of every resolution step $C_1, C_2 \rightsquigarrow R$ may be used as a literal to resolve upon.

In such cases, R is called $<$-*admissible*. For example, if $C_1 = \{a, \neg g, k, m, \neg p\}$ and $C_2 = \{\neg b, c, \neg k, \neg n\}$, then $R_1 = \{a, \neg b, c, \neg g, m\neg n, \neg p\}$ would normally be a resolvent of C_1 and C_2. However, R_1 is not a $<$-admissible resolvent of C_1 and C_2, because a, rather than k, is the $<$-smallest element of C_1. If $C_3 = \{k, m, \neg p\}$, then C_3 can be resolved with C_2, because k is the $<$-smallest element of C_3. Thus, $C_3, C_2 \rightsquigarrow R_2$ with $R_2 = \{\neg b, c, \neg k, m, \neg n, \neg p\}$ a $<$-admissible resolvent of C_3 and C_2.

We may also sort the literals in a clause, and then say that only the first element of the first clause may be used for resolution. It can be proven that ordered semantic resolution is still complete.

Semantic clash

Ordered semantic resolution may further be constrained without loosing completeness. To see how, consider

$$S = \{\{\neg p, \neg q, r\}, \{p, r\}, \{q, r\}, \{\neg r\}\}.$$

Suppose we apply ordered semantic resolution to S with $p < q < r$ and with m such that m falsifies all positive literals. Thus, m divides the clause space into positive clauses and non-positive clauses, i.e., clauses with at least one negative literal.

There are two m-falsified (i.e., positive) clauses, viz. $\{p, r\}$ and $\{q, r\}$. The $<$-least elements of both clauses are p and q, respectively. With ordered semantic resolution both clauses can only resolve with the first clause to produce $\{\neg q, r\}$ and $\{\neg p, r\}$, respectively.

No new positive clauses are added, so the new clauses can only resolve with the two positive clauses already present to produce $\{r\}$ and $\{r\}$, respectively. From $\{r\}$ (positive) and $\{\neg r\}$ (non-positive) we may then produce the empty clause.

Because the two non-positive clauses may never resolve with each other, the step to $\{r\}$ could have been taken at once:

$$\{\neg p, \overset{\{p,r\}}{\neg q}, \overset{\{q,r\}}{r}\} \quad \Rightarrow \quad \{r\}$$

This is not an ordinary resolution step, so it has to have some other name, viz. *semantic clash*, or clash for short.

Definition 5.8 (Semantic clash) Let m be a model, and let $<$ be an ordering of proposition letters. A clause set

$$\{N, C_1, \ldots, C_n\}, n \geq 1$$

is called a *semantic clash* relative to m and $<$ if

1. $R_1 = N$ and for each $1 \leq i \leq n$, there is a $<$-admissible resolvent $C_i, R_i \rightsquigarrow R_{i+1}$
2. $C_1, \ldots, C_n$ and R_{n+1} are falsified by m

N is called the *nucleus*, $C_1, \ldots, C_n$are called the *satellites*, and R_{n+1} is called a *resolvent*.

An important special case of a semantic clash is hyperresolution.

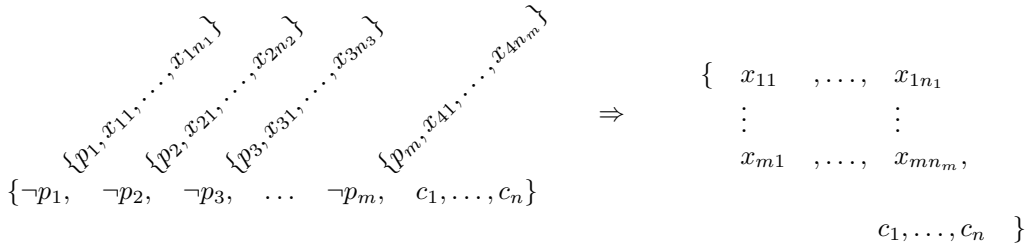


Figure 5.3: Nucleus [horizontal] and satellites [diagonal]

Hyperresolution

Hyperresolution is a special case of semantic resolution in the sense that, with hyperresolution, m is such that it falsifies all literals. As a result, the clause space is divided in positive and non-positive clauses so that in a semantic clash, the nucleus is non-positive, while the satellites and the resolvent are positive. In a clash, the satellites remove the negative literals from the nucleus to produce a positive resolvent which may then be used as a satellite in future resolution steps. Hyperresolution is the main rule of inference of the resolution based automated theorem provers such as OTTER.

A successful application of hyperresolution can be viewed as a sequence of binary resolutions in which each is required to involve exactly one positive clause. However, all such binary resolutions must occur simultaneously, thus yielding no intermediate clauses. For example, hyperresolution applied to the clauses in the LHS of Fig. 5.3 yields the clause in the RHS of Fig. 5.3.

```
============= input processing ========        # First, Otter echo's all its input.
set(hyper_res).                                 # Hyperresolution is switched on.
assign(stats_level,0).                          # Statistics report is switched off.

list(usable).                list(sos).         # Clauses are divided into two sets:
1 [] -a| -b|c.               4 [] a.            # usable, and set-of-support.
2 [] -d| -e|f.               5 [] b.
3 [] -c| -f|g.               6 [] d.       # * "usable" contains "background knowledge"
end_of_list.                 7 [] e.       #   three implications, namely a,b->c and
                             8 [] -g.      #   d,e->f and c,f->g
                             end_of_list.  # * "sos" contains four premises a,b,d,e and
                                           #   the negation of the conclusion we would
======= end of input processing =======    #   like to see proven
```

Figure 5.4: Hyperresolution in OTTER. Part 1: echo of input

Another way to understand hyperresolution, is to represent clauses as implications. If $c =_{Def} c_1 \wedge \ldots \wedge c_n$, and the hyperresolution depicted in Fig. 5.3 is written as

$$\begin{array}{rcl} \neg x_{11} \wedge \ldots \wedge \neg x_{1n_1} \rightarrow p_1 & & \\ \qquad\qquad \vdots \quad \vdots & & \\ \neg x_{11} \wedge \ldots \wedge \neg x_{1n_m} \rightarrow p_m & \text{and} & p_1 \wedge \ldots \wedge p_m \rightarrow c \end{array} \quad \text{yields} \quad \begin{array}{l} \neg x_{11} \wedge \quad \ldots \quad \wedge \neg x_{1n_1} \\ \vdots \qquad\qquad \vdots \\ \neg x_{m1} \wedge \quad \ldots \quad \wedge \neg x_{mn_m} \rightarrow c \end{array}$$

we see that the implications $\{\neg \bar{x} \rightarrow p_j\}_{j=1}^{m}$ together with the implication $p_1, \ldots, p_m \rightarrow c$, make a new implication $\neg \bar{x}_1 \wedge \ldots \wedge \neg \bar{x}_m \rightarrow c$. Thus, with a little benevolence, hyperresolution can be seen as a disguised form of rule-based reasoning.

```
========== start of search ==========   # the first clause of the sos (clause 4)
given clause #1: (wt=1) 4 [] a.         # is taken from the queue and becomes given
                                        # clause #1.  doesn't enable any resolution
given clause #2: (wt=1) 5 [] b.         # step, so #2 is taken from sos.  Then a HR
** KEPT (pick-wt=1): 9 [hyper,5,1,4] c. # with nucleus clause 1 and satellites 4, 5
9 back subsumes 1.                      # produces clause 9, viz. c.  Clause 9 makes
                                        # clause1 dependent, so clause 1 is deleted.
given clause #3: (wt=1) 6 [] d.

                                        # then clause 6 and 7 are pulled from the sos
given clause #4: (wt=1) 7 [] e.         # to act as satellites in a HR that yields 10
** KEPT (pick-wt=1): 10 [hyper,7,2,6] f. # 10 is stronger than 2, so 2 is deleted
10 back subsumes 2.

given clause #5: (wt=1) 8 [] -g.            # clause 8 is pulled from the sos

given clause #6: (wt=1) 9 [hyper,5,1,4] c.      # no HR possible, so 9 is pulled from
                                                # the sos immediately
given clause #7: (wt=1) 10 [hyper,7,2,6] f.  # still no HR possible, so 10 is pulled
** KEPT (pick-wt=1): 11 [hyper,10,3,9] g.    # from sos.  HR is possible with 10,3,9

--> UNIT CONFLICT at   0.00 sec --> 12 [binary,11.1,8.1] $F. # unit conflict is no HR
                                                             # used exclusively by
Length of proof is 3.  Level of proof is 2.          # otter to produce empty clauses
```

Figure 5.5: Hyperresolution in OTTER. Part 2a: search

Set-of-support resolution

Another practical specialization of semantic resolution is set-of-support resolution.
Set-of-support resolution is used in OTTER and other theorem provers.

Suppose S is a set of logical formulas of a more or less permanent character (such as a database). Further suppose that (we are absolutely sure that) S is consistent and that we have pre-processed S into a clause set: $S = \{C_1, \ldots, C_n\}$. Suppose that we want to know whether $S \vdash \phi$. In terms of refutation, this means that we want to know whether $S \cup \{\neg\phi\}$ is consistent. We can investigate this with resolution. To this end $\neg\phi$ must converted to a clause set. Let us say it converts to $\{D_1, \ldots, D_m\}$. Thus, the original question comes down to asking whether $\{C_1, \ldots, C_n\} \cup \{D_1, \ldots, D_m\}$ can be refuted. Since $\{C_1, \ldots, C_n\}$ is consistent, this means that every resolvent must have an ancestor in $\{D_1, \ldots, D_m\}$ in order to take part in a refutation. The latter clause set, stemming from $\neg\phi$, is called the set-of-support (SOS).

Using the fact that linear resolution is complete, it follows that set-of-support resolution is complete as well: if $\{C_1, \ldots, C_n\} \cup \{D_1, \ldots, D_m\}$ is unsatisfiable, then a minimally unsatisfiable subset must contain at least one clause D_j, because $S = \{C_1, \ldots, C_n\}$ is satisfiable. From the completeness of linear resolution it now follows that there is a linear resolution refutation of $\{C_1, \ldots, C_n\} \cup \{D_j\}$ with top clause D_j. This is also a set-of-support resolution with SOS $\{D_j\}$, and hence a set-of-support resolution with SOS $\neg\phi \equiv \{D_1, \ldots, D_m\}$.

Combinations

Restricted forms of binary resolution permit less freedom than plain binary resolution, so that fewer clauses can be selected in the refutation process. This decrease in non-determinism makes the algorithm more efficient. However, practical experiments have shown that the gain in speed is negligible, so that we must search for other approaches.

One approach is to combine two complete restriction methods to make the algorithm even *more*

```
---------------- PROOF ----------------
                                                    # Finally, the relevant
1 [] -a| -b|c.                                      # parts of the search
2 [] -d| -e|f.                                      # are combined into a
3 [] -c| -f|g.                                      # proof that is printed
4 [] a.                                             # for inspection
5 [] b.
6 [] d.
7 [] e.
8 [] -g.
9 [hyper,5,1,4] c.
10 [hyper,7,2,6] f.
11 [hyper,10,3,9] g.
12 [binary,11.1,8.1] $F.

------------ end of proof -------------
============ end of search ============
```

Figure 5.6: Hyperresolution in OTTER. Part 2b: proof

deterministic. In doing this, we must take care not to forfeit completeness. An example for which completeness is lost is the combination of P- and N- resolution. P- and N- resolution are complete resolution methods, but when we combine both restrictions, the resulting resolution method becomes incomplete (Exercise 1 on page 102).

A combination of two complete restriction methods that *is* complete, is linear resolution combined with set of support. This strategy is often followed in practice. OTTER, the theorem prover of the Argonne ATP-school uses this strategy, under the header "given-clause strategy". The given clause, then, is simply equal to the last resolvent.

PROBLEMS (Sec. 5.3)

1. Show with hyperresolution that the following clause sets are unsatisfiable. Remember that with hyperresolution the satellites are positive, so that all resolvents are positive as well.

   (a) $\{\{\neg d, e\}, \{a, \neg d, \neg e, f\}, \{d\}, \{\neg a, \neg d, g\}, \{\neg e, \neg f, g\}, \{\neg g\}\}$. (Solution.)

   (b) $\{\{d, a, \neg f, g\}, \{\neg a, \neg f\}, \{\neg d\}, \{f, b\}, \{\neg g, a, b\}, \{\neg b, d\}\}$. (Solution.)

   (c) $\{\{-g, d\}, \{g, b, -c\}, \{-d, b, -c\}, \{-b, g\}, \{-d, -b\}, \{g, c\}, \{c, -d\}\}$. (Solution.)

2. Show with negative hyperresolution that (1c) is unsatisfiable. Remember that with negative hyperresolution the satellites are negative, so that all resolvents are negative as well. (Solution.)

5.4 First-order resolution

Most of what holds for propositional resolution goes through for the first-order case, but some concepts are deepened due to the fact that the language of first-order logic is more expressive. One such concept, for example, is the conversion from non-clausal formulas to a clause set. This is called *normalization*.

Normalization

Resolution-based theorem provers operate on clauses and on clauses only. If the user inputs non-clausal formulas, then a resolution-based theorem prover immediately generates clauses from them.

We now describe how an arbitrary first-order formula can be rewritten into a clause set.

1. If the task is to prove that ψ follows from $\phi_1, \ldots, \phi_n$, then deny $\phi_1, \ldots, \phi_n \vdash \psi$ for the purpose of resolution refutation in the form of one big formula: $\phi_1 \wedge \ldots \wedge \phi_n \wedge \neg\psi$. (The objective now is to prove that the latter formula is unsatisfiable.) All subsequent operations can be performed in this one big formula, or on every conjunct in isolation. (Depends on the algorithm.)

2. Convert all implications, bi-implications, exclusive ors, nands and nors to equivalent sub-formulas in which only the connectives $\neg$, $\wedge$ and $\vee$ occur. Use rewrite rules such as $\phi \supset \psi \rightsquigarrow (\neg\phi) \vee \psi$ and $\phi \equiv \psi \rightsquigarrow (\phi \supset \psi) \wedge (\psi \supset \phi)$.

3. Ensure that all quantifiers use variables that are different and do not occur free elsewhere in the formula. This can simply be done by giving each quantifier its own variable (if it does not already have its own variable). Now the formula is said to be *rectified*.

4. Make that all quantifiers come first. This can be done with the following *quantifier rewrite rules*:

$$\begin{array}{ll} \neg(\forall x)(\phi\,) \rightsquigarrow (\exists x)(\neg\phi\,) & \neg(\exists x)(\phi\,) \rightsquigarrow (\forall x)(\neg\phi\,) \\ (\forall x)(\phi\,) \wedge \psi \rightsquigarrow (\forall x)(\phi \wedge \psi) & (\exists x)(\phi\,) \wedge \psi \rightsquigarrow (\exists x)(\phi \wedge \psi) \\ \phi \wedge (\forall x)(\psi) \rightsquigarrow (\forall x)(\phi \wedge \psi) & \phi \wedge (\exists x)(\psi) \rightsquigarrow (\exists x)(\phi \wedge \psi) \\ (\forall x)(\phi\,) \vee \psi \rightsquigarrow (\forall x)(\phi \vee \psi) & (\exists x)(\phi\,) \vee \psi \rightsquigarrow (\exists x)(\phi \vee \psi) \\ \phi \vee (\forall x)(\psi) \rightsquigarrow (\forall x)(\phi \vee \psi) & \phi \vee (\exists x)(\psi) \rightsquigarrow (\exists x)(\phi \vee \psi) \end{array}$$

   These rules only apply if no free variables of a formula become bound by global quantifiers. Therefore, this step must be preceded by steps 2 and 3. Now the formula is said to be in *prenex normal form*

5. Skolemize variables that are bound by existential quantifiers (cf. Result 5.9, below). In this way, the existential quantifiers are deleted. (This step must be preceded by Step 4.)

6. Drop all quantifiers. (By Step 5, these are universal quantifiers.)

7. Rewrite the formulas in CNF, or (optional) in $\leq$3CNF (page 85).

8. Write the CNF thus obtained as a clause set. (Easy, of course.)

9. Introduce fresh variables where necessary so that no variable occurs in more than one clause. Now the clauses are said to be *standardized apart*.

Step 5, 6, and 7 do not respect logical equivalence, but do maintain satisfiability-equivalence, which is sufficient if we want to check clause sets on satisfiability.

Here is a typical example of a normalized first-order formula:

$$\begin{array}{lll} \phi = & \{\{\neg pa, \varsigma_1(v_1) \vee \neg r v_1\}, & \{p v_3, g(b, v_3) \vee \neg q f(b), \varsigma_1(v_4), v_5 \vee \neg r a\}, \\ & \{q a, b, c \vee \neg r v_6\}, & \{p f(g(c, \varsigma_0), v_7), f(c) \vee \neg q c, \varsigma_3(v_8, v_9, f(v_{10})), c\}, \\ & \{p v_{11}, g(b, v_{11}) \vee q c, b, a\}, & \{\neg p g(b\varsigma_1(v_{12}), f(a) \vee \neg r f(v_{12})\}, \\ & \{\neg p \varsigma_4(v_{13}, v_{14})\}, & \{\neg p g(\varsigma_0, a), v_{15} \vee q v_{15}, v_{16}, v_{15} \vee \neg r \varsigma_1(v_{15})\}, \\ & \{pa, b \vee \neg q a, b, v_{17} \vee r v_{18}\}, & \{q v_{19}, b, c \vee r c\}\} \end{array}$$

(The commas separates between predicate arguments rather than predicates themselves.) This formula is perhaps unreadable to us, but resolution-based theorem provers like them this way.

PROBLEMS (Sec. 5.4)

1. Convert the following problems into clause sets. Follow the above steps up to and including standardization of clauses.
   (a) $\vdash_? (\forall x)(px \wedge \neg px)$ (Solution.)
   (b) $\vdash_? (\exists x)(px \vee \neg px)$ (Solution.)
2. Explain why standardizing clauses apart preserves logical equivalence, despite the fact that clauses in a clause set are logically related to each other.
3. Transform $(\forall x, y)(px, y) \supset \neg((\forall y)(qx, y \supset Rx, y))$ into a clause set.
4. Transform
   $$(\forall x)(rx, y \supset (\exists z)(\neg(\exists x)(rx, z))) \wedge (\forall z)((\forall n)(tn, x, z \supset ((\forall z \exists u \exists u)px, u, u)))$$
   into a clause set. (Solution.)

Skolemization

Step 5 made mention of Skolemization. The idea of Skolemization is that the dependence of existential variables on universal variables can be made explicit by a function that has the universal variables as arguments.

Result 5.9 (Skolemization) *Let ϕ be a first-order formula, and let ς be a function symbol that does not occur in ϕ. Then $(\forall x_1, \ldots, x_n)(\exists y)(\phi)$ is satisfiable if and only if*

$$(\forall x_1, \ldots, x_n)(\phi[\varsigma(x_1, \ldots, x_n)/y])$$

is satisfiable.

A proof of this result is left as Exercise 3 on page 120. (Solution included.)

It is not at all obvious how the above description of normalization can be converted to an efficient algorithm. In this light, it is perhaps interesting to know that `formula.c`—OTTER's component that converts quantified formulas to clause sets and back—consists of roughly 3100 lines of C code.

Factors

Binary resolution is still complete for first-order logic, but not in the form as we know it from the propositional case. To see the problem, suppose for a moment that CNFs are represented by clause multi-sets, rather than by clause sets.

A multi-set (or *bag*), is an unordered list, in which some elements may occur more than once. Now suppose that the CNF $(p \vee p) \wedge (\neg p \vee \neg p)$ is represented by the clause multi-set $M = \{\{p, p\}, \{\neg p, \neg p\}\}$. The problem here is that the empty clause cannot be derived with binary resolution, although M is unsatisfiable. (Verify this!) Thus, we conclude that binary resolution in propositional logic is complete only if literals in clause multi-sets are unique. The latter is essential to the completeness of propositional resolution.

To see the impact of this observation for first-order logic, consider

$$S = \{\{pu, pv\}, \{\neg px, \neg py\}\}. \tag{5.4}$$

This clause set cannot be satisfied, because $S \equiv (\forall u, v, x, y)((pu \vee pv) \wedge (\neg px \vee \neg py))$ and this formula is unsatisfiable. Because S is unsatisfiable, it is reasonable to expect that binary

resolution leads us to the empty clause eventually. However, also in this case binary resolution does not seem to work, for reasons that are similar to the propositional case, except that (5.4) is not a multiset. For example, if we choose choose $\sigma = \{x/u\}$ as a (most general) unifier of pu and $\neg px$, then $\{px, pv\}$ can be resolved with $\{\neg px, \neg py\}$ into $\{pv, \neg py\}$. All other possible resolvents are variants of this clause. The problem is that if only two complementary literals are selected and resolved away, there may be a remaining third literal, and it is possible that this remaining third literal can only be resolved away in combination with one of the two first literals. We can solve this problem by incorporating all other literals that unify with the first two literals also in this resolution step.

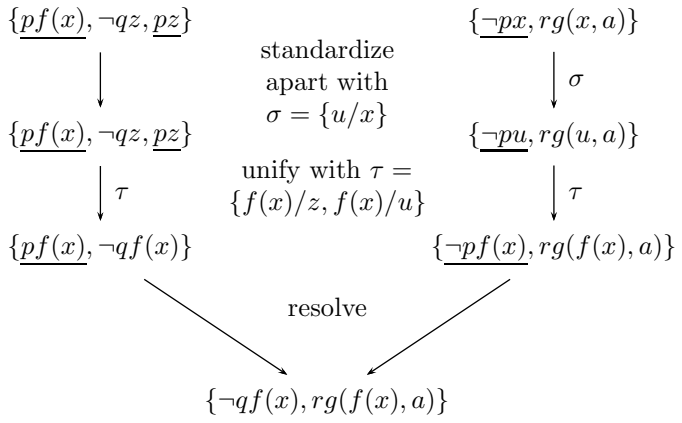


Figure 5.7: Resolution in predicate logic (example).

Example 5.10 The literals $pf(x)$, pz and $\neg px$ of the clauses $C_1 = \{pf(x), \neg qz, pz\}$ and $C_2 = \{\neg px, rg(x), a\}$ may be resolved by first standardizing C_1 and C_2 apart with $\sigma = \{u/x\}$:

$$C_1 = \{pf(x), \neg qz, pz\}, \qquad C_2\sigma = \{\neg pu, rg(u), a\}$$

Now C_1 and $C_2\sigma$ do not share variables anymore. Then C_1 and $C_2\sigma$ may be unified with $\tau = \{f(x)/u, f(x)/z\}$:

$$C_1\tau = \{pf(x), \neg qf(x), pf(x)\}, \qquad C_2\sigma\tau = \{\neg pf(x), rg(u), a\}$$

which simplifies to

$$C_1\tau = \{pf(x), \neg qf(x)\}, \qquad C_2\sigma\tau = \{\neg pf(x), rg(u), a\}.$$

A resolvent may now be produced as follows:

$$\{pf(x), \neg qf(x)\}, \{\neg pf(x), rg(u), a\} \rightsquigarrow \{\neg qf(x), rg(f(x)), a\}.$$

The literal $pf(x)$ is said to be a *factor* of $pf(x)$ and px. (End of example.)

More generally, if two or more literals (of the same sign) of a clause C have a most general unifier σ, then $C\sigma$ is called a *factor* of C. If $\sigma = \varnothing$, then C is also a factor of itself. A *first-order resolvent* of C_1 and C_2 is a binary resolvent of a factor of C_1 and a factor of C_2 (Fig. 5.8 on the next page.)

It is important to realize that factors are formed automatically during the inference process, more particularly when two groups of literals (in this case $\bar{L}_1, \ldots, \bar{L}_l$ and $L_1, \ldots, L_l$) are unified. Thus, the concept of a factor is not an additional feature that is required in order to make first-order resolution work: it is just a way of naming the phenomenon that certain substitutions make two sets of two or more literals become identical.

Definition 5.11 (First-order resolution) First-order resolution works analogous to propositional resolution, with the additional requirement that C_1 and C_2 may first have to be standardized apart into $C_1\sigma_1$ and $C_2\sigma_2$, and that a most general unifier may bring about that more than two literals may "clash" in a binary resolution. (Fig. 5.8.)

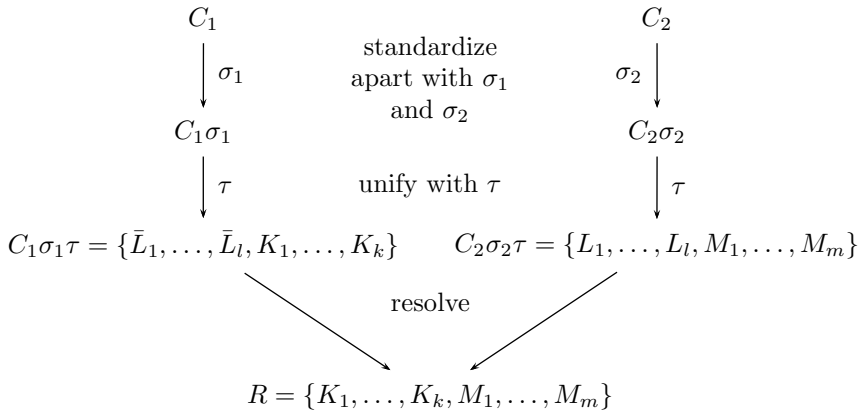


Figure 5.8: Resolution in predicate logic (general scheme).

It might be that the additional freedom that we now have in combining first-order clauses may result in the possibility that we may unjustly infer an empty clause from an otherwise satisfiable clause set. The following lemma indicates that this never happens.

Lemma 5.12 (Resolution lemma) *First-order resolution is a sound inference procedure.*

Proof. This proof is a souped-up version of the proof of the equivalence in Equation 5.1 on page 98, so it is a good idea to try to prove that formula first.

We must prove that resolution can never produce the empty clause for satisfiable clause sets, so let S be a satisfiable clause set. To prevent troubles later on in the proof, suppose further that the clauses in S are standardized apart (i.e., have no variables in common). Because S is a clause set and clause sets are not really first-order formulas, we will have to normalize S into a formula of the form

$$(\forall \bar{x})(S)$$

where the prefix $(\forall \bar{x})$ is formed by all variables in S, and the matrix S is written in CNF with $\wedge$'s and $\vee$'s. Since S is satisfiable, we may now write

$$(M, s) \models (\forall \bar{x})(S), \tag{5.5}$$

for some model M and some valuation s. (Note that this is *not* the same as writing $(M, s) \models S$, because $(\forall \bar{x})(S)$ is closed while S may contain free variables.) Let

$$C_1 = \{\bar{L}_1, \ldots, \bar{L}_l, K_1, \ldots, K_k\} \text{ and}$$
$$C_2 = \{L_1, \ldots, L_l, M_1, \ldots, M_m\},$$

be two clauses in S, and let

$$R = \{K_1, \ldots, K_k\}\sigma_1\tau \cup \{M_1, \ldots, M_m\}\sigma_2\tau$$

be a resolvent of C_1 and C_2, where σ_1 and σ_2 are variable renamings, and τ is a substitution. If $\bar{y}$ are the variables in R, we are done if we are able to prove that $(M, s) \models (\forall \bar{y})R$, for then R is

satisfiable and hence non-empty. By (5.5) and the equivalence $(\forall x)(\phi \wedge \psi) \equiv ((\forall x)\phi) \wedge ((\forall x)\psi)$ for formulas that are standardized apart, we now have

$$(M, s) \models (\forall \bar{x})C_1 \text{ and } (M, s) \models (\forall \bar{x})C_2$$

Since $(M, s) \models (\forall \bar{x})\phi$ implies $(M, s) \models (\forall \bar{z})(\phi\sigma_1\tau)$ for every substitution τ and variable renaming σ and variables $\bar{z}$ in $\phi\sigma_1\tau$ (exercise), we have

$$(M, s) \models (\forall \bar{z})(C_1\sigma_1\tau) \text{ and}$$
$$(M, s) \models (\forall \bar{z})(C_2\sigma_2\tau).$$

where $\bar{z}$ are the variables in $C_1\sigma_1\tau$ and $C_2\sigma_2\tau$. Hence,

$$(M, s) \models (\forall \bar{z})\{\bar{L}_1, \ldots, \bar{L}_l, K_1, \ldots, K_k\}\sigma_1\tau \text{ and}$$
$$(M, s) \models (\forall \bar{z})\{L_1, \ldots, L_l, M_1, \ldots, M_m\}\sigma_2\tau.$$

Since $\bar{L}_1\sigma_1\tau = \ldots = \bar{L}_l\sigma_1\tau$ and $L_1\sigma_1\tau = \ldots = L_l\sigma_2\tau$,

$$(M, s) \models (\forall \bar{z})\{\bar{L}_1\sigma_1\tau, K_1\sigma_1\tau, \ldots, K_k\sigma_1\tau\} \text{ and}$$
$$(M, s) \models (\forall \bar{z})\{L_1\sigma_2\tau, M_2\sigma_2\tau, \ldots, M_m\sigma_2\tau\},$$

so that

$$(M, s) \models (\forall \bar{z})(\bar{L}_1\sigma_1\tau) \vee (\forall \bar{z})\{K_1\sigma_1\tau, \ldots, K_k\sigma_1\tau\} \text{ and}$$
$$(M, s) \models (\forall \bar{z})(L_1\sigma_2\tau) \vee (\forall \bar{z})\{M_1\sigma_2\tau, \ldots, M_m\sigma_2\tau\}.$$

Since $(\forall \bar{z})(\bar{L}_1\sigma_1\tau) \equiv (\forall \bar{z})(\neg L_1\sigma_1\tau) \equiv \neg(\forall \bar{z})(L_1\sigma_1\tau)$

$$(M, s) \models \neg(\forall \bar{z})(L_1\sigma_1\tau) \vee (\forall \bar{z})\{K_1\sigma_1\tau, \ldots, K_k\sigma_1\tau\} \text{ and}$$
$$(M, s) \models (\forall \bar{z})(L_1\sigma_2\tau) \vee (\forall \bar{z})\{M_1\sigma_2\tau, \ldots, M_m\sigma_2\tau\}.$$

Now either $(\forall \bar{z})(L_1\sigma_1\tau)$ is true or false in (M, s), so that either

$$(M, s) \models (\forall \bar{z})\{K_1\sigma_1\tau, \ldots, K_k\sigma_1\tau\} \text{ or}$$
$$(M, s) \models (\forall \bar{z})\{M_1\sigma_2\tau, \ldots, M_m\sigma_2\tau\}.$$

Hence,

$$(M, s) \models (\forall \bar{z})\{K_1\sigma_1\tau, \ldots, K_k\sigma_1\tau\} \vee (\forall \bar{z})\{M_1\sigma_2\tau, \ldots, M_m\sigma_2\tau\}.$$

which, by the equivalence $((\forall x)\phi) \vee ((\forall x)\psi) \equiv (\forall x)(\phi \vee \psi)$ for formulas that are standardized apart, amounts to

$$(M, s) \models (\forall \bar{z})\{K_1\sigma_1\tau, \ldots, K_k\sigma_1\tau, M_1\sigma_2\tau, \ldots, M_m\sigma_2\tau\}.$$

Finally, since $\bar{y} \subseteq \bar{z}$ and variables in $\bar{z} \backslash \bar{y}$ do not occur in R, also

$$(M, s) \models (\forall \bar{y})\{K_1\sigma_1\tau, \ldots, K_k\sigma_1\tau, M_1\sigma_2\tau, \ldots, M_m\sigma_2\tau\}.$$

which was to be proven. □

Example 5.13 Suppose

$$\phi = \{\{pf(x), \neg qz, pz\}, \{\neg px, rg(x, a)\}\}$$

Then Fig. 5.7 shows a refutation of ϕ.

```
======= input processing ==============
set(binary_res).                    # select binary resolution
   dependent: set(factor).          # factorization and unit deletion are set by
   dependent: set(unit_deletion).   # default when binary resolution is selected
clear(factor).                      # both defaults are unset
clear(unit_deletion).
assign(stats_level,0).              # in this case, we are not interested in
                                    # run time statistics
list(usable).                       # empty
end_of_list.

list(sos).                          # the clauses that we know are contradictory
1 [] a.                             # are put in the set of support;
2 [] -a|b.                          # since we do not claim to have any a priori
3 [] -a| -b|c.                      # knowledge of which clauses will produce a
4 [] -c.                            # contradiction, everything is put in the SOS
end_of_list.
======= end of input processing =======
```

Figure 5.9: Binary resolution in propositional logic: input.

5.5 Otter

A state-of-the-art resolution-style theorem-proving program for first-order logic with equality, is Otter (Organized Techniques for Theorem-proving and Effective Research). From the manual:

> Otter includes the inference rules binary resolution, hyperresolution, UR-resolution, and binary paramodulation (as opposed to hyperparamodulation). Some of its other abilities and features are conversion from first-order formulas to clauses, forward and back subsumption, factoring, weighting, answer literals, term ordering, forward and back demodulation, evaluable functions and predicates, and Knuth-Bendix completion. Otter is coded in C, is free, and is portable to many different kinds of computer. Version 0.9 of Otter was distributed at CADE-9 in May 1988, version 1.0 was released in January 1989. Otter 3.0 and patches have been released from 1994 onwards.

Features of Otter include the following:

1. Statements of the problem may be input either with first-order formulas or with clauses (a clause is a disjunction with implicit universal quantifiers and no existential quantifiers). If first-order formulas are input, Otter translates them to clauses.
2. Forward demodulation rewrites and simplifies newly inferred clauses with a set of equalities, and back demodulation uses a newly inferred equality (which has been added to the set of demodulators) to rewrite all existing clauses.
3. Forward subsumption deletes an inferred clause if it is subsumed by any existing clause, and back subsumption deletes all clauses that are subsumed by an inferred clause.
4. A variant of the Knuth-Bendix method can search for a complete set of reductions and help with proof searches.
5. Weight functions and lexical ordering decide the importance of clauses and terms.
6. Answer literals can give information about the proofs that are found.
7. Evaluable functions and predicates build in integer arithmetic, Boolean operations, and lexical comparisons and enable users to "program" aspects of deduction processes.

Although Otter has an autonomous mode, most work with it involves interaction with the user. After the user has encoded a problem into first-order logic or into clauses, he or she usually

```
========== start of search ==========   # Otter starts resolution

given clause #1: (wt=1) 1 [] a.     # get the first clause from the SOS with
                                    # smallest weight wt=1; becomes given clause
given clause #2: (wt=1) 4 [] -c.    # nothing interesting happens, so pick the
                                    # second clause from the SOS and
given clause #3: (wt=2) 2 [] -a|b.          # get the third clause (weight=2)
** KEPT (pick-wt=1): 5 [binary,2.1,1.1] b.   # clause #2 resolves with clause #1
5 back subsumes 2.                          # this clause is more specific than
                                            # the second clause -a|b, so the
                                            # latter is deleted (by default)
given clause #4: (wt=1) 5 [binary,2.1,1.1] b.    # clause number 5 is the fourth
                                                  # given clause; produces nothing
given clause #5: (wt=3) 3 [] -a| -b|c.           # clause number 3 is the fifth
** KEPT (pick-wt=2): 6 [binary,3.1,1.1] -b|c.    # clause 6,7, and 8 are produced
** KEPT (pick-wt=2): 7 [binary,3.2,5.1] -a|c.    # and retained (i.e. kept)
** KEPT (pick-wt=2): 8 [binary,3.3,4.1] -a| -b.
6 back subsumes 3.                              # clause number 3 is deleted
                                                # because less specific
given clause #6: (wt=2) 6 [binary,3.1,1.1] -b|c.
** KEPT (pick-wt=1): 9 [binary,6.1,5.1] c.

----> UNIT CONFLICT at   0.00 sec ----> 10 [binary,9.1,4.1] $F. # clause 9 clashes
                                                               # with clause 4
```

Figure 5.10: Binary resolution in propositional logic: search.

chooses inference rules, sets options to control the processing of inferred clauses, and decides which input formulas or clauses are to be in the initial set of support and which (if any) equalities are to be demodulators. If OTTER fails to find a proof, the user may wish to try again with different initial conditions. In the autonomous mode, the user inputs a set of clauses and/or formulas, and OTTER does a simple syntactic analysis and decides inference rules and strategies. The autonomous mode is frequently useful for the first attempt at a proof.

A sample proof of a propositional binary resolution in OTTER is given in
Fig. 5.9-5.11 (pp. 112-114).

Otter's main loop

OTTER uses the following procedure to produce (and delete) clauses [66]:

1. Select the lightest, or one of the lightest, clauses from the set-of-support `sos`, and move it from `sos` to `usable`. This will be the "given" clause.
2. Infer all clauses that have the given clause as one parent, and clauses from the usable list as their other parents.
3. Foreach newly inferred clause, see if it is worthwhile to keep it.
4. Append to the `sos` each new clause that is not discarded as a result of processing

With (2), notice that one or more of the other parents may be the given clause itself, since it is now on the usable list. Further, exactly which clauses are inferred of course depends on the rules that are in effect. Stage (3) is the cleaning stage. Newly inferred clauses are simplified, renamed, and screened for redundancy. In Table 5.2 on page 123 it is exactly described how newly inferred clauses are processed.

```
---------------- PROOF ----------------    # Otter has found a proof (in the form
                                           # of a refutation) and starts printing
 1 [] a.                                   # it
 2 [] -a|b.
 3 [] -a| -b|c.
 4 [] -c.                                  # a printed proof is useful for
 5 [binary,2.1,1.1] b.                     # inspection
 6 [binary,3.1,1.1] -b|c.
 9 [binary,6.1,5.1] c.
10 [binary,9.1,4.1] $F.

------------ end of proof -------------
============ end of search ============
```

Figure 5.11: Binary resolution in propositional logic: proof.

5.6 Equality

As already indicated[2] the addition of equality yields expressive power at the price of proof complexity. With equality rules present, theorem-provers have so many paths to explore that smart methods are necessary to deal with equality.

Two important resolution rules that deal with equality are demodulation and paramodulation.

```
======= input processing ==============
set(binary_res).
   dependent: set(factor).
   dependent: set(unit_deletion).
assign(stats_level,0).

list(usable).
1 [] -F(x,y) | F(x,S(y)).  % if two such persons X and Y are friends, ...
2 [] -F(x,y) | F(y,x).     % being friends is a symmetric relation
end_of_list.

list(sos).
3 [] F(S(D),S(E)).         % Dick's spouse is a friend of Eve's spouse
end_of_list.

list(passive).
4 [] -F(D,E).              % denial of what has to be shown
end_of_list.

list(demodulators).
5 [] S(S(x))=x.            % in a marriage, you are the spouse of your spouse
end_of_list.               % there is nothing you can do about that..
======= end of input processing =======
```

Figure 5.12: Puzzle of friends: input.

[2] At the beginning of Section 3.8 on page 75

Demodulation

Demodulation, or *rewriting*, is based on the idea that every =-equivalence class of terms should have a canonical member to which all other elements should be reduced. For example, the symbols, or strings, "1.0000," "one," "1," and "`$DIFF`$(2,1)$" all mean the same thing, namely, one. [In OTTER, `$DIFF`$(2,1)$ means: the string that represents the difference of the numbers that are represented by the strings 2 and 1.] The idea is that, to prevent dealing with different representations of the same thing, we may adopt the convention that all things equal to the same thing should be rewritten into a canonical representation of that thing. In this case, for example, we may stipulate that all things equal to one are mapped onto the symbol 1. In this way the number of redundant terms and, hence, the number of redundant clauses is significantly reduced.

Example 5.14 Some repeated demodulations.

	Predicates to Be Demodulated	Demodulators	Demodulants
(1)	$pf(f(f(a)))$	$f(x) = x$	pa
(2)	$pf(a, f(a, f(a, f(a, x))))$	$f(a, x) = x$	px
(3)	$pf(f(f(f(a, a), a), a), x)$	$f(a, x) = x$	px
(4)	$pf(f(f(f(a, a), a), a), x)$	$f(x, a) = x$	$pf(a, x)$

A clause that cannot be further reduced with rewrite rules is said to be in *normal form* or *irreducible*.

The term "demodulation" was introduced by Wos *et al.* [130, 129]. As Kalman writes [48], the power and uses of demodulation were vastly underestimated at the time it was introduced. Nowadays, procedures similar to OTTER's demodulation is often called *rewriting*, or *term rewriting*.

Example 5.15 (Demodulation) Consider the following "puzzle of friends" (From [48].)

1. Dick's spouse (i.e., Dick's wife) is a friend of Eve's spouse (i.e., Eve's husband).
2. If two such persons x and y are friends, then x is also a friend of y's spouse.

Further,

a. Being friends is a symmetric relation.
b. In a marriage, you are the spouse of your spouse.

The problem is to show that Dick is a friend of Eve. If this problem is translated in clause sets (Fig. 5.12 on the facing page) and run in OTTER (Fig. 5.13-5.14, pp. 116-116), we see that derived literals such as `-F(S(S(D)),E)`, `-F(D,S(S(E)))`, and `-F(S(S(S(S(D)))),E)`, are immediately rewritten, or demodulated, into `-F(D,E)`. Similarly, derived literals such as `F(S(S(D)),S(E)))`, `F(D,S(S(S(E))))`, and `F(S(S(S(S(D)))),s(E))`, are immediately rewritten (demodulated) into `F(D,S(E))`.

Demodulation is normally used to post-process (simplify) newly generated clauses. When `demod_inf` is set, however, demodulation is set as an inference rule. In that case the given clause is copied and then processed just like any newly generated clause. Treating demodulation as an inference rule is useful when term rewriting is the main objective.

Paramodulation

Paramodulation is based on Leibniz' law for replacement of equals by equals, and may be considered as a generalization of this law. The difference between paramodulation and demodulation is that paramodulation is a technique to infer new clauses from other clauses, while demodulation is a rewrite technique to simplify existing clauses. (Another term for

```
=========== start of search ===========

given clause #1: (wt=5) 3 [] F(S(D),S(E)).
** KEPT (pick-wt=5): 6 [binary,3.1,2.1] F(S(E),S(D)).
** KEPT (pick-wt=4): 7 [binary,3.1,1.1,demod,5] F(S(D),E).

given clause #2: (wt=4) 7 [binary,3.1,1.1,demod,5] F(S(D),E).
** KEPT (pick-wt=4): 8 [binary,7.1,2.1] F(E,S(D)).

given clause #3: (wt=4) 8 [binary,7.1,2.1] F(E,S(D)).
** KEPT (pick-wt=3): 9 [binary,8.1,1.1,demod,5] F(E,D).

given clause #4: (wt=3) 9 [binary,8.1,1.1,demod,5] F(E,D).
** KEPT (pick-wt=3): 10 [binary,9.1,2.1] F(D,E).

----> UNIT CONFLICT at   0.00 sec ----> 11 [binary,10.1,4.1] $F.
```

Figure 5.13: Puzzle of friends: search.

```
---------------- PROOF ----------------

1 [] -F(x,y)|F(x,S(y)).
2 [] -F(x,y)|F(y,x).
3 [] F(S(D),S(E)).
4 [] -F(D,E).
5 [] S(S(x))=x.
7 [binary,3.1,1.1,demod,5] F(S(D),E).
8 [binary,7.1,2.1] F(E,S(D)).
9 [binary,8.1,1.1,demod,5] F(E,D).
10 [binary,9.1,2.1] F(D,E).
11 [binary,10.1,4.1] $F.

------------ end of proof -------------
============ end of search ============
```

Figure 5.14: Puzzle of friends: proof.

demodulation is term rewriting.) Another difference between paramodulation and demodulation is that paramodulation is more powerful. In fact, paramodulation is resolution-complete for first-order logic with equality, whereas demodulation is not. By its power, paramodulation is also more "dangerous" than demodulation, in the sense that it has the potential to produce a large number of clauses.

Fig. 5.15 on the next page shows what happens with paramodulation. First we need two clauses, C and D, of which the first must contain a literal in the form of an equality ($s = t$, here). D, then, must contain a literal, K in this case, with a term, s', that can be unified with s. To emphasize that literal K contains term s', we write $K(s')$. (Thus, K is *not* a function of some sort.) $D\sigma$, now contains a literal $K(s')\sigma$. This literal is syntactically equal to $K(s'\sigma)$, because K is a predicate symbol, and predicate symbols do not contain variables that can be substituted. Hence $K(s')\sigma$ is syntactically equal to $K(s\sigma)$. The latter justifies clashing $K(s\sigma)$ against $t\sigma$, so that we end up with paramodulant P. The soundness of such a maneuver has to be proven, of course. Since the paramodulation scheme (Fig. 5.15 on the facing page) resembles that of the scheme of binary resolution rule (Fig. 5.8 on page 110), and a proof of the soundness of the latter has already been given (p. 110), a proof of the soundness of the paramodulation rule can now relatively easily be produced, since it will run along the same lines as the proof of the Resolution

Lemma on page 110. For now, I leave the proof of the soundness of the paramodulation rule as an exercise.

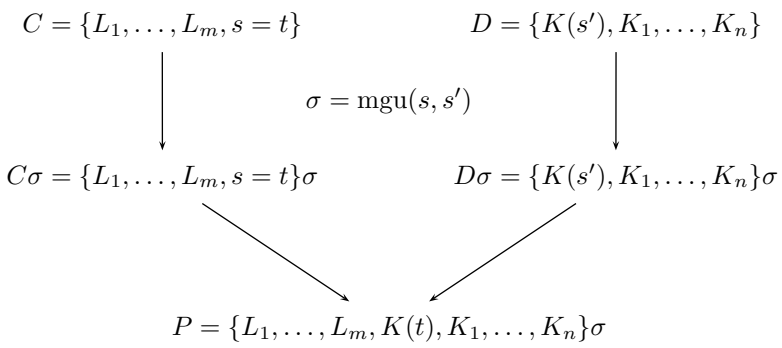


Figure 5.15: One way of paramodulating from C into D. Paramodulant is P.

In paramodulation, the clause substituted into is called the *into* clause, and the clause containing the equality is called the *from* clause. Thus, we substitute "from" a clause containing an equality "into" another clause. For example, the result of paramodulating from the clause $a = b$ into the clause pa is the clause pb.

For the sake of convenience it is assumed in Fig. 5.15 that both parent clauses already have been standardized apart. Further, Fig. 5.15 actually shows one way of paramodulating from C into D. The other version of paramodulation occurs when s and t are interchanged.

Paramodulation can handle cases that are beyond the reach of demodulation. Consider the following example:

i. Rob's father is Pete.
ii. Pete's mother is Eve.
iii. The mother of the father of x is the grandmother of x.

The problem is to show that Eve is the grandmother of Pete. Ordinary demodulation is of no

```
---------------- PROOF ----------------

1 [] father(rob)=pete.
2 [] mother(pete)=eve.
3 [] grandmother(mother(father(x)),x).
4 [] -grandmother(eve,rob).
5 [para_into, 4.1.1, 2.1.2] -grandmother(mother(pete),rob).
6 [para_into, 5.1.1.1, 1.1.2] -grandmother(mother(father(rob)),rob).
7 [binary, 6.1, 3.1] $F.

------------ end of proof -------------
```

Figure 5.16: The problem of the grandmother: proof.

help here, because `grandmother(mother(father(x)),x)` and `-grandmother(eve,rob)` cannot be unified (which is necessary prior to demodulation). What does work, however, is paramodulating (2) into (4), which gives us `-grandmother(mother(pete),rob)`, and then paramodulating (1) into (5) to obtain (6).

	"From" clause	*"Into" clause*	*Paramodulants*
(1)	$a = b$	pa	pb
(2)	$c = d$	qd	qc
(3)	$e = f$	re, e	rf, e, and re, f
(4)	$g(h) = i$	$sg(x)$	si, $sg(i)$, and $sg(g(h))$
(5)	$j(k(x)) = l(m, x)$	$t\ l(x, n)$	$t\ j(k(n))$, $t\ l(l(m, x), n)$, and $t\ l(j(k(x)), n)$
(6)	$f(g(x), y) = f(x, r(y))$	$u\ f(g(z), z))$	$u\ f(z, r(z)))$, $u\ f(g(g(r(y))), y))$, $u\ f(g(f(g(x), y)), f(x, r(y))))$, and $u\ f(g(f(x, r(y))), f(g(x), y)))$
(7)	$\{f(x) = a,\ px\}$	$qf(g(x))$	$\{qa,\ pg(y)\}$, $\{qf(g(a)),\ px\}$, and $\{qf(g(f(x))),\ px\}$
(8)	$\{h(x) = b,\ rx\}$	$\{sb,\ th(x)\}$	$\{sh(x),\ th(y),\ rx\}$, $\{sb,\ tb,\ rx\}$, $\{sb,\ th(b),\ rx\}$, and $\{sb,\ th(h(x)),\ rx\}$

Table 5.1: Examples of paramodulation.

Paramodulation diffes from demodulation with regards to the following aspects:
- it allows variable substitution in both arguments of the equation and the substitutee,
- it does not require the clause with the equality to be unit,
- it allows binding of variables outside of the equation, and
- it does not delete the substitutee.

Other examples of paramodulation are given in Table 5.1.

1. The clause pb may also be derived by demodulating pa with the demodulator $a = b$
2. It is not possible to derive qc by demodulating qd with the demodulator $c = d$, since demodulation only replaces instances of the left argument of a demodulator by the corresponding instances of the right argument. Paramodulation enables an instance in the "into" clause of either argument of an equality in the "from" clause to be replaced by the corresponding instance of the other side of that equality.
3. In paramodulation, every occurrence of every subterm of the "into" clause is considered individually for replacement by one side of the equality in the "from" clause. Thus both rf, e and re, f are inferred when we paramodulate once from $e = f$ into re, e. These clauses are immediate rewrites of re, e by the rewrite rule $e = f$. In contrast, the only demodulant of re, e by the demodulator $e = f$ is the clause rf, f. The clause rf, f may be inferred by paramodulating once from $e = f$ into either rf, e or re, f.
4. In demodulation, we are allowed to make a substitution in a demodulator, but we are not allowed to make a substitution in the clause being demodulated. Thus it is not possible to derive si from $sg(x)$ by applying the demodulator $g(h) = i$. In paramodulation, we are allowed to make a substitution in the "into" clause (as well as in the "from" clause), and hence by the respective substitutions $\{h/x\}$, $\{g(h)/x\}$, and $\{i/x\}$ can infer si, $sg(i)$, and $sg(g(h)))$ by paramodulating from $g(h) = i$ into $sg(x)$. Note that the paramodulants $sg(i)$

and $sg(g(h))$) are inferred by paramodulating "into" a variable. Paramodulating into variables can lead to an explosion of the clause set.

Example 5.16 (Possible paramodulants) Suppose we want to paramodulate $j(k(x)) = l(m, x)$ into $T(l(y, n))$.

First, we examine whether the left-hand side of $j(k(x)) = l(m, x)$ can be unified with a sub-term of $T(l(y, n))$. This can only be accomplished by the trivial substitution $\sigma = \{j(k(x))/y\}$. In that case, $T(l(y, n))$ becomes $T(l(\underline{j(k(x))}, n))$ which then paramodulates into $T(l(l(m, x), n))$. There are no other such substitutions.

Next, we examine whether the right-hand side of $j(k(x)) = l(m, x)$ can be unified with sub-term of $T(l(y, n))$. This is possible with $\sigma = \{n/x, m/y\}$ and $\sigma = \{l(m, x)/y\}$, so that we get two extra paramodulants.

Summary (sub-terms that unify are underlined):

		"Into" clause		
	"From" clause	$T(l(x, n))$		
	$\mathbf{j(k(x)) = l(m, x)}$	$\mathbf{T(l(y, n))}$	*Paramodulant*	*Renamed p.m.*
$\sigma = \{j(k(x))/y\}$	$\underline{j(k(x))} = l(m, x)$	$T(l(\underline{j(k(x))}, n))$	$T(l(l(m, x), n))$	same
$\sigma = \{n/x, m/y\}$	$j(k(n)) = \underline{l(m, n)}$	$T(\underline{l(m, n)})$	$T(j(k(n)))$	same
$\sigma = \{l(m, x)/y\}$	$j(k(x)) = \underline{l(m, x)}$	$T(l(\underline{l(m, x)}, n))$	$T(l(j(k(x)), n))$	same

No paramodulant need be renamed because their first, second, third, etc. variable is already x, y, z, etc.

Example 5.17 (Possible paramodulants) Suppose we want to paramodulate the clause set $\{h(x) = b, R(x)\}$ into the clause set $\{S(b), T(h(y))\}$.

First, we examine whether the left-hand side of the clause $h(x) = b$ can be unified with a sub-term of a clause of $\{S(b), T(h(y))\}$. This can only be accomplished by $\sigma = \{x/y\}$. In that case, $h(x) = b$ remains unaltered while $\{S(b), T(h(y))\}$ changes into $\{S(b), T(h(x))\}$ which then paramodulates into $\{S(b), T(b), R(x)\}$. There are no other such substitutions.

Next, we examine whether the right-hand side of $h(x) = b$ can be unified with sub-term of a clause of $\{S(b), T(h(y))\}$. Since b is a constant, this is only possible with the empty substitution, so that $\{S(\underline{b}), T(h(y))\}$ paramodulates into $\{S(h(x)), T(h(y)), R(x)\}$.

Summary (sub-terms that unify are underlined):

		"Into" clause		
	"From" clause	$\{S(b), T(h(x))\}$		
	$\mathbf{\{h(x) = b, R(x)\}}$	$\mathbf{\{S(b), T(h(y))\}}$	*Paramodulant*	*Renamed p.m.*
$\sigma = \{x/y\}$	$\{\underline{h(x)} = b, R(x)\}$	$\{S(b), T(\underline{h(x)})\}$	$\{S(b), T(b), R(x)\}$	same
$\sigma = \varnothing$	$\{h(x) = \underline{b}, R(x)\}$	$\{S(\underline{b}), T(h(y))\}$	$\{S(h(x)), T(h(y)), R(x)\}$	same

As with the previous example, renaming is not necessary.

PROBLEMS (Sec. 5.6)

1. (a) Translate the following sentence into the language of first-order logic: "every person has a father".

(b) Skolemize the formula found at (1a).

(c) What is the intuitive meaning of the Skolem function found at (1b)?

2. Consider the first-order structure M with domain all integers, with $<$ and with $f(x) = x + 1$. Show that $(\forall x)(\exists y)(x > y))$ is not equivalent to its Skolemization, if the Skolem function on x is interpreted as f. Show that both formulas are satisfiable or unsatisfiable.

3. Prove Theorem 5.9 on page 108. You will need the Substitution Lemma on page 56. (Solution.)

4. If the clauses in Column 1 are rewritten by means of the demodulators in Column 2, then what clauses are inferred as demodulants?

	Clauses to be demodulated	Demodulators
(1)	$D(f(x,x), f(a,a))$	$f(x,a) = a$
(2)	$D(f(f(x,x), f(a,a)))$	$f(x,a) = a$
(3)	$D(f(a,a), f(a,x))$	$f(x,a) = a$
(4)	$D(f(f(a,a), f(a,x)))$	$f(x,a) = a$
(5)	$D(f(a,a), f(a,x))$	$f(x,a) = a, f(x,a) = f(a,x)$
(6)	$D(f(f(a,a), f(a,x)))$	$f(x,a) = a, f(x,a) = f(a,x)$

(Solution.)

5. Solve the following problem with binary resolution and demodulation.

- Alice's and Betty's spouses are friends.
- If two persons are friends, then the first person is also a friend of the spouse of the second person.
- Friendship is symmetrical.
- Married persons are the spouse of their spouse.

Are Alice and Betty friends? (Solution.)

6. Show which clauses are generated if $D(f(1+2, 7-3))$ is left-most inner-most reduced with demodulant $f(x,y) \rightarrow x + 2y$:

```
set(demod_inf).
assign(stats_level,0).

list(sos).
D(f($SUM(1,2),$DIFF(7,3))).
end_of_list.

list(demodulators).
f(x,y)=$SUM(x,$PROD(2,y)).
end_of_list.
```

(Solution.)

7. Compute which clauses are generated if Otter is run with

```
set(hyper_res).
assign(stats_level,0).

list(usable).
-P(x,y)|P($SUM(x,y),$PROD(x,y)).
```

```
end_of_list.

list(sos).
P(2,3).
end_of_list.
```

(Solution.)

8. Refute the clause set

$$\{\{\neg px;\, qx;\, rx, f(x)\}, \{\neg px;\, qx;\, sf(x)\}, \{ta\}, \{pa\}, \{\neg ra, z;\, tz\}, \{\neg tx;\, qx\}, \{\neg ty;\, sy\}\}$$

9. Zie Fig. 5.17. Bepaal voor elke rij de clauses die kunnen worden verkregen door de tweede kolom te resolveren met clauses uit de eerste kolom. Pas eventueel factorisatie toe. N.B. 1.

	Clause set	*Actieve clause*
(1)	$-P \mid Q(a,x) \mid Q(y,b)$	P
(2)	$-P \mid Q(a,x) \mid Q(y,b),\ -Q(a,b)$	P
(3)	$-P \mid Q(a,x) \mid Q(y,b),\ P$	$-Q(a,b)$
(4)	$-P \mid Q(a,x)$	$P \mid Q(x,b)$
(5)	$-P \mid Q(a,x),\ -Q(a,b)$	$P \mid Q(x,b)$
(6)	$P(a,x) \mid P(y,b)$	$-P(a,b) \mid Q$
(7)	$P(a,x) \mid P(y,b),\ -Q$	$-P(a,b) \mid Q$
(8)	$P(a,x) \mid P(y,b),\ Q$	$-P(a,b) \mid -Q \mid R$
(9)	$P(a,x,y,z) \mid P(u,b,y,z)$	$-P(x,y,c,z) \mid -P(x,y,u,d)$
(10)	$-P \mid Q(x) \mid Q(y),\ -Q(u) \mid -Q(v)$	P
(11)	$-P \mid Q(x) \mid Q(y),\ -Q(u) \mid -Q(v) \mid$	P
(12)	$-P \mid Q(x) \mid Q(y) \mid R,\ -Q(u) \mid -Q(v),\ -R$	P
(13)	$-P \mid Q(a) \mid Q(x),\ -Q(y) \mid -Q(z)$	P
(14)	$P(a) \mid P(b)$	$-P(x) \mid -P(y)$
(15)	$P(a) \mid P(x)$	$-P(b) \mid -P(x)$
(16)	$P(a) \mid P(x)$	$-P(a) \mid -P(x)$
(17)	$P(x,b) \mid P(a,y)$	$-P(x,y) \mid Q(x,y)$
(18)	$P(x,b) \mid P(a,x)$	$-P(x,y) \mid Q(x,y)$
(19)	$P(x,a) \mid P(a,x)$	$-P(x,y) \mid Q(x,y)$
(20)	$-P(x,y) \mid Q(x,y),\ -P(x,y) \mid R(x,y),\ -Q(x,y) \mid -R(x,y) \mid S(x,y)$	$P(x,b) \mid P(a,x)$

Figure 5.17: Clause sets and active clauses.

Alternatieve notaties voor een clause $\{p, q, r\}$ zijn $p \vee q \vee r$ en $p \mid q \mid r$.

N.B. 2. Het antwoord op onderdeel (1) kan worden gecontroleerd door een op resolutie gebaseerde stellingbewijzer zoals bijvoorbeeld OTTER de volgende input te geven:

```
set(hyper_res).
set(factor).
set(very_verbose).

list(usable).
-P | Q(a,x) | Q(y,b).
end_of_list.

list(sos).
```

```
P.
end_of_list.
```

Analoog voor onderdelen (2)-(20). Meer informatie over (de installatie van) OTTER is te vinden op blz. 253 van dit dictaat. (Solution.)

10. Geef aan wat er gebeurd als de predikaten in linkerkolom gedemoduleerd worden met de gelijkheden in de rechterkolom.

	Te demoduleren predikaten	*Demodulatoren*
(i)	$Pf(a)$	$f(x) = g(x, y)$
(ii)	$Pg(a, b)$	$f(x) = g(x, y)$
(iii)	Pa	$x = f(x)$

(Solution.)

11. Gegeven is de clause $C = \{Px, Qf(x, f(x, f(x, y))), Rz\}$ en de demodulator $f(x, f(x, y)) = a$.

    (a) Normaliseer C met een leftmost-innermost reductiestrategie. Onderstreep telkens de redex (reduceerbare expressie) die in een volgende stap gereduceerd wordt.

    (b) Normaliseer C met een leftmost-outermost reductiestrategie. Onderstreep ook hier.

    (c) Zijn beide normaalvormen gelijk?

    (Solution.)

12. Geef alle paramodulanten van de volgende paren.

	"Vanuit" clause	*"In" clause*
(i)	$a = b$	Pa
(ii)	$c = d$	Qd
(iii)	$e = f$	Re, e
(iv)	$g(h) = i$	$Sg(x)$
(v)	$j(k(x)) = l(m, x)$	$Tl(x, n)$
(vi)	$\{f(x) = a,\ Px\}$	$Qf(g(x))$

(Solution.)

13. Geef, voor de paren (i), (ii) en (iii) uit de vorige som, alle 2e-generatie paramodulanten. Daarna alle 3e-generatie paramodulanten. (Solution.)

14. Paramoduleer de clause $\{h(x) = b, Rx\}$ in de clause $\{Sb, Th(x)\}$. Er zijn tenminste vier paramodulanten. (Solution.)

5.7 Practice

Problems

Practical problems with resolution are:

1. *Clause retention.* Program keeps too many clauses because it does can't tell important clauses from less important ones.
2. *Inadequate focus.* Program gets lost.
3. *Redundancy.* Program generates same clauses over and over again.
4. *Clause proliferation.* Program generates many clauses, most of which are irrelevant.
5. *Wrong size of deduction step.*

	1.	Renumber variables.
*	2.	Output `new_clause`.
	3.	Demodulate `new_clause` (including $-evaluation).
*	4.	Orient equalities.
*	5.	Apply unit deletion.
	6.	Merge identical literals (leftmost copy is kept).
*	7.	Apply factor-simplification.
*	8.	Discard `new_clause` and exit if `new_clause` has too many literals or variables.
	9.	Discard `new_clause` and exit if `new_clause` is a tautology.
*	10.	Discard `new_clause` and exit if `new_clause` is too 'heavy'.
*	11.	Sort literals.
*	12.	Discard `new_clause` and exit if `new_clause` is subsumed by any clause in usable, sos, or passive (forward subsumption).
	13.	Integrate `new_clause` and append it to sos.
*	14.	Output kept clause.
	15.	If `new_clause` has 0 literals, a refutation has been found.
	16.	If `new_clause` has 1 literal, then search usable, sos, and passive for unit conflict (refutation) with `new_clause`.
*	17.	Print the proof if a refutation has been found.
*	18.	Try to make `new_clause` into a demodulator.
*	19.	Back demodulate if Step 18 made `new_clause` into a demodulator.
*	20.	Discard each clause in usable or sos that is subsumed by `new_clause` (back subsumption).
*	21.	Factor `new_clause` and process factors.

Note:

i. Steps marked with * are optional.

ii. Steps 19–21 are delayed until steps 1–18 have been applied to all clauses inferred from the active given clause.

Table 5.2: How OTTER handles newly inferred clauses. (After [66].)

6. *Wrong demodulator choice*. The canonical form of terms is not clear to the program.
7. *Bad heuristics*. Bad choices of one of the folowing aspects: respresentation, infereces rule(s), strategy, canonicalization of terms (demodulators), clause reduction.
8. *Inefficient data-structures*.

Besides these practical problems, theorem provers can be tested and compared against each other by running them on a collection of benchmark problems. One such collection is the TPTP library.

The TPTP library

The TPTP (Thousands of Problems for Theorem Provers) Problem Library is the de-fact standard library of test problems for automated theorem provers. By means of a websearch you may find the library and get connected to it.

The TPTP supplies the ATP community with the following:

1. A comprehensive list of the ATP test problems that are available today.

2. New generalized variants of those problems whose original presentation is hand-tailored towards a particular automated proof.
3. The availability of these problems via FTP in a general-purpose format, together with a utility to convert the problems to existing ATP formats. (Currently the METEOR, MGTP, OTTER, PTTP, SETHEO, and SPRFN formats are supported, and the utility can easily be extended to produce any format required.)
4. A comprehensive list of references and other information on each problem.
5. General guidelines outlining the requirements for ATP system evaluation.

TPTP is the work of Geoff Sutcliffe and Christian Suttner. The TPTP is maintained and distributed from the Department of Computer Science, James Cook University, Australia. The TPTP is also supported and distributed by the Institut fuer Informatik, TU Muenchen, Germany.

The TPTP is regularly updated with new problems, additional information, and enhanced utilities. Each release of the TPTP is identified by a version number, an edition number, and a patch level, in the form `TPTP v<Version>.<Edition>.<Patch level>`. The version number enumerates major new releases of the TPTP, in which important new features have been added. The edition number is incremented each time new problems are added to the current version. The patch level is incremented each time errors, found in the current edition, are corrected.

The TPTP collection also has system performance results of various ATPs. The ATPs are programmed in almost every kind of programming language varying from ANSI C (most), C++, Scheme, Prolog, and Common Lisp, to plain Shell scripts. Currently, results of the following ATP systems are documented:

`SPTHEO m256n1`, `CLIN 1.0`, `CLIN-E 0.35FM`, `CLIN-S 1.0+`, `CLIN-S 1.0-`, `LINUS 1.0.1`, `DISCOUNT 2.0-GL`, `PROTEIN V2.20`, `DISCOUNT TSM2.1`, `SNARK 990218`, `FDP 0.9.2`, `SETHEO 3.3`, `MUSCADET 2.2`, `EQP 0.9d`, `SPASS 1.03`, `Waldmeister 600`, `Otter 3.0.6`, `Bliksem 1.12`, `MACE 1.4b`, `Gandalf c-1.9c`, `DCTP 0.1`, `Vampire 2.0`, `PizEAndSATO 0.2`, `SCOTT 6.0.0`, `E-SETHEO csp01`, `GandalfSat 1.1`, `E 0.62`, and `S-SETHEO 0.0`.

As an illustration, an exercept of the record of `DISCOUNT 2.0-GL` is shown in Fig. 5.18 on the facing page.

```
ATP system name    : DISCOUNT 2.0-GL
Developed by       : Stephan Schulz (schulz@informatik.tu-muenchen.de)
                     (Proof protocols, Learning)
                     Joerg Denzinger (denzinge@informatik.tu-muenchen.de)
                     (Distribution concept, theory for existentially qualified
                      goals)
                     Werner Pitz (pitz@comsoft.de)
                     (Implementation of inference engine and distribution
                      scheme)
                     Martin Kronenburg (kronburg@informatik.uni-kl.de)
                     (Planning component, his domain recognition engine is
                      reused by the learning system)
Availability       : Email: schulz@informatik.tu-muenchen.de
References         : [DS96]  Denzinger & Schulz (1996), Learning Domain Knowled
TPTP version       : v1.2.1
tptp2X flags       : -f otter (No use of clause type information)
CPU                : SparcStation 10, 50MHz sun4m (SuperSparc Processor)
RAM size           : 128MB
Operating system   : Solaris 2.5.1
Implementation     : C. Compiler: gcc 2.4.5 Flags: -O2 -D SPARC
Resource limits    : 150 seconds
Settings/flags     : -I 5 -D 1 -G 15 -R 2 -T 1 -A -3 -B 1500 -x c_global_learn
Results format     :   Solns     Time  SrhTime
Results values     : Solns   -  P(X) and number of proofs found
                             or M(X) and the number of models found, where X is
                                "C" when the goal is universally quantified,
                                and "P" when the goal is existentially
                                quantified and thus overlapping into the goal
                                is performed.
                             or timeout
                     Time    - CPU time, in seconds
                     SrhTime - Time in proof search, in seconds
Results summary    : 1733 problems (2752 in TPTP v1.2.1, 1019 since Bugfixed)
                     1369 not_tested
                      255 proofs
                        1 models
                      108 timeout
                      21% tested
                      256 successes (proofs + models)
                      70% success, over those tested
                      15% success, overall
Comments           : Tested on unit equality problems only.
                   : Used a knowledge base (implicitly named "KNOWLEDGE")
                     generated from 664 proofs/226 problems
                   : PCL, the "Proof Communication Language" is currently
                     supported only for universally quantified goals.
                   : Various other people work on DISCOUNT. In particular, the
                     learning system profits from proofs found with the
                     distributed system and Matthias Fuchs'
                     (fuchs@informatik.uni-kl.de) analogical learning.
Submitted by       : Stephan Schulz (schulz@informatik.tu-muenchen.de)
```

Figure 5.18: Performance record of DISCOUNT 2.0-GL.

Chapter 6

Satisfiability checkers

Resolution can only be used to prove that a clause set is unsatisfiable. It can not be used to prove that a clause set is satisfiable. To be able to discover satisfiable formulas as well, it is necessary to include satisfiability tests. Guided by a certain heuristic, such tests try to find a satisfying assignment of ϕ. If such an assignment is found, the formula is proven satisfiable and the search can be stopped.

There are two popular satisfiability tests:

1. The first test is based on taking the so-called *gradient* of polynomial transforms of CNFs. The idea is that a proposition is chosen and set to true if its number of unnegated occurrences is higher than its number of negated occurrences, otherwise it is set to false. Subsequently the simple rules are applied, and a next proposition is chosen, until satisfiable assignment is constructed.

2. The other test is a weighted variant of a *greedy local search algorithm* [100]. Starting from a random assignment, the truth values of a number of variables is inverted (set from true to false and vice versa) such that the weight of satisfied clauses increases. If further inversion yields no improvement, the weights of the unsatisfied clauses are increased until an improving inversion comes into existence. In this way, 'difficult' clauses receive large weights and thus are more likely to be satisfied in the end. This process is repeated until either a satisfiable assignment is constructed, or the number of inversions exceeds a certain maximum.

For many satisfiable formulas the local search algorithm is sufficient for finding a model quickly. The next section is devoted to such a local search algorithm, namely GSAT.

6.1 GSAT

Given a clause set S, GSAT tries to find a model m such that $m \models S$ by performing a greedy local search within the space of possible models. The procedure starts with a randomly generated model and then changes ('flips') the assignment of the variable that leads to the largest increase in the total number of satisfied clauses. Such flips are repeated until either a satisfying assignment is found or a preset maximum number of flips (MAX-FLIPS) is reached. This process is repeated as needed up to a maximum of MAX-TRIES times. (Algorithm 2 on the next page.)

GSAT only explores potential solutions that are close to the one currently being considered. More specifically, it explores the set of assignments that differ from the current one on only one variable.

Algorithm 2 GSAT

Input: A clause set S, a value for MAX-FLIPS and a value for MAX-TRIES
Output: A model m, such that $m \models S$, if found

```
1: for i = 1 to MAX-TRIES do
2:    choose a random model m
3:    for j = 1 to MAX-FLIPS do
4:       return m if m ⊨ S
5:       p = a propositional variable such that a change in its truth assignment gives the largest
         increase in the total number of clauses of S that are satisfied by m
6:       m = m with the truth assignment of p reversed
7: return "no satisfying model found"
```

The GSAT procedure requires the setting of two parameters, namely, MAX-FLIPS and MAX-TRIES. MAX-FLIPS says how many flips the procedure will attempt before giving up and restarting, and MAX-TRIES says how many times this search may be restarted. Selman *et al.* discovered that, as a rough guideline, setting MAX-FLIPS equal to a few times the number of variables is sufficient. The setting of MAX-TRIES will generally be determined by the total amount of time that one wants to spend looking for an assignment, which in turn depends on the application.

It should be clear that GSAT could fail to find an assignment even if one exists, i.e. GSAT is incomplete. We will discuss this below.

Local search procedures like GSAT need to be implemented with some care: naively, one simply iterates through the variables, and determines the number of clauses satisfied if that variable is flipped. For L clauses and N variables this takes $\mathcal{O}(NL)$ time. However, if we remember for each variable the change in score if it would flip, we need only to update this information after each flip just by examining each clause that the flipped variable appears in.

formulas		GSAT			DP		
vars	clauses	M-FLIPS	tries	time	choices	depth	time
50	215	250	6.4	$0.4s$	77	11	$1.4s$
70	301	350	11.4	$0.9s$	42	15	$15s$
100	430	500	42.5	$6s$	84×10^3	19	$2.8m$
120	516	600	81.6	$14s$	0.5×10^6	22	$18m$
140	602	700	52.6	$14s$	2.2×10^6	27	$4.7h$
150	645	1500	100.5	$45s$	−−	−−	−−
200	860	2000	248.5	$2.8m$	−−	−−	−−
250	1062	2500	268.6	$4.1m$	−−	−−	−−
300	1275	6000	231.8	$12m$	−−	−−	−−
400	1700	8000	440.9	$34m$	−−	−−	−−
500	2150	10000	995.8	$1.6h$	−−	−−	−−

Table 6.1: Results for GSAT and DPP on hard random 3CNF formulas. (Taken from [100]).

One feature that sets GSAT apart from other model-finders, is that the variable whose assignment is to be changed is chosen at random from those that would give an equally good improvement. Such non-determinism makes it very unlikely that the algorithm makes the same sequence of changes over and over.

Another characteristic feature of GSAT is that it makes *sidesteps*. [GSAT is said to perform a sidestep if it flips a variable without increasing the total number of satisfied clauses.] Selman *et al.* demonstrated the importance of side steps by re-running some experiments in which side steps were forbidden. In such experiments only flips were allowed that increase the number of

satisfied clauses, and else a restart was performed. It turned out that finding an assignment for satisfiable formulas became much harder without the ability to side-step. This is easy to understand if one realizes that the "landscape" over which GSAT moves has many plateaus. GSAT may thus move to "better spots" by side-stepping over such plateaus.

6.2 GSAT for non-clausal formulas

In 1994, Roberto Sebastiani showed that GSAT can also be applied to non-clausal formulas [98]. To show how, we first formally define the criterion for satisfaction on clause sets:

Definition 6.1 The *penalty* of a model m on a clause set S, written $pen(S, m)$, is equal to the number of clauses in S that are made false by m.

Thus, the purpose of GSAT is to find a model m for S with a penalty as low as possible. If the penalty $pen(S, m)$ is equal to zero then the algorithm can stop, because $pen(S, m) = 0$ means $m \models S$. GSAT works by considering proposition variables of a model that, when flipped, bring the penalty down as much as possible. Thus, if we write

$$\Delta(S, m, p) =_{Def} pen(S, m \text{ with } p \text{ flipped}) - pen(S, m) \tag{6.1}$$

then GSAT tries to go downhill and searches for p in the direction where the slope $\Delta(S, m, p)$ is negative.

The penalty function *pen* may now be extended on clause sets to a penalty function *Pen* (note the capital) on arbitrary formulas, in the following way:

For literals L, $Pen(L, m) = pen(m, L)$, that is,

$$Pen(m, L) = \begin{cases} 1 & \text{if } m \not\models L, \\ 0 & \text{otherwise.} \end{cases} \tag{6.2}$$

Further, $Pen^-(m, L) = 1 - pen(m, L)$, and

$$\begin{aligned} Pen(m, \neg\phi_1) &= Pen^-(m, \phi_1) \\ Pen(m, \phi_1 \wedge \phi_2) &= Pen(m, \phi_1) + Pen(m, \phi_2) \\ Pen(m, \phi_1 \vee \phi_2) &= Pen(m, \phi_1) \cdot Pen(m, \phi_2) \\ Pen(m, \phi_1 \supset \phi_2) &= Pen^-(m, \phi_1) \cdot Pen(m, \phi_2) \\ Pen(m, \phi_1 \equiv \phi_2) &= Pen^-(m, \phi_1) \cdot Pen(m, \phi_2) + Pen(m, \phi_1) \cdot Pen^-(m, \phi_2) \end{aligned}$$

and

$$\begin{aligned} Pen^-(m, \neg\phi_1) &= Pen(m, \phi_1) \\ Pen^-(m, \phi_1 \wedge \phi_2) &= Pen(m, \phi_1) \cdot Pen(m, \phi_2) \\ Pen^-(m, \phi_1 \vee \phi_2) &= Pen(m, \phi_1) + Pen(m, \phi_2) \\ Pen^-(m, \phi_1 \supset \phi_2) &= Pen(m, \phi_1) + Pen^-(m, \phi_2) \\ Pen^-(m, \phi_1 \equiv \phi_2) &= (Pen(m, \phi_1) + Pen^-(m, \phi_2)) \cdot (Pen^-(m, \phi_1) + Pen(m, \phi_2)) \end{aligned}$$

(Note the symmetry in the RHS of the last formula.) Sebastiani then proved that, for every arbitrary formula ϕ,

$$Pen(m, \phi) = pen(m, \text{CNF}(\phi)), \tag{6.3}$$

where $\text{CNF}(\phi)$ is a CNF-conversion of ϕ. (Not surprisingly, the prove goes by induction.) So actually, we should work with equivalence classes of CNF-conversions, and then prove that the

penalty for each equivalence class is the same. However, this is unnecessarily strict. It is easy to see that both both Pen and Pen^- can be computed in time linear to the length of ϕ.

That's all. We can now plug in the new function Pen into GSAT and apply GSAT to arbitrary formulas. If we call this new algorithm NC-GSAT (non-clausal GSAT), then it follows from (6.3) that, in hill-climbing (well, actually hill-descending, here), NC-GSAT always returns the same set of proposition variables as in GSAT, so that NC-GSAT performs the same flips and returns the same results as GSAT.

NC-GSAT has more advantages: the current implementation of Selman *et al.*'s GSAT [100] utilizes a highly optimized version of the penalty update function. This function only analyzes the clauses which the last-flipped variable occurs in. This allows a strong reduction in computational cost. In his 1994 article, Sebastiani reports that he has written a conservative extension of Selman's penalty update function that operates on non-clausal formulas as well. Thus, Sebastiani has shown that GSAT can be ported to NC-formulas without loss of computational efficiency.

As Sebastiani notes, it is interesting to compare NC-GSAT with other methods of dealing with non-clausal formulas, such as computing the Tseitin derivative (page 85). The Tseitin method converts propositional formulas into polynomial-size clausal formulas by means of new variables, each representing a subformula of the original input formula ϕ. Unfortunately, the issue of size-polynomiality is valid only if no equivalence symbol occurs in ϕ, as the number of clauses of Tseitin(ϕ) grows exponentially with the number of $\equiv$-symbols in ϕ. Even worse, the introduction of k new variables enlarges the search space of Tseitin(ϕ) by a factor 2^k and reduces strongly the solution ratio, since any model for Tseitin(ϕ) is also a model for ϕ, but for any model of ϕ we only know that one of its 2^k extensions is a model of Tseitin(ϕ).

Shortcomings of GSAT

GSAT was a major breakthrough in satisfiability-checking, but it suffers from shortcomings that are characteristic to all connectionistic methods. One such shortcoming is that, due the algorithm and the structure (the "landscape") of the search space, GSAT often "wanders" through large plateaus of truth-assignments that show no variation. On these plateaus, the algorithm has no clue as to which direction it should traverse and is thus at the mercy of the random part of the algorithm that is responsible for the selection of the next variable to flip.

Another characteristic shortcoming of GSAT is that it can easily be misled into exploring the wrong part of the search space. Cf. Exercise 1 on the facing page.

A third deficiency of GSAT is that the search is non- deterministic so that trials are not reproducible. This last shortcoming may be remedied by using a pseudo-random generator.

6.3 Improvements on GSAT

GSAT is a mix of success and obvious shortcomings. This led to a host of improvements on GSAT.

In 1993 Selman *et al.* proposed to extend GSAT with a random component so as to allow it to escape from local minima. The resulting algorithm is called the *random walk strategy*:

- With probability p, flip a variable that occurs in some unsatisfied clause.
- With probability $1 - p$, follow the standard GSAT scheme, i.e., make the best possible local move.

The smart thing about this algorithm is that upward moves (which would otherwise lead us astray) are now used to "repair" unsatisfied clauses.

The WalkSAT algorithm [31, 101] takes the random walk idea one step further and makes it the central component of the algorithm. WalkSAT randomly selects an unsatisfied clause, and then does the following: if the selected clause has a variable that can be flipped without breaking other clauses, then that variable is flipped. Else, with probability p we flip the variable that brakes the fewest clauses, else with probability $1-p$ we flip a random variable in the selected clause.

PROBLEMS (Sec. 6.3)

1. A major shortcoming of GSAT is that it can be misled into exploring the wrong part of the search space. The following example is of Selman *et al.* [100]. (The numbers represent propositional variables.)

$$\begin{array}{lllll} \mathcal{C} = \{\{1,\neg 2,3\}, & \{1,\neg 3,4\}, & \{1,\neg 4,\neg 2\}, & \{1,5,2\}, & \{1,\neg 5,2\}, \\ \{\neg 1,\neg 6,7\}, & \{\neg 1,\neg 7,8\}, & \ldots, & \{\neg 1,\neg 98,99\}, & \{\neg 1,\neg 99,6\}\}. \end{array}$$

   (a) How many clauses are there in $\mathcal{C}$? (Solution.)

   (b) Investigate the impact of p_1 (i.e., 1) on the number of satisfied clauses in $\mathcal{C}$. What happens if $p_1 = 0$? What if $p_1 = 1$? What is your conclusion with regards to the truth value of p_1 and the satisfiability of $\mathcal{C}$? (Solution.)

   (c) Give an optimal assignment of truth values to $p_1, \ldots, p_{99}$. (Solution.)

   (d) Given an average truth assignment, describe what happens if GSAT considers to flip p_1. (Solution.)

   (e) Describe a situation in which GSAT moves to the optimal solution. (Solution.)

2. There are different ways to extend GSAT in such a manner that it does not get stuck in local minima. One way to extend GSAT's inner loop is with *simulated annealing*. Simulated annealing introduces uphill moves into local search by using a noise model based on statistical mechanics (Kirkpatrick et al. 1982). The idea is to start with a randomly generated truth assignment. We then repeatedly pick a random variable, and compute (6.1), i.e., the change in the number of unsatisfied clauses when that variable is flipped. If $\Delta(S,m,p) < 0$ (a downhill or sideways move), make the flip. Otherwise, flip the variable with probability $e^{-\Delta/T}$, where T is a formal parameter called the temperature. The temperature may be either held constant, or slowly decreased from a high temperature to near zero according to a cooling schedule. One often uses geometric schedules, in which the temperature is repeatedly reduced by multiplying it by a constant factor $c < 1$. Given a finite cooling schedule, simulated annealing is not guaranteed to find an assignment that satisfies all clauses. Therefore Selam *et al.*'s SA-extension of GSAT uses multiple restarts as well.

   The basic GSAT algorithm is very similar to annealing at temperature zero.

   (a) Formulate the SA-extension GSAT algorithm, analogous to Algorithm 2 on page 128.

   (b) What kind of algorithm do you get when the simulated temperature is equal to zero?

   (c) Same question as (2b), but now what if temperature does not decrease (and remains clamped to a high value).

6.4 Propositional formula checkers

Theorem proving amounts to verifying whether ψ follows from $\phi_1, \ldots, \phi_n$, for some $\phi_1, \ldots, \phi_n$ and ψ. There are two possibilities: either $\phi_1, \ldots, \phi_n \vdash \psi$ or $\phi_1, \ldots, \phi_n \nvdash \psi$.

1. If "$\vdash$", then $\phi_1 \wedge \ldots \wedge \phi_n \wedge \neg\psi$ is not satisfiable, which can be shown by means of a refutation method, such as resolution or the tableaux method.

2. If "$\not\vdash$", then $\phi_1 \wedge \ldots \wedge \phi_n \wedge \neg\psi$ is satisfiable and this can be proven by finding a countermodel.

It is true that satisfiability as well as unsatisfiability can be expressed by an existential statement. By definition, ϕ is satisfiable if there *exists* a satisfying assignment for ϕ. On the other hand, ϕ is unsatisfiable if there *exists* a refutation (resolution refutation or tableau refutation) of ϕ. As discussed above, there is a catch to this apparent symmetry. In general, writing down a resolution proof is harder than writing down a satisfying assignment. This non-symmetry is caused by the fact that the satisfiability problem (aka SAT) is NP-complete, and propositional provability is co-NP-complete (page 37).

The current state of affairs in ATP is that resolution is the most powerful refutation method. State-of-the-art theorem provers are based on the manipulation of clause sets, and not on the manipulation of refutation trees. Still, a striking feature of the tableaux method is that the search for a countermodel coincides with the search for a refutation. If the sequent in question is falsifiable, then the tableau method will halt at one of the leaves with a countermodel. In it is not, then the tableau will close, so that the refutation fails and the sequent has been proven. Thus, a tableau is *always* useful. A closed tableau yields a proof and an open tableau yields a countermodel. This positive feature is unfortunately not shared by resolution refutation. Resolution can only prove (1), but it can never disprove it, i.e. it can never prove (2). To prove (2), so-called model-checking techniques are used. A model-checking technique tries to guess countermodels for $\phi_1 \wedge \ldots \wedge \phi_n \wedge \neg\psi$.

A *propositional formula checker* is an automated theorem proving program that is able to prove valid formulas, and disprove invalid formulas. From the discussion above it follows that the tableau method is "complete enough" to form the basis for a propositional formula checker, while resolution is not. Resolution always needs to be supplemented with some sort of model-checking technique in order to qualify as a propositional formula checker.

6.5 Challenges in propositional reasoning

Inspired by the rapid developments in the area of satisfiability testing, the inventors of GSAT announced ten challenges in propositional reasoning and search that pertain to three core areas: systematic search, stochastic search, and problem encodings [99]. These challenges often frequently arise in informal discussions among researchers. The main evaluation criteria that are adopted in the satisfiability testing (SAT) community is empirical performance on shared benchmark problems. Selman *et al.* believe that this remains the best way to evaluate responses to any of the algorithmic challenges we will present.

Reasonably good benchmark collections of satisfiability problems, mainly in CNF form, are publically available from a number of sources. These include the DIMACS collection, which was used as a testbed for a large number of algorithms as reported in (Trick and Johnson 1996); a collection of circuit diagnosis problems encoded as SAT (Larrabee 1992); the planning problems Kautz and Selman (1996) devised for their SATPLAN system; hardware and software verification problems from the model checking community; and others. All these collections have the property that published papers exist specifying the best known algorithms and running times for solving each of their instances. See also the TPTP-library on page 123.

We follow the global outline of [99]:

SAT problems

Two specific open SAT problems are particularly interesting. The first is to develop a way to prove that unsatisfiable 700 variable 3-CNF formulas, randomly generated in the "hard" region where the ratio of variables to clauses is 4.3, are in fact unsatisfiable (Mitchell et al. 1992; Crawford and Auton 1993). Randomly-generated satisfiable formulas of this size are regularly

solved by GSAT, but they are unable to prove unsatisfiability, and no systematic algorithm has been able to solve hard random formulas of this size. A generator for these hard random problem instances can be found at the DIMACS benchmark archive (Trick and Johnson 1996).

CHALLENGE 1: (1-2 yrs) *Prove that a hard 700 variable random 3-SAT formula is unsatisfiable.*

The second challenge problem is satisfiable. It is an encoding of a 32-bit parity problem, that appears in the DIMACS benchmark set. It appears to be too large for current systematic algorithms. It also defeats hillclimbing techniques such as GSAT.

CHALLENGE 2: (2-5 yrs) *Develop an algorithm that finds a model for the DIMACS 32-bit parity problem.*

For the second challenge, of course, the algorithm should not be told in advance the known solution! Given the amount of effort that has been spent on these two instances, any algorithm solving one or both will have to do something significantly different from current methods.

Systematic search

All of the best systematic methods for propositional reasoning are based on creating a resolution proof tree. This includes depth-first search algorithms such as the Davis-Putnam procedure, where the proof tree can be recovered from the trace of the algorithm's execution, but is not explicitly represented in a data structure (the algorithm only maintains a single branch of the proof tree in memory at any one time). Most work on systematic search concentrates on heuristics for variable-ordering and value selection, all in order to the reduce size of the tree. However, there are known fundamental limitations on the size of the shortest resolution proofs for certain problems. For example, "pigeon hole" problems (showing that n pigeons cannot fit in $n-1$ holes) are intuitively easy, but shortest resolution refutation proofs are of exponential length. Shorter proofs do exist in more powerful proof systems. Examples of proof systems more powerful than resolution include extended resolution, which allows one to introduce new defined variables, and resolution with symmetry-detection, which uses symmetries to eliminate parts of the tree without search.

CHALLENGE 3: (2-5 yrs) *Demonstrate that a propositional proof system more powerful than resolution can be made practical for satisfiability testing.*

Selman *et al.* also offered a challenge to show that the well-developed body of tools and techniques from Operations Research does in fact have something new to offer for propositional reasoning.

CHALLENGE 4: (2-5 yrs) *Demonstrate that integer programming can be made practical for satisfiability testing.*

Stochastic search

Stochastic algorithms are inherently incomplete, because if they fail to find a model for a formula one cannot be certain that the formula is unsatisfiable. This has led to an asymmetry in our ability to solve satisfiable and unsatisfiable instances drawn from the same problem distribution. Stochastic algorithms can solve hard random satisfiable formulas containing thousands of variables, but we cannot solve unsatisfiable instances of the same size. The step in the local search would try to transform the proof into one that rules out a larger fraction of assignments.

CHALLENGE 5: (5-10 yrs) *Design a practical stochastic local search procedure for proving unsatisfiability.*

CHALLENGE 6: (1-2 yrs) *Improve stochastic local search on structured problems by efficiently handling variable dependencies.*

A general question is: can we develop a procedure that leverages the strengths of systematic and stochastic search? The obvious way, of course, is to simply run good implementations of each approach in parallel. But is there a more powerful way of combining the two?

CHALLENGE 7: (1-2yrs) *Demonstrate the successful combination of stochastic search and systematic search techniques, by the creation of a new algorithm that outperforms the best previous examples of both approaches.*

Challenges for Problem Encodings

The value of research on propositional reasoning ultimately depends on our ability to find suitable SAT encodings of real-world problems. As discussed in the introduction, there has been significant recent progress along this front. Examples include classical constraint-based planning, problems in finite algebra, verification of hardware and software, scheduling, circuit synthesis and diagnosis, and other domains. Experience has shown that different encodings of the same problem can have vastly different computational properties.

CHALLENGE 8: (5-10 yrs) *Characterize the computational properties of different encodings of a realworld problem domain, and/or give general principles that hold over a range of domains.*

For completeness' sake we list the last two research problems:

CHALLENGE 9: (1-2 yrs) *Find encodings of real world domains which are robust in the sense that "near models" are actually "near solutions".*

CHALLENGE 10: (2-5 yrs) *Develop a generator for problem instances that have computational properties that are more similar to real-world instances.*

It must be noted that some of these problems has already been solved. Challenge 2 is such an example: in 1998, this problem was solved by GT6, an algorithm based on extended digital functions representation and inference. Follow-ups to [99] come out all the time. Cf., e.g., CiteSeer for a reasonably complete overview of what happened since then.

Part II

Non-deductive forms of automated reasoning

Motivation

The following is taken from Simon Parsons' and Anthony Hunter's [73]. It is the introduction to an article entitled "A Review of Uncertainty Handling Formalisms," and gives in my opinion a fair picture of the state of affairs in this area. They start their introduction by arguing that

"(...) practical AI systems are constrained to deal with imperfect knowledge, and are thus said to reason approximately under conditions of ignorance. Attempts to deal with ignorance [49, 104] for example, often attempt to form general taxonomies relating different types and causes of ignorance such as uncertainty, incompleteness, dissonance, ambiguity, and confusion. A taxonomy, taken from Smithson [104], that is perhaps typical, is given in Figure 6.1. The importance of such taxonomies is not so much that they accurately characterize the nature of ignorance that those who build practical AI systems have to deal with—they are far too open to debate for that—but more that they allow distinctions to be drawn between different types of ignorance. This has motivated the development of a multitude of diverse formalisms each intended to capture a particular nuance of ignorance, each nuance being a particular leaf in Smithson's taxonomy tree. The most important distinction is that made between what Smithson calls uncertainty and absence, though this may be confused by a tendency in the literature to refer to "absence" as "incompleteness". Uncertainty is generally considered to be a subjective measure of the certainty of something and is thus modelled using a numerical value, typically between 0 and 1 with 0 denoting falsity and 1 denoting truth. Absence is the occurrence of missing facts, and is usually dealt with by essentially logical methods. The wide acceptance of the suggestion that uncertainty and absence are essentially different, and must therefore be handled by different techniques has lead to a schism in approximate reasoning between the "symbolic camp" who use logical methods to deal with absence and the "numerical camp" who use quantitative measures to deal with uncertainty.

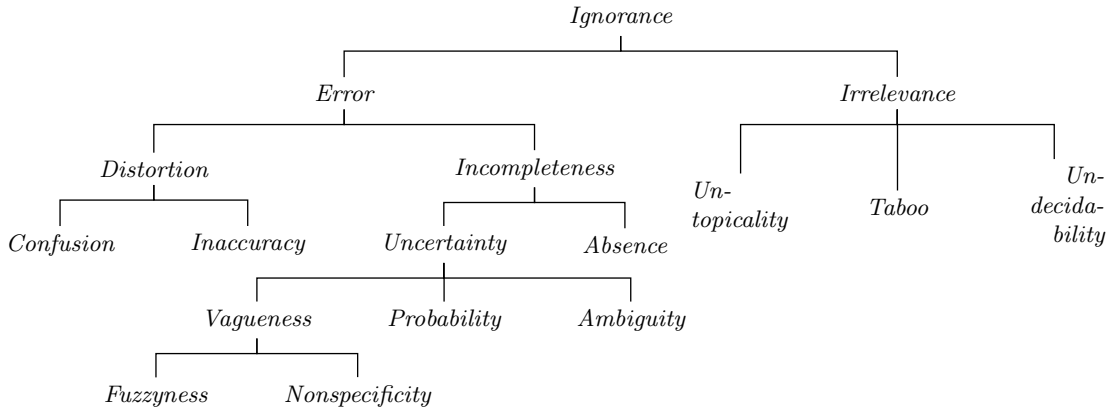


Figure 6.1: Smithson's taxonomy of different types of ignorance.

This void between symbolic and numerical techniques, which remained unaddressed for many years as researchers concentrated on the finer technical details of their particular formalism, can be seen as symptomatic of the way in which research into approximate reasoning has been pursued. For many years researchers indulged in ideological slanging matches of almost religious fervour in which the formalism that they championed was compared with its "competitors" and found to exhibit superior performance. Examples of this behavior abound, particularly notable are [10, 12, 42, 66, 88, 105]. It is only recently that a more moderate eclectic view has emerged, [13, 32, 48, 78] for example, which acknowledges that all formalisms are useful for the solution of different problems. A general realization of the strength of this eclectic position has motivated research both into ways in which different formalisms may be used in combination to solve interesting practical problems, and into establishing the formal differences and similarities

between different systems.
End of quotation from [73].
manipulate certainty in a sound manner. Still, it is

Chapter 7

Fuzzy logic

Among the various uncertainty calculi, fuzzy logic is a rather popular approach of dealing with uncertainty. This popularity is partly due to the relative simplicity and plausibility with which standard ($\{0,1\}$-valued) logical operators can be extended to fuzzy ($[0,1]$-valued) logical operators. The concept of "$\times$" as a continuously-valued conjunction, for example, is straightforward and easy to understand. But this simplicity also opens the door to semantic misinterpretation and abuse, such as the erroneous combination of dependent evidence. This is also why fuzzy logic possesses a controversial reputation and is considered inferior to, for example, Bayesian belief networks. These issues will be briefly touched upon in Section 7.2 on page 145.

In spite of its controversial status, fuzzy logic has spectacular applications in industry and fundamental research. The control of subway in Sendai, Japan, for example, is one of the many success-stories of fuzzy logic. The (fuzzy) system controls acceleration, deceleration, and braking of the train. Since its introduction, it not only reduced energy consumption by 10%, but the passengers hardly notice now when the train is changing its velocity. The builders claim that, in the past neither conventional, nor human control could have achieved such performance.

Figure 7.1: Sendai subway in Japan.

There are more success stories. The automatic control of dam gates for hydroelectric-powerplants (Tokio Electric Pow), simplified control of robots (Hirota, Fuji Electric, Toshiba, Omron), camera aiming for the telecast of sporting events (Omron), substitution of an expert for the assessment of stock exchange activities (Yamaichi, Hitachi), preventing unwanted temperature fluctuations in air-conditioning systems (Mitsubishi, Sharp), efficient and stable control of car-engines (Nissan) optical inspection of airflow censors (Profactor), cruise-control for automobiles (Lexus) improved efficiency and optimized function of industrial control applications (Aptronix, Meiden) positioning of wafer-steppers in the production of semiconductors (Canon) optimized planning of bus time-tables (Toshiba, Nippon-System, Keihan-Express) archiving system for documents (Mitsubishi Elec.) prediction system for early recognition of earthquakes (Inst. of Seismology Bureau of Metrology, Japan) medicine technology: cancer diagnosis. And the list goes on ...

The list also shows that fuzzy logic particularly flourishes in Japan. In the US there is more skepticism toward fuzzy logic, to begin with the name, which became a controversy on the American market. Some experts argued that fuzzy logic is too "fuzzy" and too unpredictable. While the States stalled in the controversy, there were thousands of successful fuzzy logic implementations performed in Japan.

Below is only a modest introduction to fuzzy logic. It will only touch upon the basics of fuzzy logic such as t-norms and truth-functionality. After that, some problems with truth-functionality, and the possibly to use fuzzy logic to solve classical satisfiability problems are discussed.

7.1 Triangular norms

A triangular norm, or t-norm, is a conservative extension of the conjunction operator $\wedge : \{0,1\}^2 \to \{0,1\}$.

Definition 7.1 (t-norm) A function $\wedge : [0,1]^2 \to [0,1]$ is a t-norm if it is monotone, commutative, associative, and has 1 as a unit element:

i. $a \wedge c \leq b \wedge c$ whenever $a \leq b$
ii. $a \wedge b = b \wedge a$
iii. $a \wedge (b \wedge c) = (a \wedge b) \wedge c$
iv. $a \wedge 1 = a$

Well-known examples of t-norms are

$$\begin{aligned}
\wedge_{-1}(x,y) &=_{Def} a = 1 \,?\, b : b = 1 \,?\, a : 0,\\
\wedge_{\min}(x,y) &=_{Def} \min\{x,y\},\\
\wedge_{\mathrm{Luk}}(x,y) &=_{Def} \max\{0, x+y-1\}, \text{ and}\\
\wedge_{\mathrm{prod}}(x,y) &=_{Def} xy.
\end{aligned}$$

The graphs of these norms look like straight planes with a crack, except the graph of $\wedge_{\mathrm{prod}}$, which is curved and smooth (continuously differentiable).

Which one is the best? Well, that depends on your taste and needs. If you want $\wedge$ to be idempotent (i.e., $x \wedge x = x$) then $\wedge_{\min}$ is a good choice. Actually, there is no other choice if you want your norm to be idempotent. (This can be proven.) But why insisting on a property such as idempotence? In the next section, we will discuss the desirability of idempotence and other properties.

Completely analogously, there are conservative extensions of the classical disjunction $\vee$.

Definition 7.2 (t-conorm) A function $\vee : [0,1]^2 \to [0,1]$ is a *t-conorm* if it is monotone, commutative, associative, and has 0 as a unit element

Every t-norm defines a *dual t-conorm* by $\vee(x,y) =_{Def} 1 - (1-x) \wedge (1-y)$. The dual t-conorms of the above t-norms are

$$\begin{aligned}
\vee_{-1}(x,y) &=_{Def} a = 0 \,?\, b : b = 0 \,?\, a : 1\\
\vee_{\max}(x,y) &=_{Def} \max\{x,y\},\\
\vee_{\mathrm{Luk}}(x,y) &=_{Def} \min\{a+b, 1\}, \text{ and}\\
\vee_{\mathrm{prod}}(x,y) &=_{Def} x + y - xy.
\end{aligned}$$

How do the different norms compare? Here is a result.

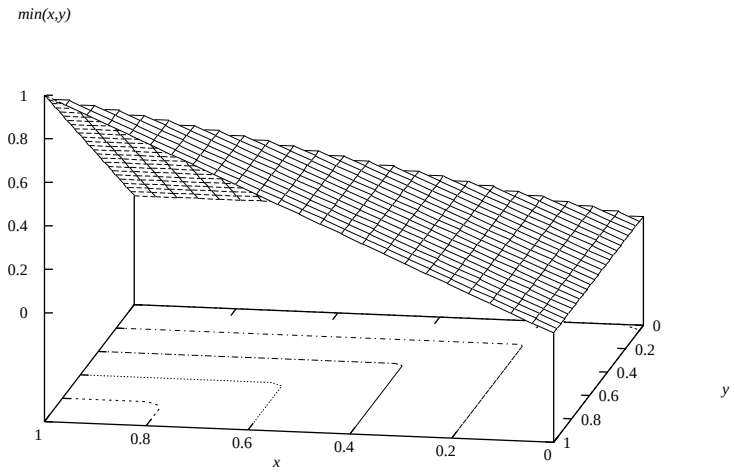


Figure 7.2: $\wedge_{\min}(x, y)$

Result 7.3 *$\wedge_{-1}$ is the smallest norm, $\wedge_{\min}$ is the largest norm, $\vee_{\max}$ is the smallest conorm, $\vee_{-1}$ is the largest conorm.*

Thus $\wedge_{-1}$ and $\wedge_{\min}$ are the most extreme norms and similarly for conorms. There are many

	Weber	Hamacher	Yager
$\wedge_{\min}$, $\vee_{\max}$:	-	-	$p \to \infty$
$\wedge_{\text{Luk}}$, $\vee_{\text{Luk}}$:	$\lambda = 0$	-	-
$\wedge_{\text{prod}}$, $\vee_{\text{prod}}$:	$\lambda \to \infty$	$\gamma = 1$	-
$\wedge_{-1}$, $\vee_{-1}$:	$\lambda \to -1$	$\gamma \to \infty$	$p \to 0$

Table 7.1: Families of t-(co)norms and their members.

more (co)norms:

Definition 7.4 (Families of t-(co)norms) For $\lambda > -1$ the *Weber family*:

$$\wedge_{\text{Web},\lambda}(x, y) =_{Def} \max\left\{\frac{x + y - 1 + \lambda xy}{1 + \lambda}, 0\right\}$$
$$\vee_{\text{Web},\lambda}(x, y) =_{Def} \min\left\{x + y - \frac{\lambda xy}{1 + \lambda}, 1\right\}$$

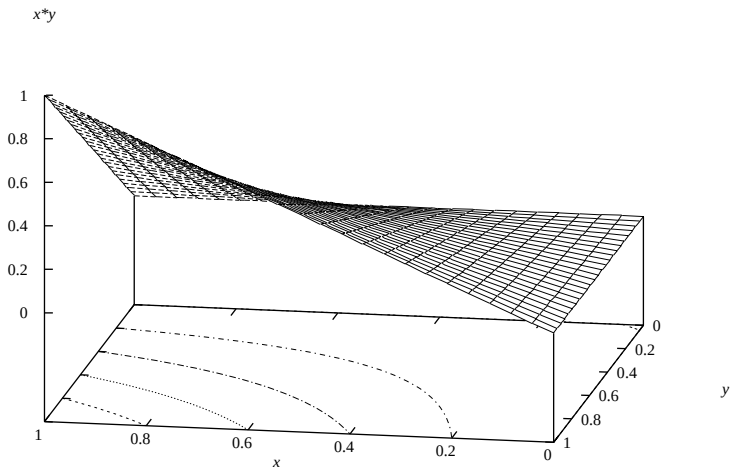


Figure 7.3: $\wedge_{\text{prod}}(x, y)$

For $\gamma > 0$ the *Hamacher family*:

$$\wedge_{\text{Ham},\gamma}(x, y) =_{Def} \frac{xy}{\gamma + (1-\gamma)(x + y - xy))}$$
$$\vee_{\text{Ham},\gamma}(x, y) =_{Def} \frac{x + y - xy - (1-\gamma)xy}{1 - (1-\gamma)xy}$$

For $p > 0$ the *Yager family*:

$$\wedge_{\text{Yag},p}(x, y) =_{Def} 1 - \min\{[(1-x)^p + (1-y)^p]^{1/p}, 1\},$$
$$\vee_{\text{Yag},p}(x, y) =_{Def} \min\{[x^p + y^p]^{1/p}, 1\}.$$

From Table 7.1 on the previous page and Result 7.3 on the preceding page it follows that the Yager family covers a wide range of norms.

Fuzzy negation

What about negation? We certainly want fuzzy negation to behave like classical negation on 0 and 1: $\neg 1 = 0$ and $\neg 0 = 1$. Further, it is reasonable to demand that $\neg : [0, 1] \to [0, 1]$ is monotonically non-increasing: a propositional and its negation cannot gain truth simultaneously.

The function $x \mapsto 1 - x$ seems to be a good choice, because it has all the properties we expect from a decent fuzzy negation function: among others, it is bijective, linear, involution, continuous, and continuously differentiable.

Other negations proposed in the literature are $x \mapsto x = 0\,?\,1 : 0$ and $x \mapsto x = 1\,?\,0 : 1$. FIXME.

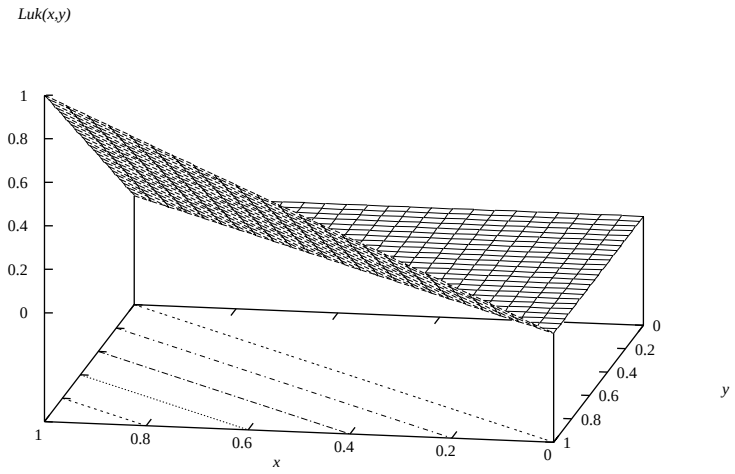


Figure 7.4: $\wedge_{\text{Luka}}(x, y)$

Fuzzy implication

A standard technique to derive an implication from a t-norm is to take the so-called *residuum* of it.

The idea behind a residuum is the following. In classical logic, we have for all ϕ, ψ and χ

$$w(\phi \wedge \psi) \leq w(\chi) \text{ iff } w(\phi) \leq w(\psi \supset \chi). \tag{7.1}$$

This is a relation between (classical) $\wedge$ and (classical) $\supset$. The residuum generalizes this relation to the fuzzy version of $\wedge$ and $\supset$.

Definition 7.5 (Residuum) A *residuum* of a t-norm $\wedge$ is a function $\supset: [0,1]^2 \to [0,1]$ such that for every x, y, z: $x \wedge y \leq z$ iff $x \leq y \supset z$.

Result 7.6 (The residuum is unique) *Every t-norm possesses exactly one residuum, which is*

$$y \supset z =_{Def} \text{ the smallest number } x \text{ such that } x \wedge y \leq z \tag{7.2}$$

Henceforth, we may speak of the implication that "belongs" to a particular t-norm. Cf. Table 7.2 on page 145.

Quantifiers

We confine ourselves by stating the fuzzy counterparts of the universal and existential quantifiers.

- $w(\forall xpx) =_{Def}$ the largest y such that, for all x, $y \leq w(P(x))$
- $w(\exists xpx) =_{Def}$ the smallest y such that, for all x, $w(P(x)) \leq y$

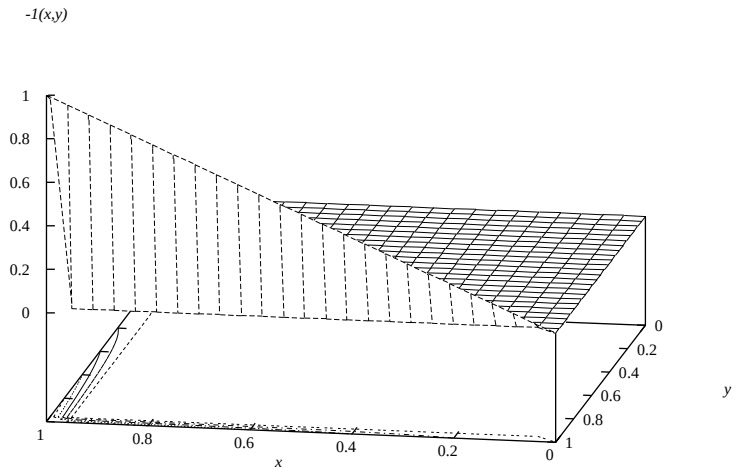


Figure 7.5: $\wedge_{-1}(x, y)$

PROBLEMS (Sec. 7.1)

1. Show that, for every t-norm $\wedge$, it holds that $p \wedge 0 = 0$.

2. Show that $\wedge_{\min}$, $\wedge_{\text{Luk}}$, $\wedge_{\text{prod}}$, and $\wedge_{-1}$ are t-norms.

3. (a) Give a meaning to the propositional variables P and Q such that $\Pr(P \wedge Q) < \wedge_{\text{prod}}(P, Q)$.

   (b) Give an interpretation to the propositional variables P and Q such that $\Pr(P \wedge Q) > \wedge_{\text{prod}}(P, Q)$.

   Explain your answers.

4. (a) Define the notion of duality of t-(co)norms.

   (b) Verify:

   i. $\wedge_{\min}$ is the dual of $\vee_{\max}$.

   ii. $\wedge_{\text{prod}}$ is the dual of $\vee_{\text{prod}}$.

5. Suppose the following axioms for fuzzy negation $\neg$:

   - Conservative extension of classical negation ($\neg 0 = 1$, $\neg 1 = 0$).
   - Strictly decreasing ($x < y \Rightarrow f(y) < f(x)$).
   - Involution ($\neg\neg x = x$).

   Prove:

   (a) Fuzzy-$\neg$ is onto (i.e., all numbers in $[0, 1]$ can be obtained by negation). (Solution.)

   (b) Fuzzy-$\neg$ is continuous.

   (c) (Difficult.) Fuzzy-$\neg$ is isomorphic to $x \mapsto 1 - x$. I.e., there is a bijection $f : [0, 1] \to [0, 1]$ such that $f(\neg x) = 1 - f(x)$. (Solution.)

Name	$w(\phi \supset \psi)$	*Residuum*
Łukasiewicz	$\min\{1 - w(\phi) + w(\psi), 1\}$	$\wedge_{\text{Luk}}$
Gödel	$w(\phi) < w(\psi)\,?\,1 : w(\psi)$	$\wedge_{\min}$
Goguen	$w(\phi) = 0\,?\,1 :\ \min\{w(\psi)/w(\phi), 1\}$	$\wedge_{\text{prod}}$
Kleene-Dienes	$\max\{1 - w(\phi), w(\psi)\}$	
Zadeh	$\max\{1 - w(\phi), \min\{w(\phi), w(\psi)\}\}$	
Reichenbach	$1 - w(\phi) + w(\phi)w(\psi)$	

Table 7.2: Examples of fuzzy implication.

7.2 Problems with truth-functional decomposition

A characteristic aspect of fuzzy logic is that it assumes that the truth-value of a logical formula can be determined by the truth-values of all immediate sub-formulas, regardless of the logical relation between these sub-formulas. For example, $w(\phi \wedge \psi)$ depends on $w(\phi)$ and $w(\psi)$ only, no matter what the relationship between ϕ and ψ is. In particular this means that fuzzy truth-values cannot be interpreted as plausibility measures. If we do, we get in trouble.

Example 7.7 Let $w(p) = 0.5$.

1. $w(p \wedge_{\min} \neg p) = \min\{0.5, 1 - 0.5\} = 0.5$. This is a paradox since we expect $w(p \wedge \neg p)$ to be 0.
2. Then $w(p \wedge_{\text{prod}} p) = 0.5 \times 0.5 = 0.25$. Thus, according to $\wedge_{\text{prod}}$, $p \wedge p$ appears twice less credible than p. This is a paradox since we expect $w(p \wedge p) = w(p)$. Moreover, $w(p \wedge_{\text{prod}} \ldots \wedge_{\text{prod}} p)$ tends to zero for large conjunctions. The situation is even worse for $\wedge_{-1}$, since $w(p \wedge_{-1} p)$ does not exceed zero whenever p is not absolutely certain (i.e., whenever $p < 1$).

Probability theory is a fine example of a calculus that is made to deal with the dependence or independence among propositions. For instance, the probability of $\phi \vee \psi$ cannot be computed, unless we know something about the relationship between ϕ and ψ:

$$\Pr(\phi \vee \psi) = \Pr(\phi) + \Pr(\psi) - \Pr(\phi \wedge \psi)$$

If we know that ϕ and ψ are independent, for example, then probability theory allows us to write $\Pr(\phi \wedge \psi) = \Pr(\phi)\Pr(\psi)$, so that $\Pr(\phi \vee \psi)$ may now be computed from $\Pr(\phi)$ and $\Pr(\psi)$ alone. Also if $\psi \equiv \neg\phi$, we know that $\Pr(\phi \wedge \psi) = 0$, so that also in this case $\Pr(\phi \vee \psi)$ may be computed from $\Pr(\phi)$ and $\Pr(\psi)$ alone. Finally, if $\psi \equiv \phi$, we know that $\Pr(\phi \wedge \psi) = \Pr(\phi)$, so that again $\Pr(\phi \vee \psi)$ may be computed from the individual probabilities.

Thus, fuzzy logic is not suited as a means for reasoning about the probability, plausibility, or credibility of propositions.

7.3 Theoretical applications of fuzzy logic

If fuzzy logic cannot be used to reason about belief, then what good is it? To answer this question we have to go back to the basic ideas of fuzzy logic.

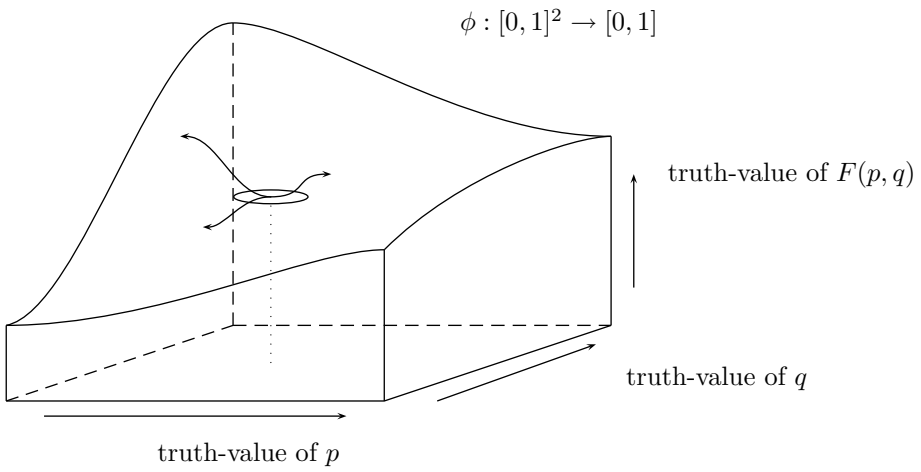


Figure 7.6: Hill-climbing with fuzzy satisfiability

An essential aspect of fuzzy logic is that it extends the interpretation of classic connectives (such as $\wedge$, $\vee$ and $\neg$) from a discrete to a real-valued domain. With this move, we leave the world of discrete mathematics and enter the world of continuous mathematics, i.e. the world of calculus with real-valued and (continuously) differentiable functions. In particular, this means that we have all the tools and techniques of at our disposal that belong to arsenal of real-valued analysis, such as the ability to compute partial and directional derivatives, linear approximations, and gradient and ascent. Such tools and techniques may come in helpful when we try to solve problems in the discrete domain.

Satisfiability

One application of fuzzy logic is to attack the discrete $\{0,1\}$-satisfiability problem SAT with fuzzy means.

Example 7.8 (Fuzzy GSAT) Suppose we would like to know whether

$$\neg((q \vee p) \wedge (\neg p \vee q) \wedge (p \supset q)) \tag{7.3}$$

is satisfiable. We rewrite (7.3) into a function

$$F(p,q) = 1 - (((q + p - qp) * ((1 - p + q) - ((1 - p)q)))((1 - p + q) - ((1 - p)q))) \tag{7.4}$$

by recursively substituting each conjunction with a multiplication, and each disjunction with an expression of the form $x + y - xy$. Remember that in GSAT the score function is the number of clauses that is satisfied by a particular valuation w. Here, the score function is F.

Suppose we start out in $(p,q) = (0,1)$. In this point, $F(0,1)$ is 0. If p is flipped, $F(1,1)$ stays 0, and if q is flipped, $F(0,0)$ becomes 1. Therefore, we flip q and our algorithm halts with solution $(p,q) = (0,0)$.

If (7.3) contains more than two variables, we may generalize the search process to more dimensions.

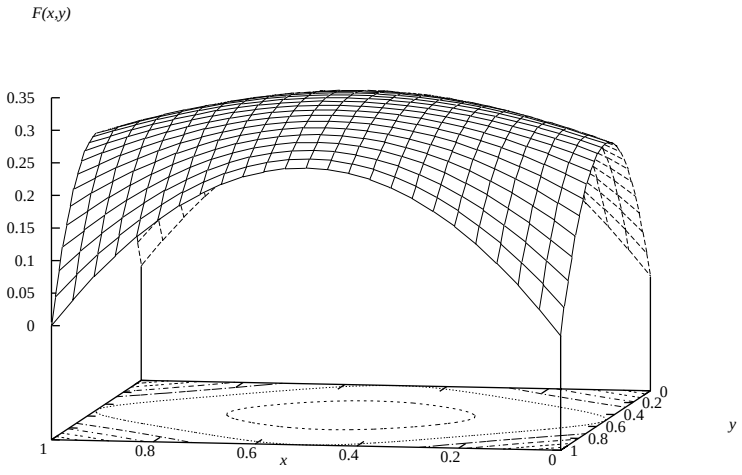


Figure 7.7: $F(x, y)$

Fuzzy satisfiability

Instead of attacking the discrete satisfiability problem SAT, we may attack the fuzzy satisfiability problem (fuzzy SAT). Fuzzy SAT is like SAT, except that all variables may assume real, rather than discrete, values.

Example 7.9 (Fuzzy SAT) Consider

$$\phi = (p \vee q) \wedge (\neg p \vee q) \wedge (p \vee \neg q) \wedge (\neg p \vee \neg q). \tag{7.5}$$

This propositional is not satisfiable with discrete values. (Verify if in doubt.) But we may investigate which continuous values of (p, q) ϕ attains it highest truth-value.

Again, we rewrite (7.5) into an arithmetic expression

$$F(p,q) = ((p+q-pq)(p+1-q-p(1-q))(1-p+q-(1-p)q))(2-p-q-(1-p)(1-q))$$

Cf. Figure 7.7. We then may maximize F with optimization techniques known from calculus and neural computing, such as gradient ascent (i.e., hill climbing), simulated annealing or tabu search. Cf. Figure 7.6 on the preceding page. The present case is simple enough to do by hand, however, and doing so we find out that F attains a maximum of $0.75^4 = 0.316$ at $(p, q) = (0.5, 0.5)$.

That's it for now. Of course there is much more to say on fuzzy logic. Examples are fuzzy set theory, multiple-valued logic, fuzzy statistics, fuzzy machine learning, soft computing, fuzzy control, neuro-fuzzy computing, and possibilistic logics. Since this is clearly beyond the scope of this chapter I refer the reader to more specialized literature.

Chapter 8

Argumentation

Not all reasoning proceeds deductively. In fact, most reasoning consists of non-deductive patterns of reasoning, better known as *arguments*. Arguments look like proofs: tree-like formal structures that start with premises and end with a conclusion. Unlike proofs, arguments do not stand once and for all, but may be defeated by stronger counterarguments. We shall illustrate this form of defeasible reasoning with the following example, taken from [92].

An example

Two persons, A and B, disagree on whether it is morally acceptable for a newspaper to publish a certain piece of information concerning a politician's private life. Let us assume that the two parties have reached agreement on the following points.

(1) The piece of information I concerns the health of person P;
(2) P does not agree with publication of I;
(3) Information concerning a person's health is information concerning that person's private life

A now states the moral principle that

(4) Information concerning a person's private life may not be published if that person does not agree with publication.

and A says "So the newspapers may not publish I" (Fig. 8.1, page 150). Although B accepts principle (4) and is therefore now committed to (1-4), B still refuses to accept the conclusion that the newspapers may not publish I. B motivates his refusal by replying that:

(5) P is a cabinet minister
(6) I is about a disease that might affect P's political functioning
(7) Information about things that might affect a cabinet minister's political functioning has public significance

Furthermore, B maintains that there is also the moral principle that

(8) Newspapers may publish any information that has public significance

B concludes by saying that therefore the newspapers may write about P's disease (Fig. 8.2, page 151). A agrees with (5–7) and even accepts (8) as a moral principle, but A does not give up his initial claim. (It is assumed that A and B are both male.) Instead he tries to defend it by arguing that he has the stronger argument: he does so by arguing that in this case

(9) The likelihood that the disease mentioned in I affects P's functioning is small.

(10) If the likelihood that the disease mentioned in I affects P's functioning is small, then principle (4) has priority over principle (8).

Thus it can be derived that the principle used in A's first argument is stronger than the principle used by B (Fig. 8.3, page 151), which makes A's first argument stronger than B's, so that it follows after all that the newspapers should be silent about P's disease.

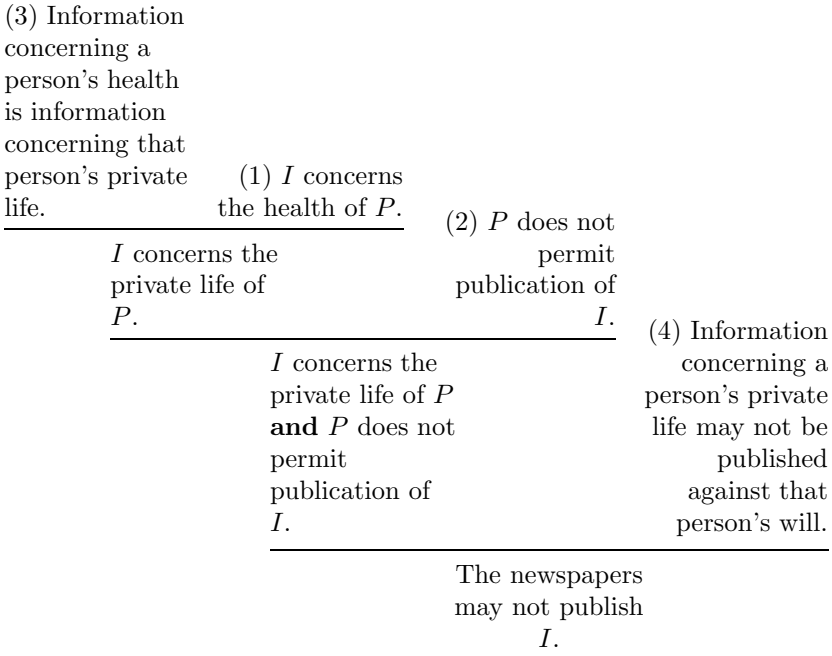


Figure 8.1: A's argument.

Let us examine the various stages of this dispute in some detail. Intuitively, it seems obvious that the accepted basis for discussion after A has stated (4) and B has accepted it, viz. (1, 2, 3, 4), warrants the conclusion that the piece of information I may not be published. However, after B's counterargument and A's acceptance of its premises (5-8) things have changed. At this stage the joint basis for discussion is (1-8), which gives rise to two conflicting arguments. Moreover, (1-8) does not yield reasons to prefer one argument over the other: so at this point A's conclusion has ceased to be warranted. But then A's second argument, which states a preference between the two conflicting moral principles, tips the balance in favor of his first argument: so after the basis for discussion has been extended to (1-10), we must again accept A's moral claim as warranted.

This chapter is about logical systems that formalize this kind of reasoning. We shall call them "logics for defeasible argumentation," "defeasible logics," or "argument systems". An important difference between defeasible logics and deductive logics, is that deductive logics work with proofs while defeasible logics work with arguments. A proof is a special type of argument. It is, or should be, a watertight and rigorous mathematical demonstration that, once found, establishes its conclusion once and for all. An argument, on the other hand, can be defeated by stronger counterarguments, provided such counterarguments exist, are put forward, and are not defeated themselves. An important property of deductive reasoning is that it is *monotonic*:

if $p \vdash r$, then $p, q \vdash r$.

Thus, new information cannot invalidate a proven fact. Non-deductive reasoning, however, is *nonmonotonic*: if there is an argument A on the basis of p for r then it is reasonable to conclude

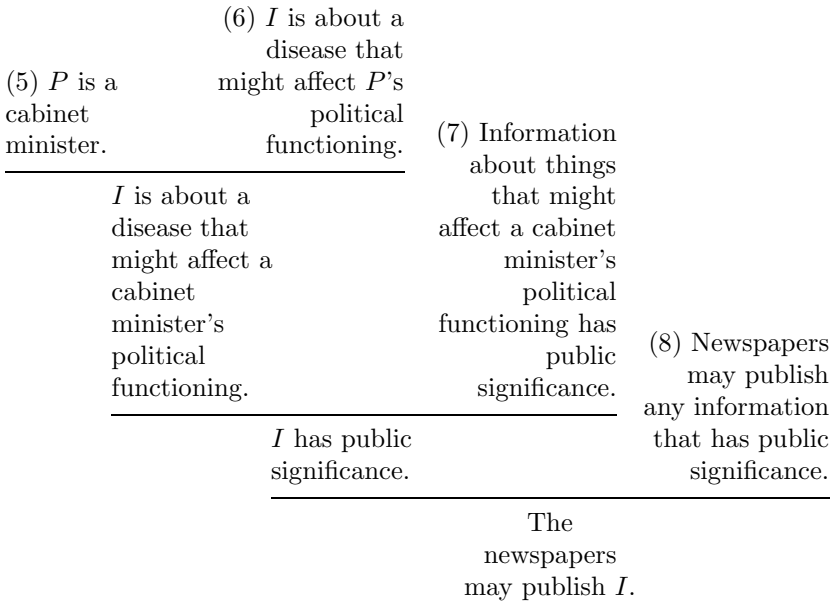


Figure 8.2: B's argument.

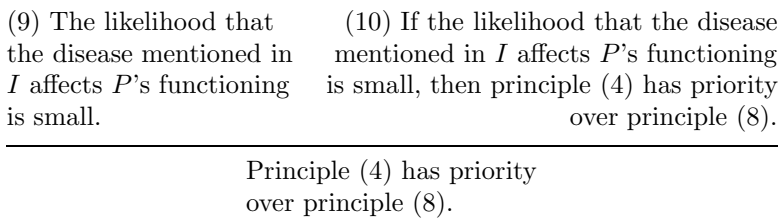


Figure 8.3: A's priority argument.

$p \mathrel{|\!\sim} r$, especially if there are no counterarguments, or the only counterarguments that do exist are defeated by other arguments. However, if B is a strong argument against (a sub-argument of) A on the basis of q, and q becomes true, then $p, q \not\mathrel{|\!\sim} r$. Thus,

if $p \mathrel{|\!\sim} r$ then it may be that $p, q \not\mathrel{|\!\sim} r$

(Fig. 8.4 on the following page).

Another important difference between defeasible logics and deductive logics, is that in deductive logics, the mere *existence* of a proof is sufficient to conclude that a theorem holds, and to conclude that it cannot be "disproven" by other proofs. Thus, in deductive logics, it is important *that* we arrive at a conclusion, but it is unimportant *how* we arrive at that particular conclusion. More precisely, the proof becomes irrelevant, once it has been given. (Like Wittgensteins's "ladder" in his tractatus.[1]) Defeasible reasoning, on the other hand, works differently. In defeasible reasoning, arguments form an essential part of the support of a conclusion, because the intermediate steps of reasoning of such an argument are uncertain and must remain available for further inspection by parties who want to dispute such a conclusion.

[1] Wittgenstein ended his tractatus by saying that his propositions are like a ladder which can be kicked away when it has gotten us where we want to be.

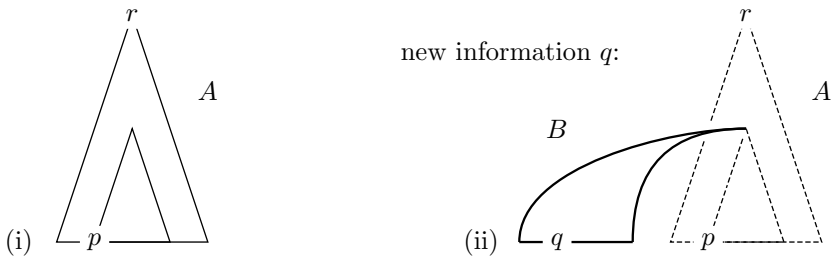


Figure 8.4: Nonmonotonicity: $p \mathrel{|\!\sim} r$ (left) and $p, q \not\mathrel{|\!\sim} r$ (right)

8.1 A new connective

An essential insight of defeasible reasoning is that practical argumentation more often than not contains non-deductive steps of reasoning.

As an example, suppose that we attend a conference, listen to a panel, and wonder whether a particular panellist is reasonable. If we do not know him (or her),[2] we would probably assume, by default, that he is a reasonable person. Thus, panellists (p) are reasonable (r), unless indicated otherwise. This way of reasoning is known as *default reasoning*. In propositional logic, this would be written as:

$$p \supset r \tag{8.1}$$

(panellists are reasonable persons). But what if your friend, who is also at the conference, and who is very reliable, prompts you with the information that this particular panellist is a quarrelsome person (q)? Do we, in that case, still suppose that that person is reasonable? Probably not. This would lead us to

$$\neg(p \wedge q \supset r) \tag{8.2}$$

(it is not the case that quarrelsome panellists are reasonable persons) and probably even to

$$p \wedge q \supset \neg r \tag{8.3}$$

(quarrelsome panellists are unreasonable persons). We are now confronted with a knowledge-representation problem: (8.1) and (8.2) are logically inconsistent, while in reality it does make sense to concurrently maintain that panellists tend to be reasonable and that quarrelsome panellists tend to be unreasonable. Also the combination (8.1) and (8.3) is logically inconsistent. Thus, "$\supset$" is a logical connective that cannot be used to represent defaults, rules of thumb and similar "speculative" rules, with which it is possible to "jump to conclusions".

One solution would be to have a new binary connective "$\rightsquigarrow$," on the object-level of the language, named "defeasible implication". This connective should be akin to material implication but, at the same time, permit situations like

$$(p \rightsquigarrow r) \wedge \neg(p \wedge q \rightsquigarrow r) \qquad \text{and} \qquad p \wedge (p \rightsquigarrow q) \wedge \neg q.$$

In other words, it should be possible that

$$p \rightsquigarrow r \not\mathrel{|\!\sim} p \wedge q \rightsquigarrow r \qquad \text{and} \qquad p, p \rightsquigarrow q \not\mathrel{|\!\sim} q$$

Due to work from Post and Yablonsky on the functional completeness of logical operator, we know that "$\rightsquigarrow$" cannot be defined by means of standard logical connectives, such as "$\neg$," "$\wedge$" and "$\vee$" [88, 131]. Thus, the new connective must have a semantics that differs from the standard two-valued truth-functional semantics.

Below, we try to define such a connective.

[2] I use the generic masculine form, intending no bias.

Epsilon-semantics

It is possible to introduce a new binary connective "$\rightsquigarrow$" as described above, and to integrate its semantics with familiar logical connectives such as "$\neg$," "$\wedge$" and "$\vee$" by means of the so-called *epsilon semantics*. The basic idea of ϵ-semantics can be traced back to work of Adams [1, 3, 2], Geffner [78], and Pearl [79, 75].

Epsilon semantics (ϵ-semantics, for short) is an interpretation of defeasible conditionals $p \rightsquigarrow q$, based on the idea that "most situations in which p holds, are situations in which q holds". This idea can be put to work by interpreting logical connectives in a set-theoretical manner. Accordingly, the connectives "$\neg$," "$\wedge$" and "$\vee$" correspond to the set-theoretical operations "$^-$" (complement), "$\cap$" (intersection) and "$\cup$" (union), respectively, while the connective "$\supset$"corresponds to the set-theoretical relation "$\subseteq$".

Example 8.1 - The sequent $p \wedge q \vdash q$ is verified by the expression $P \cap Q \subseteq Q$ or, alternatively, by the inference

$$\frac{x \in P \cap Q}{x \in P}$$

- The sequent $(p \wedge q) \supset r, p \supset (r \vee q) \vdash p \supset r$ can be verified by examining the inference

$$\frac{(P \cap Q) \subseteq R, \quad P \subseteq (R \cup Q)}{P \subseteq R}$$

- The sequent $(p \wedge q) \supset r, p \supset (r \vee q), p \vdash r$ is verified by

$$\frac{(P \cap Q) \subseteq R, \quad P \subseteq (R \cup Q), \quad x \in P}{x \in R}$$

- The sequent $p \vdash q \supset p$ can be verified by writing $q \supset p$ as $\neg q \vee p$ and verifying that $P \subseteq \bar{Q} \cup P$.

Note that verifying formulas with nested occurrences of "$\supset$" requires that we rewrite the formula into an equivalent formula such that nested occurrences of "$\supset$" do not occur, since the subset-predicate "$\subseteq$"cannot be nested. $\square$

The set-theoretical semantics enables us to give a meaning to formulas with unnested occurrences of "$\rightsquigarrow$," based on the idea that "most situations in which p holds, are situations in which q holds". Accordingly, the truth of $p \rightsquigarrow q$ is related to the fraction of situations that verify p and q.

$$\frac{|P \cap Q|}{|P|} \rightsquigarrow 1 \tag{8.4}$$

This time, the arrow "$\rightsquigarrow$" is not a logical connective, but a mathematical operator indicating that a certain fraction approaches one. *How* the fraction approaches 1 is left unspecified. Equation (8.4) merely says that P is non-empty and that P and Q move in such a way that the proportion of P's that are Q, tends to 1.

If we further abstract from counting situations in which a particular proposition holds, we may write

$$Pr(Q|P) = \frac{Pr(PQ)}{Pr(P)} \rightsquigarrow 1 \tag{8.5}$$

where "Pr" is a probability measure on propositions.

This apparently vague criterion leads to a surprisingly clear semantics.

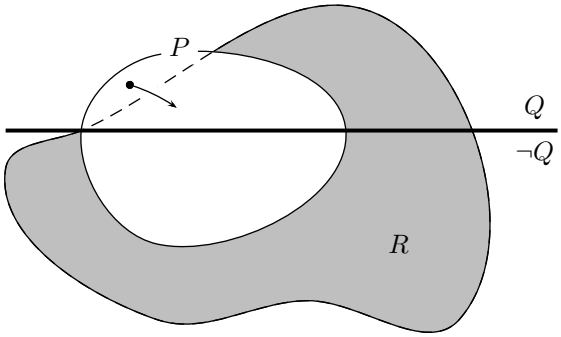


Figure 8.5: Sketch to suggest the correctness of the inference displayed at Eq. 8.6

Example 8.2 The validity of the sequent $p \supset q \vdash p \rightsquigarrow q$ can be verified by examining the validity of

$$\frac{P \subseteq Q}{Pr(Q|P) \rightsquigarrow 1}$$

In this case, there is no limit-process going on here, and the inference is valid, because $Pr(Q|P) = Pr(PQ)/Pr(P) = Pr(P)/Pr(P) = 1$ for non-empty $P \subseteq Q$. □

Example 8.3 The validity of the sequent $p \wedge q \rightsquigarrow r, p \wedge \neg q \supset r \vdash p \rightsquigarrow r$ can be verified by examining the inference

$$\frac{Pr(R|PQ) \rightsquigarrow 1, \quad P \backslash Q \subseteq R}{Pr(R|P) \rightsquigarrow 1} \tag{8.6}$$

The correctness of this inference is suggested in Figure 8.5. To prove (8.6), we prove

$$\frac{Pr(R|PQ) = 1 - \epsilon, \quad P \backslash Q \subseteq R}{Pr(R|P) \geq 1 - \epsilon}$$

for every $0 < \epsilon < 1$.

$$\begin{aligned}
Pr(R|P) &= Pr(R|PQ)Pr(Q|P) + Pr(R|P\bar{Q})Pr(\bar{Q}|P) && \text{conditionalizing } Pr(R|P) \text{ on } Q \\
&= (1-\epsilon)Pr(Q|P) + Pr(R|P\bar{Q})Pr(\bar{Q}|P) \\
&= (1-\epsilon)Pr(Q|P) + 1 \cdot Pr(\bar{Q}|P) && \text{since } P\backslash Q \subseteq R \text{ implies } Pr(R|P\bar{Q}) = 1 \\
&= (1-\epsilon)Pr(Q|P) + 1 \cdot (1 - Pr(Q|P)) \\
&= 1 - \epsilon Pr(Q|P) \\
&\geq 1 - \epsilon \cdot 1
\end{aligned}$$

Now Eq. (8.6) follows if $\epsilon \rightsquigarrow 0$. □

The inferences shown in Table 8.1 on the next page are semantically valid. Adams [1, 3, 2] showed that subsumption, cumulativity, contraction, and disjunction can function as basic rules of inference from which other rules can be derived in a pure syntactical manner. Further, it is shown that there are algorithms, quadratic in the number of "$\rightsquigarrow$"'s, to check whether a set of formulas is ϵ-satisfiable [79, 76].

Not all inferences are semantically valid. The following are semantically *in*valid:

- *Monotonocity:* $p \rightsquigarrow r \nvdash p \wedge q \rightsquigarrow r$

$\dfrac{p \supset q}{p \sim\!\!> q}$ subsumption	$\dfrac{p \vdash q}{p \sim\!\!> q}$ subsumption (2)
$\dfrac{p \sim\!\!> q \quad (p \wedge q) \sim\!\!> r}{p \sim\!\!> r}$ contraction	$\dfrac{p \sim\!\!> q \quad p \sim\!\!> r}{(p \wedge q) \sim\!\!> r}$ cumulativity
$\dfrac{p \sim\!\!> r \quad q \sim\!\!> r}{(p \vee q) \sim\!\!> r}$ disjunction	$\dfrac{p \sim\!\!> q \quad p \wedge q \supset r}{p \sim\!\!> r}$ closure
$\dfrac{p \sim\!\!> r \quad p \equiv q}{q \sim\!\!> r}$ equivalence	$\dfrac{p \sim\!\!> r \quad p \wedge q \sim\!\!> \neg r}{p \sim\!\!> \neg q}$ exception
$\dfrac{p \sim\!\!> q \quad p \sim\!\!> r}{p \sim\!\!> (q \wedge r)}$ right conjunction	$\dfrac{p \wedge c \sim\!\!> r \quad p \wedge \neg c \sim\!\!> r}{p \sim\!\!> r}$ cases

Table 8.1: Inferences that are valid in the ϵ-semantics.

- *Transitivity:* $p \sim\!\!> q, q \sim\!\!> r \nvdash p \sim\!\!> r$
- *Left conjunction:* $p \sim\!\!> r, q \sim\!\!> r \nvdash (p \wedge q) \sim\!\!> r$. Example: if you marry Pat (p) you will be happy (r); if you marry Quentin (q), you will be happy as well, but if you marry both ($p \wedge q$) your happiness is no longer guaranteed.
- *Contraposition:* $p \sim\!\!> q \nvdash \neg q \sim\!\!> \neg p$. To see this, consider the following example. It is true that men (p) are usually beardless (q), i.e. wear no beards. However, this not mean that people (we're reasoning in the universe of human beings) that *do* wear a beard ($\neg q$) are usually women ($\neg p$).
- *Defeasible modus ponens:* $p, p \sim\!\!> q \nvdash q$ ($\widetilde{\text{MP}}$). Example: if birds (p) tend to fly (q), then seeing one particular bird does not necessarily mean that it flies.

Despite its invalidity, most commonsense arguments seem make use of $\widetilde{\text{MP}}$, because it enables conclusions that would otherwise be unreachable.

The fact that conclusions obtained with $\widetilde{\text{MP}}$ might be false, is often considered less important than the fact that conclusions can be obtained at all. Because $\widetilde{\text{MP}}$ is frequently used, we use the following notation:

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ p & p \sim\!\!> q\end{array}}{\quad q \quad}\ \widetilde{\text{MP}}$$

The double ruler is meant to remind you of the fact that $\widetilde{\text{MP}}$ is ϵ-invalid.

Defeasible modus ponens, i.e., $\widetilde{\text{MP}}$, may be considered as the defeasible counterpart of $\supset$-elimination. In standard propositional logic there is but one type of inference, namely *deduction*, denoted by "$\vdash$". A deduction can be compressed by the introduction of a material

implication "$\supset$":

$$\dfrac{\dfrac{[p]}{\vdots}}{\dfrac{q}{p \supset q}}\ \supset\text{-introduction}$$

The "$\supset$," then compresses, or summarizes, the derivation from p to q.

Conversely, a material implication can be used, or decompressed, by eliminating it via modus ponens, or $\supset$-elimination:

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ p & p \supset q\end{array}}{q}\ \supset\text{-elimination, or MP}$$

Thus, "$\supset$" can be viewed as the object-level representation of the meta-logical concept "$\vdash$" of logical deduction. In any case, the symbols are clearly related. Thus, $\widetilde{\text{MP}}$ may be considered as $\rightsquigarrow$-elimination. The counterpart of $\rightsquigarrow$-elimination, viz. $\rightsquigarrow$-introduction, is not considered here, because it introduces numerous other questions.

PROBLEMS (Sec. 8.1)

1. Above, the contraposition rule is falsified with the help of an example where men usually do not wear beards. Draw a Venn-diagram that corresponds to this counterexample. (Solution.)

2. Disprove the left-conjunction rule. Hint: it is impossible to give a counterexample with the help of a static Venn-diagram. (Solution.)

3. (a) Prove the cumulativity rule (Table 8.1 on the preceding page). (Solution.)
   (b) Prove the contraction rule.
   (c) Prove the disjunction rule. (Solution.)
   (d) Prove the rule that says that we may distinguish cases. (Solution.)

8.2 Three types of inference

It is possible to split argumentation into three types of inference, corresponding with the interpretation of "$\supset$" and two different interpretations of "$\rightsquigarrow$".

1. *Deductive arguments.* All inferences in which the conclusion necessarily follows from the premises. For example, $p, p \supset q \vdash q$; $p \wedge q \vdash q$, et cetera, but also ϵ-valid inferences such as $p \rightsquigarrow q, q \supset r \vdash p \rightsquigarrow r$ (cf. Table 8.1 on the previous page.)

2. *Statistic arguments.* P statistically entails ψ if there exists a ϕ such that $P \vdash \phi$ and $P \vdash \phi \rightsquigarrow \psi$, while there exists no ϕ' such that

   (a) $P \vdash \phi'$ and $P \vdash \phi' \rightsquigarrow \psi'$;
   (b) ϕ' is strictly more specific than ϕ relative to P (i.e., $P \vdash \phi' \supset \phi$ while $P \nvdash \phi \supset \phi'$);
   (c) ψ and ψ' contradict each other relative to P (i.e., $P \vdash \neg(\psi \wedge \psi')$.

To see why there may be no such ϕ', consider $P = \{b \rightsquigarrow f\} \cup \{b \wedge c\}$, where b means "bird," f means "flies," and c means "clipped wings". Surely, we do not want to conclude that birds with clipped wings fly simply because they are birds. Thus $\phi \rightsquigarrow \psi$ may be applied only if ϕ is the most specific information that is available.

3. *Generic arguments.* Statistical reasoning may be appropriate in some circumstances, but this should not be our only model for defeasible reasoning. Sometimes, we wish to go beyond what is statistically sound. Therefore we distinguish a second type of reasoning, namely, the type of reasoning in which inferences are made, not on the basis of statistical considerations or majority arguments, but by default, in the absence of better information. In such cases, sentences of the form $\phi \rightsquigarrow \psi$ are read as "ϕ is a good reason to infer ψ". What that good reason might be, then, is left undetermined.

Reading (3) emphasizes the non-statistical use of defeasible implications; (3) subsumes (2): a reason based on statistics, or frequencies, is a good reason; but a good reason is not necessarily based on statistics. The difference is that (2) has a statistical connotation, while (3) has a *prototypical* connotation. For example, a typical mosquito tends to pass malaria although most mosquito's in fact do not, statistically speaking.

We can also look at the matter from the perspective of satisfiability. In that case, generic consistency implies statistical consistency which, on its turn, implies deductive consistency. Thus, it is possible to have an ϵ-satisfiable set of formulas that nevertheless enables conflicting arguments. Consider, for instance, the set P (Equation 8.7 on the following page). This set is ϵ-satisfiable, while it is definitely not consistent in a generic reading of $\rightsquigarrow$, witness the arguments displayed in Eq. 8.8 on the next page.

These three types of inference should be understood as disjoint—not inclusive. Thus, if we say "statistic argument," we suppose that the argument is non-deductive; if we say "generic argument," we suppose that the argument lacks a statistic underpinning.

8.3 Principles of defeat

Depending on the type of inferences that were used, arguments may possess more or less conclusive force. As long as competing arguments clearly differ in conclusive force, it is also clear which of them should defeat the others.

1. *Deductive arguments defeat non-deductive arguments*. This is obvious.

2. *Statistic arguments defeat generic arguments*. The underlying idea is that statistical evidence carries more weight than just a line of reasoning.

To be sure, it may happen that a generic inductive argument is right where, at the same time, a statistically based inductive argument is wrong. But this does not run counter to the principles on which statistically based arguments are preferred.

Conflicting deductive arguments refer to logically incompatible premises. These include statistically unsatisfiable premises like $P = \{a \rightsquigarrow b, a \rightsquigarrow \neg b\}$ or
$Q = \{a \rightsquigarrow \neg c, b \rightsquigarrow \neg c, (a \vee b) \rightsquigarrow c\}$. In that case some of these premises should be given up. This is the domain of *belief revision*, which is not at issue here.
Conflicting statistically based inductive arguments are the result of conflicting statistical information. (Not of statistically unsatisfiable premises). Example: $P = \{a \rightsquigarrow c, (a \wedge b) \rightsquigarrow \neg c\}$. Here, it is reasonable to let the most specific reference class $a \wedge b$ settle the matter.

Here are some examples of arguments that contain instantiations of defeasible modus ponens:

Example 8.4 (Whether Fred flies) Fred is a penguin. Penguins are birds, and birds tend to fly, while penguins do not fly. Does Fred Fly? If

Variable	*Meaning*	and	$P = \{p \supset b, p \supset \neg f, b \rightsquigarrow f\} \cup \{p\}$,	(8.7)
p	'Fred is a penguin,'			
b	'Fred is a bird,'			
f	'Fred flies'.			

then the union tries to express that P can be thought of as available information, divided into background knowledge (knowledge that is practically always true) $\{p \supset b, p \supset \neg f, b \rightsquigarrow f\}$ and contingent knowledge (knowledge that differs from case to case) $\{p\}$. Often (but not always) defeasible implications is background knowledge. The idea is that observations will trigger elements of the background knowledge so as to obtain new and/or interesting conclusions. From a logical point of view, however, all members of P are of equal importance.

Arguments for $\neg f$ and f would be

$$\dfrac{p \quad p \supset \neg f}{\neg f}\ \mathrm{MP} \qquad \text{and} \qquad \dfrac{\dfrac{p \quad p \supset b}{b}\ \mathrm{MP} \qquad b \rightsquigarrow f}{f}\ \widetilde{\mathrm{MP}} \tag{8.8}$$

respectively. Normally, the argument against flying is stronger than the argument against flying, because the first argument is deductive, while the second is not. □

Without the possibility of making generic inferences, arguments would already get stuck on an elementary level, since many inferences lack statistical support. A generic inference tries to overcome this by surpassing statistical considerations. A logically fundamental option like *chaining* is achievable only by means of generic argumentation principles.

Example 8.5 (Chaining) If $P = \{p_1, p_1 \rightsquigarrow p_2, \ldots, p_{n-1} \rightsquigarrow p_n\}$, then the argument

$$\dfrac{\dfrac{\dfrac{p_1 \quad p_1 \rightsquigarrow p_2}{p_2} \qquad p_2 \rightsquigarrow p_3}{p_3}}{\vdots}$$
$$\dfrac{\dfrac{\dfrac{p_{n-3} \qquad p_{n-3} \rightsquigarrow p_{n-2}}{p_{n-2}} \qquad p_{n-2} \rightsquigarrow p_{n-1}}{p_{n-1}} \qquad p_{n-1} \rightsquigarrow p_n}{p_n}$$

will typically be judged as valid in a generic reading of $\rightsquigarrow$. Statistically, the rule is invalid as soon as $n \geq 3$. The information that almost all p_1-occasions are p_2-occasions and almost all p_2-occasions are p_3-occasions simply does not imply that almost all p_1-occasions are p_3-occasions. A corresponding counterexample may be produced in the ϵ-calculus. □

Inferences that are (2)-valid are (3)-valid as well. This is so, because a statistical argument is, *a fortiori*, an argument. In the same manner, it follows that (3)-satisfiability implies (2)-satisfiability: if a contradiction cannot be derived with a generic argument, it certainly cannot be derived with a statistical argument.

Example 8.6 (Skew diamond) [46]. Native speakers of Pennsylvanian Dutch tend to be born in Pennsylvania, and people born in Pennsylvania are born in the USA. Native speakers of

Pennsylvanian Dutch are native speakers of German, and native speakers of German tend *not* to be born in the USA. Herman speaks Pennsylvanian Dutch. Is Herman born in the USA?

$$P = \{p \supset q, q \rightsquigarrow \neg s, p \rightsquigarrow r, r \supset s\} \cup \{p\},$$

where

Variable	*Meaning*
p	'native speaker of Pennsylvanian Dutch,'
q	'native speaker of German,'
r	'born in Pennsylvania,' and
s	'born in the USA'.

Does P entail s?

There are no deductive arguments for or against s (i.e. for $\neg s$). However, there are two non-deductive arguments, viz.

$$\dfrac{p \quad \dfrac{p \rightsquigarrow r \quad r \supset s}{p \rightsquigarrow s}}{s} \qquad \text{and} \qquad \dfrac{p \quad p \supset q}{\dfrac{q \qquad q \rightsquigarrow \neg s}{\neg s}}$$

The first argument concludes s on a statistical basis. The second argument lacks such a statistical basis, because $P \vdash p$ and $P \vdash p \supset q$. It might well be the case that $p \supset s$, so that $p \supset q, q \rightsquigarrow \neg s \nvdash p \rightsquigarrow \neg s$. □

Example 8.7 (Weak diamond) Aka the Nixon diamond [33]. Quakers tend to be pacifists, while republicans tend not to be pacifists. Nixon is both a quaker and a republican.[3] Is Nixon a pacifist?

$$P = \{q \rightsquigarrow p, r \rightsquigarrow \neg p\} \cup \{q \wedge r\}.$$

Does P entail p? All arguments for and against p are generic. (Verify!) Consequently, there is no statistical correlation between P and p. Put differently, both $P \cup \{p\}$ and $P \cup \{\neg p\}$ are satisfiable within the ϵ-semantics. □

Example 8.8 (Disjunctive antecedents) Consider

$$P = \{p \rightsquigarrow r, q \rightsquigarrow r\} \cup \{p \vee q\}$$

The proposition r cannot be inferred on either $p \rightsquigarrow r$ or $q \rightsquigarrow r$ alone. These rules must be combined to infer r.

$$\dfrac{p \vee q \qquad \dfrac{p \rightsquigarrow r \quad q \rightsquigarrow r}{p \vee q \rightsquigarrow r}}{r} \text{ left-disjunction}$$

A number of nonmonotonic logics, such as Reiter's default logic [94], lack the expressive power to make this inference. □

Principles of defeat: a warning

The main thrust of argumentation systems is to derive general principles for adjudicating among conflicting lines of argumentation. Here it will be shown that we must take care in defining such

[3] Nixon was the 36th President of the USA (1968-74). A quaker is a person who belongs to a Christian group called The Society of Friends.

principles. There is a point beyond which further inspection on the structure of arguments makes no sense.

We give two isomorphic but semantically distinct examples. These examples have the same syntax but a different semantics.

Benchmark 8.9 (whether bankrupt conservatives are selfish) [92]. Conservatives tend to be selfish, bankrupt conservatives tend to be poor, while poor people tend to be unselfish. James is a bankrupt conservative. Is James selfish?

A possible formalization is

$$P = \{c \rightsquigarrow s, (b \wedge c) \rightsquigarrow p, p \rightsquigarrow \neg s\} \cup \{b \wedge c\},$$

where c stands for conservative, s for selfish, and so forth. P is ϵ-consistent and enables

$$A = \dfrac{\dfrac{b \wedge c}{c} \quad c \rightsquigarrow s}{s} \qquad \text{and} \qquad B = \dfrac{\dfrac{b \wedge c \quad b \wedge c \rightsquigarrow p}{p} \quad p \rightsquigarrow \neg s}{\neg s}$$

Both arguments lack a statistical basis, so that neither one can be preferred on the basis of statistical considerations. The question is whether s or $\neg s$ should be accepted as warranted, on the basis of P.

Intuition. Suppose, for the sake of the argument, that we generally believe that, on the basis of P, conservatives, whether they are bankrupt or not, poor or not, always tend to be selfish. So, in the absence of further substantial (i.e. statistical) evidence, common sense compels us to believe s on the basis of P. □

Benchmark 8.10 (whether young adults are employed) [79]. Adults tend to be employed, young adults tend to be university students, while university students tend to be unemployed. Tom is a young adult. Is Tom employed?

A possible formalization is

$$Q = \{a \rightsquigarrow e, (y \wedge a) \rightsquigarrow u, u \rightsquigarrow \neg e\} \cup \{y \wedge a\}.$$

where a stands for adults, u for university students, and so forth. Q enables

$$C = \dfrac{\dfrac{y \wedge a}{a} \quad a \rightsquigarrow e}{e} \qquad \text{and} \qquad D = \dfrac{\dfrac{y \wedge a \quad y \wedge a \rightsquigarrow u}{u} \quad u \rightsquigarrow \neg e}{\neg e}$$

Both arguments lack a statistical basis, so that neither one can be preferred on the basis of statistical considerations. The question is whether e or $\neg e$ should be accepted on the basis of Q.

Intuition. Suppose, again for the sake of the argument, that we *know* that $\neg e$ is the case. We know $\neg e$, based on our experience and our commonsense knowledge of the world. Thus, besides knowing that Q is true (because we have accepted it as input), let us for the moment assume that we know that young adults are generally unemployed. □

Obviously, the two examples are isomorphic: if $\sigma = \{c/a, s/e, b/y, p/u\}$ is a variable renaming, then $Q = P\sigma$, $C = A\sigma$, and $D = B\sigma$.

Here is the problem: Benchmark (8.9) indicates that s should be believed on the basis of P. We have two criteria to adjudicate between competing arguments (157). These two criteria, however, do not point in the direction of s. If we think that this is a problem, i.e., if we think that defeasible logics should in this case prefer s at the expense of $\neg s$, then we are bound refine the conflict-resolution mechanism in such a way that A is preferred at the expense of B. If done in

the right way, s would be preferred as well, so that the semantics of the resulting defeasible logic is in line with common sense.

One way to enforce the "right" conclusion, is to prefer arguments that make less use of $\widetilde{\text{MP}}$ than other arguments. Let us call this the $<\widetilde{\text{MP}}$-*principle*. The $<\widetilde{\text{MP}}$-principle seems plausible, because it is hard to imagine a system for selecting among competing arguments that would not favor argument with less instances of $\widetilde{\text{MP}}$. (There might be other, more sophisticated, criteria than the $<\widetilde{\text{MP}}$-principle, but the current criterion suffices to get the point across.)

Thanks to the $<\widetilde{\text{MP}}$-principle, s emerges victorious. We conclude that, as far as (8.9) is concerned, the $<\widetilde{\text{MP}}$-principle is on a par with common sense. However, since (8.9) and (8.10) are isomorphic, the $<\widetilde{\text{MP}}$-principle will support $\neg e$ as well as it supports s, which contradicts our common sense knowledge that most students are unemployed ($\neg e$). Providing the system with other features of defeat will not help, since in that case (8.9) would be invalidated.

It seems that the $<\widetilde{\text{MP}}$-principle yields correct conclusions in one situation and incorrect conclusions in other situations. Thus, the $<\widetilde{\text{MP}}$-principle cannot act as a general principle to adjudicate between competing arguments, at least not for and (8.9) and (8.10). Neither can any other syntactical principle be used to this end, since the examples are syntactically equal.

More generally, the above case illustrates that in many situations unique and unambiguous solutions based on the syntactic structure of arguments, simply do not exist. Generally, one will need further information from the semantics of the domain to have a definite saying in which argument is better than the other. The history of search provides an instructive analogy here: at first, it was thought that general principles, such as α-β pruning, would yield tractable search spaces for a wide class of domains. But experience with these principles has shown that, in many cases, knowledge-intensive heuristics tailored to a given domain are necessary for adequate performance.

In this section, we have seen that the language of propositional logic supplemented with "$\rightsquigarrow$" cannot distinguish between at least two semantically different scenarios. Semantically different scenarios can be mapped onto a single set of formal premises. Conversely, the "inverse image" of this set of premises shows too much (semantic) variation to lift domain-independent principles of defeat out of it. We are bound to conclude that valid principles must, just because of their generality, be very weak.

8.4 What is an argument?

It is difficult to maintain that automated theorem proving methods, such as the analytic tableaux method and resolution, faithfully reflect everyday reasoning. The analytic tableaux method is more like an analysis of logical formulas, rather than a synthesis, or construction, of an argument. Likewise, the main objective of resolution is finding a resolution refutation,—not to reflect patterns of commonsense reasoning. In this context, think it is fair to state that resolution is more like cleverly manipulating clause sets than constructing a transparent and intuitively convincing argument. Alternative (and perhaps more intuitive) proof methods, such as natural deduction, suffer from the defect that their hypothesis introduction and discharge mechanism makes them unsuitable for automated theorem proving.

Example 8.11 Consider the problem to produce an argument for r based on

$$P = \{(p \wedge q) \rightsquigarrow r,\ p \rightsquigarrow (r \vee q),\ p\}.$$

If the "$\rightsquigarrow$"'s are replaced by "$\supset$"'s, then it is possible to prove r by means of the tableaux method or by resolution. (See first two chapters.) If we put the "$\rightsquigarrow$"'s back in, we have an argument.

Let us use the tableaux method and its proof-theoretic counterpart, the cut-free Gentzen sequent calculus to illustrate this idea. An automatically generated argument for r in the *style* of the cut-free Gentzen sequent calculus would be

$$\dfrac{\dfrac{\dfrac{\dfrac{p \Rightarrow p}{p \Rightarrow r, (p \wedge q), p}\ \text{thin} \qquad \dfrac{\dfrac{\dfrac{p \Rightarrow p}{p, q \Rightarrow r, p}\ \text{thin} \quad \dfrac{q \Rightarrow q}{p, q \Rightarrow r, q}\ \text{thin}}{p, q \Rightarrow r, (p \wedge q)}\ \text{right-}\wedge \quad \dfrac{r \Rightarrow r}{p, r \Rightarrow r, (p \wedge q)}\ \text{thin}}{p, (q \vee r) \Rightarrow r, (p \wedge q)}\ \text{left-}\vee}{p, (p \rightsquigarrow (q \vee r)) \Rightarrow r, (p \wedge q)}\ \text{left-}\rightsquigarrow \qquad X}{\text{conclusion: } ((p \wedge q) \rightsquigarrow r), p, (p \rightsquigarrow (q \vee r)) \Rightarrow r}\ \text{left-}\rightsquigarrow$$

where $X =$

$$\dfrac{r \Rightarrow r}{r, p, (p \rightsquigarrow (q \vee r)) \Rightarrow r}\ \text{thin}$$

and where "thin" stands for "thinning the antecedent". (Chapter 2.) A variable X is used, because the entire argument does not fit on the paper ....

The above structure, although simple in the sequent calculus, hardly counts as an ordinary argument: the entire structure is more like the analysis of a logical formula than a "train of reasoning". □

There are two other problems. The first problem is that general-purpose theorem provers are computationally inefficient. (Cf. previous chapter.) This is a relevant problem, because argumentation is a trial-and-error process in which usually a large number of different arguments are produced for and against a particular claim, before a certain thesis is established (or refuted). In contrast, theorem provers may stop after finding at least one proof.

A second problem with ATP methods is that commonsense reasoning is typically based on a vaguely bounded and immense reservoir of data stemming from perception and memory. Individual persons are in possession of hundreds of thousands of facts, observations, memories, and other data, while groups of persons have even more data as its disposal. Suppose, for the sake of the argument, that this "pool of knowledge" may be represented by the set D. Obviously, D is far too large act as the LHS of a sequent. I.e., if we would like to find an argument for e on the basis of D, the rules of the sequent calculus (and of the tableaux method, and of resolution refutation) would require that we put the entire database D in the LHS of the sequent $D \vdash q$. This, obviously, is not going to work in practical argumentation processes, and commonsense reasoning most likely does not work that way in the first place.

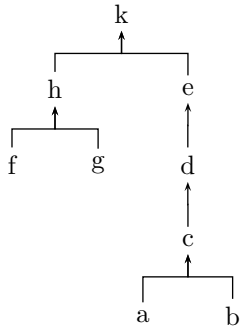
The above mentioned problems are partly a representation problem. If logical data may be represented in any format, then finding an argument supposes that we decipher and detect all logical dependencies in the data available that might be relevant to our main claim. This, obviously, is *also* not going to work in practical argumentation processes. Further, on page 48, it was shown that it is difficult to find proofs for arbitrarily formatted formulas. But obviously such formulas do not stem from practical problem domains, and seem therefore not as important as formulas that come from practical data. Thus, in commonsense reasoning, some formulas occur more often than other formulas.

A compromise that partly solves the above mentioned problems is to represent all data in a uniform format that is amenable to automated reasoning. One such approach is to divide data in rules and facts, and to construct arguments top-down by backward-chaining through the rules. This was done in the examples above.

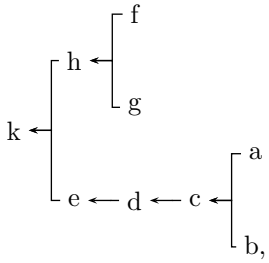
One step further is to represent arguments such as, say,

$$\dfrac{\dfrac{f \wedge g \quad f \wedge g \supset h}{h}\,\widetilde{\text{MP}} \qquad \dfrac{\dfrac{\dfrac{a \wedge b \quad a \wedge b \supset c}{c}\,\text{MP} \qquad c \rightsquigarrow d}{d}\,\widetilde{\text{MP}} \qquad d \supset e}{e}\,\text{MP} \qquad h \wedge e \rightsquigarrow k}{k}\,\widetilde{\text{MP}}$$

as



or



thus neglecting the uniformity of the rules of inference MP and $\widetilde{\text{MP}}$. A more common notation, then, is

$$\begin{array}{l} k \Leftarrow h \Leftarrow f \\ \qquad\quad\; g \\ \quad e \leftarrow d \Leftarrow c \leftarrow a \\ \qquad\qquad\quad\; b, \end{array}$$

where the double arrows represent defeasible implications. Another representation, with numbers, is

$$\begin{array}{l} k \leftarrow\!(0.56)\!- h \leftarrow\!(0.68)\!- f \\ \qquad\qquad\qquad\quad\; g \\ \qquad\qquad e \leftarrow\!(1.00)\!- d \leftarrow\!(0.86)\!- c \leftarrow\!(1.00)\!- a \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad\; b. \end{array}$$

The number in the arrow represents the so-called *implicational strength* (IS) of the rule, a number between 0 and 1 representing the transfer of support through that particular rule. For example, if s has strong support, and we know that $s \text{–}(0.87)\!\rightarrow \neg p$ is true, then it is plausible to assume that some of s's support will propagate through $\neg p$. Notice that strict (i.e., non-defeasible) implications correspond with a rule strength of 1.00.

The use of numbers to indicate the strength of an implication is simple but controversial. It is simple, because numbers can easily be manipulated by computers. It is controversial, because numbers are often too precise an indication for the implicational strength. Even worse: we often do not have the faintest idea what these numbers should be. Getting the right numbers for uncertain data is a classical problem in knowledge representation. Traditionally, they are, or *should be*, supplied by the knowledge-engineer or other domain experts. But the problem is that domain experts are often also people do not have the time, skills, or capacity to provide numerical data. For example, it is known that physicians have great difficulties in estimating probabilities and reverse probabilities consistently,–which is not their fault: it is definitely unreasonable to expect from experts to cough up a large amount of numbers just like that.

One solution is to translate numbers into verbal modalities like "deductively implies" (IS = 1) "strongly implies" ($0.9 \leq$ IS < 1) "suggests" ($0.7 \leq$ IS < 0.9), etc. Another solution is to acquire the numbers by machine learning techniques. (Chapter 11 on page 217.)

Here is another example:

Example 8.12 If it is foggy, cars tend to drive slower and lighten their head lamps. If it is foggy, cars tend to form a line. If cars form a line, they tend to drive slower. It is foggy. (No questions asked.)

This particular situation might be formalized into

$$P = \{f \rightsquigarrow (s \wedge h), f \rightsquigarrow l, l \rightsquigarrow s\} \cup \{f\}$$

where f stands for "foggy," s for "slow driving," and so forth. Possible arguments:

$$\frac{f \quad f \rightsquigarrow (s \wedge h)}{s \wedge h} \qquad \frac{\dfrac{f \quad f \rightsquigarrow (s \wedge h)}{s \wedge h}}{h} \qquad \frac{f \quad f \rightsquigarrow l}{l} \qquad \frac{\dfrac{f \quad f \rightsquigarrow l}{l} \qquad l \rightsquigarrow s}{s}$$

Without representing uniform rules of inference:

$$\begin{array}{c} s \wedge h \\ \uparrow \\ f \end{array} \qquad \begin{array}{c} h \\ \uparrow \\ s \wedge h \\ \uparrow \\ f \end{array} \qquad \begin{array}{c} l \\ \uparrow \\ f \end{array} \qquad \begin{array}{c} s \\ \uparrow \\ l \\ \uparrow \\ f \end{array}$$

If the strength of the defeasible implications "$\rightsquigarrow$" are graded with numbers, we might obtain something like

$$h \leftarrow\!\!(1.00)\!\!- s \wedge h \leftarrow\!\!(0.81)\!\!- f, \qquad l \leftarrow\!\!(0.68)\!\!- f,$$
$$s \wedge h \leftarrow\!\!(0.81)\!\!- f, \text{ and} \qquad s \leftarrow\!\!(0.94)\!\!- l \leftarrow\!\!(0.68)\!\!- f.$$

The numbers are arbitrary here, except in case of a deductive inference. In such cases, the strength of an implication is set to one. □

8.5 Probability theory

Probability theory is one of the most accurate calculi to reason with uncertain information. It deals appropriately with contraposition, transitivity, dependency, and other phenomena that are typical to non-deductive forms of reasoning. (Cf. Table 8.1 on page 155.) In order to work properly, however, probability theory needs a large number of given probabilities to depart from. For example, if every variable in the rule $a, b, c, d \rightarrow e$ can assume three different values, then $3^4 = 81$ conditional probabilities must be obtained (for only one rule). The problem is that the different probabilities that are needed are often hard to come by.

Ideally, some of the probabilities needed can be acquired by counting frequencies or by statistical sampling. These are called *objective probabilities*. But we do not live in the ideal world, so that objective probabilities are nearly always hard or impossible to get. As an alternative, software engineers must rely on domain experts to acquire so-called *subjective probabilities*, i.e., probabilities that are personal estimates as to the chance or probability that a certain event will occur. Clearly, without such accurate numerical assessments the subtle theory of probability is of little value. This argument has been made time and again perhaps most tersely by Cheeseman [14], who asked: "where all all the numbers coming from?"

Another point is that, even *if* the numbers are available, they seem to make little difference to the business of weighing up the evidence. This line of thought has been taken up by Wellman and others by making probability a qualitative rather than a numeric concept [127, 54, 95, 72].

The resulting solution called *qualitative probabilistic networks* is akin to the argumentation approach and there are indeed a number of connections. Cf. [90, 123].

A last point is that probability propagation (i.e., computing the right probability for each node) for realistically sized probabilistic networks is a complex process. The two existing authoritative algorithms (junction tree propagation and triangulation [52, 4, 63]) though efficient can only be applied locally but is globally intractable. In this respect, it is informative to note that the complexity of probabilistic inference was the reason why *ad hoc* propagation formalism such as certainty factors were invented in the first place.

More on the relation between probabilistic networks and argumentation in Section 9.5 on page 190.

8.6 Degree of belief

With defeasible reasoning and argumentation, the emphasis is more on procedure than on language and syntax. To this end, it was shown above how the relatively rich language of symbolic logic can be stripped down to an impoverished language that is more suitable to the study of argumentation processes. In this section, we will become more specific about the representation of arguments. Thus, we will become more specific about what is an argument.

For simplicity's sake we depart from the following elementary assumptions.

1. The most elementary language elements are $p, q, r, \ldots$ together with their negations $\neg p, \neg q, \neg r, \ldots$. (Also known as literals.)

2. Some (but not all) language elements enjoy a *degree of belief* (DOB), which is a number between 0 and 1. For example, if $\text{DOB}(p) = 0.75$, this means that p enjoys a degree of belief, or an irreducible degree of support, of 0.75.

3. A second type of language elements are rules of inference, e.g., formulas of the form

   $r, \neg t \dashv(0.97)\!\rightarrow p$
   $\neg u, \neg s, p \dashv(0.89)\!\rightarrow \neg r$
   $s \dashv(0.87)\!\rightarrow \neg p$

   Rules can also have a DOB. For example,

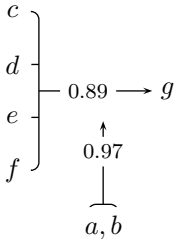
   $$\text{DOB}(\neg k, f, m \dashv(0.89)\!\rightarrow \neg c) = 0.67.$$

   Again, numbers can be translated into verbal modalities such as "certain" ($\text{DOB} = 1$), "evident" ($0.9 \leq \text{DOB} < 1$), "probable" ($0.5 \leq \text{DOB} < 0.9$), etc.

   How implicational strength and the DOB of rules combine will be explained in Sec. 8.7 on page 167.

4. In theory, it is possible to form expressions such as

   $a, b \dashv(0.97)\!\rightarrow (c, d, e, f \dashv(0.89)\!\rightarrow g)$, i.e.,

This expression is a rule R_1 with antecedent a, b, and with consequent the rule R_2, where $R_2 = c, d, e, f \text{ –}(0.89)\!\!\rightarrow g$. The antecedent of R_2 is c, d, e, f and the consequent of R_2 is G. It is possible to attach DOB-values to such expressions. Just say $\text{DOB}(R_1) = 0.45$.

According to Toulmin's famous argument schema (data / reason / warrant / claim / justification) [118], we would have data = $\{c, d, e, f\}$, reason = "$c, d, e, f \text{ –}(0.89)\!\!\rightarrow g$", claim = G, backing = "$a, b \text{ –}(0.89)\!\!\rightarrow$ reason", and justification = $\{a, b\}$.

5. Rules may chain into arguments. For example, if we have

$$\begin{array}{l} b, c \text{ –}(0.76)\!\!\rightarrow a \\ d, e \text{ –}(0.97)\!\!\rightarrow b \\ f, g \text{ –}(0.98)\!\!\rightarrow c \\ f \text{ –}(0.93)\!\!\rightarrow e, \end{array}$$

we may form the argument

$$\begin{array}{lll} a \leftarrow\!\!(0.76)\!\!- & b \leftarrow\!\!(0.97)\!\!- & d \\ & & e \leftarrow\!\!(0.93)\!\!- \; f \\ & c \leftarrow\!\!(0.98)\!\!- & f \\ & & g \end{array}$$

(read from left to right). This argument is grounded in d, f, and g and ends with conclusion a.

Item (4) and (5) describe different linguistic constructions: it is possible to have (long) arguments in which none of the rules themselves are justified. On the other hand, it is possible to have rules justifying each other while none of the rule antecedents is justified.

PROBLEMS (Sec. 8.6)

1. Give an example of a long argument for which none of the rules are justified. Is it possible to give a practical example? (Solution.)

2. Give an example of a constellation of rules that justify each other while no element of any rule antecedent is justified. Same question: is it possible to give a practical example? (Solution.)

3. Give an example of a (formal) argument of which none of the rules is justified. Can you think of a practical example of this argument? Give an example of a realistic argument for which it is plausible that none of the rules is called into question by an opponent. (Hint: mathematical discourse. Compare, if necessary, the arguments displayed at pp. 149-151.) (Solution.)

4. Give an example of a rule or reason that is supported by an argument that consists of more than one rule. Give an example of a rule or reason from daily practice that is believable only when it is supported by an adequate argument. (Hint: think of a "because"-reason that needs clarification.) (Solution.)

5. Give an example of a thesis that is supported by an argument of which some rules are the consequent of other rules, which are the consequent of still other rules, and so forth. (Hint: philosophical discourse.) (Solution.)

6. Let the following rules be given:

$P \leftarrow\!(0.75)\!- Q$	$Q \leftarrow\!(0.75)\!- P$	$R \leftarrow\!(0.75)\!- P$
$P \leftarrow\!(0.75)\!- \neg Q$	$Q \leftarrow\!(0.75)\!- \neg P$	$R \leftarrow\!(0.75)\!- \neg P$
$P \leftarrow\!(0.75)\!- R$	$Q \leftarrow\!(0.75)\!- R$	$R \leftarrow\!(0.75)\!- Q$
$P \leftarrow\!(0.75)\!- \neg R$	$Q \leftarrow\!(0.75)\!- \neg R$	$R \leftarrow\!(0.75)\!- \neg Q$
$P \leftarrow\!(0.75)\!- Q, R$	$Q \leftarrow\!(0.75)\!- P, R$	$R \leftarrow\!(0.75)\!- P, Q$
$P \leftarrow\!(0.75)\!- \neg Q, R$	$Q \leftarrow\!(0.75)\!- \neg P, R$	$R \leftarrow\!(0.75)\!- \neg P, Q$
$P \leftarrow\!(0.75)\!- Q, \neg R$	$Q \leftarrow\!(0.75)\!- P, \neg R$	$R \leftarrow\!(0.75)\!- P, \neg Q$
$P \leftarrow\!(0.75)\!- \neg Q, \neg R$	$Q \leftarrow\!(0.75)\!- \neg P, \neg R$	$R \leftarrow\!(0.75)\!- \neg P, \neg Q$

all with DOB $= 1$. Let P and $\neg P$ be literals with DOB(P) = DOB$(\neg P) = 1$.

(a) Give all arguments for R. (Solution.)

(b) Suppose DOB$(R) = 1$ as well. Again, give all arguments for R. (Solution.)

8.7 Degree of support

Support may be conceived as inferred degree of belief. Conversely, a degree of belief may be conceived an an irreducible degree of support. The idea behind local search is that support "flows" through the rules to propositions that thus receive an indirect, or inferred, support. We also speak of the *propagation of support* through rules.

Suppose

Proposition	DOB	where
$g \dashrightarrow\!(0.92)\!\rightarrow \neg h$	1.00	h = try-homer,
$g, l \dashrightarrow\!(0.98)\!\rightarrow h$	1.00	g = good-hitter,
$h \dashrightarrow\!(0.91)\!\rightarrow \neg t$	1.00	t = good-hit, and
$l, \neg h \dashrightarrow\!(0.97)\!\rightarrow t$	1.00	l = homered-lately.
h	0.87	
l	0.93.	

while DOB$(x) = 0.00$ for all other propositions x. If the DOB of h and l are known while the DOB of g and t are not, then the quintessential problem is: how do we compute the degree of support of propositions for which the degree of support is unknown? Obviously, the DOS-values can be obtained *in some way* by propagating the DOB-values through the rules, with the emphasis on "in some way". A number of important questions come to the fore:

1. Should the DOB of a proposition contribute to its DOS?
2. It is a good idea to update the DOB of a proposition with its DOS?
3. How do we cope with circular support?
4. How do we cope with contradictory support?

The answer to the first question is that whether or not incorporating the DOB in the DOS is a matter of convention, and there are argument for or against such a convention. An argument in favor to letting the DOB contribute to the DOS, is that DOB is a special kind of support, namely "self-support," that must also be incorporated into the DOS of a proposition. According to this approach, it is always the case that

$$\text{DOS}(x) \geq \text{DOB}(x), \text{ for every proposition } x \tag{8.9}$$

An argument against letting the DOB contribute to the DOS, is that the DOB is a form of epistemic appraisal that is obtained by means of sources other than inference, such as perception

and memory. In particular, the DOB and DOS are two different attributes of a proposition (so goes the second argument). We follow the latter approach.

Regardless of (1), the answer the question (2) is a definite "no". Let us consider the case in which $\text{DOB}(x)$ is not incorporated in $\text{DOS}(x)$, for every proposition x. Suppose

$$r = \text{“}A \text{–(0.80)}\rightarrow B\text{”}, \text{DOS}(r) = 1.00, \text{DOS}(A) = 1.00, \text{ and } \text{DOB}(B) = 0.50.$$

Since $\text{DOB}(B)$ is not incorporated in $\text{DOS}(B)$, most approaches would arrive at $\text{DOS}(B) = 0.80$. Now, if $\text{DOB}(B)$ is set to this new value, i.e., if $\text{DOB}(B)$ is set to 0.80, then the original information $\text{DOB}(B) = 0.50$ becomes lost, which is unfavorable. In the other case, when $\text{DOB}(x)$ is incorporated in $\text{DOS}(x)$, for every proposition x, then (8.9) creates a so-called *support-pump* for B. Often, $\text{DOS}(x) > \text{DOB}(x)$ which means that repeated updates of $\text{DOS}(x)$ bring it to 1.

When rules lead to inconsistencies (here t and $\neg t$), then a complicating difficulty is that support can't propagate freely as it may arrive to a proposition and its negation at the same time. Chapter 9 further deals with this issue (see page 9.3).

Generally speaking, there are two ways to deal with DOSs. The first category of standards, called *rule-based reasoning*, presupposes that it is possible to compute support recursively, first by computing the conclusive force of intermediary conclusions, and then combing the results of the intermediary conclusions through the last rule of inference (the top-rule) to obtain the conclusive force of the proposition in question. One trivial example of such a standard is a measure that counts the number of non-deductive rules used in an argument. The more non-deductive rules used, the weaker the argument. Clearly, this measure can be computed recursively, so that we may speak of a measure of conclusive force that can be determined locally. Another technique, reminiscent of certainty factors, is the subject of Chapter 9 (page 175 and further).

A second category of standards, known as *argument-based reasoning*, is computationally more expensive. As opposed to the previous category it does not allow recursive descent through rule antecedents but demands that we need to see the whole argument to determine its conclusive force and hence the support of its conclusion. This idea is further elaborated in Section 8.9 on page 171.

8.8 Global search

The purpose of this section is to explicate the concept of global search and to describe a number of aspects of it to contrast global search with forming DOSs on the rule level. Consider

Proposition	DOS
$b, c \text{–(0.89)}\rightarrow a$	1.00
$d \text{–(0.94)}\rightarrow c$	1.00
b	0.97
c	0.68
d	0.98

If a proponent of a, PRO, tries to establish a in a dispute that is steered by a global protocol, PRO would first make an inventory of (all) the arguments supporting a, in this case

Argument	Grounds
$a \leftarrow\!(0.89)\!\vdash\ b_{0.97}$ $\quad\quad\quad\quad\quad\quad\ c_{0.68}$	$\{b, c\}$
and	
$a \leftarrow\!(0.89)\!\vdash\ b_{0.97}$ $\quad\quad\quad\quad\quad\quad\ c \leftarrow\!(0.94)\!\vdash\ d_{0.98}$	$\{b, d\}$

In an artificial disputer such as IACAS [124], this would look like

```
1. (1) | PRO searches arguments for A ...
2. (1) | ... found 2, moving best to front ... done
3. (1) | first argument:
4. (1) | A <-(0.89)-
5. (1) |     B [0.97]
6. (1) |     C <-(0.94)- D [0.98]
7. (2) || CON accepts PRO's attempt to establish A
etc.
```

Many search procedures forget the second argument, because they search from A onward to propositions with DOS > 0, and halt if such propositions are found. Probably this search strategy is based on the idea that a certain argument cannot become stronger if it gets longer, called *sequential weakening* in [121]. This is true, but does not take away the fact that deeper search sometimes may yield more support, as the present case shows.

It is assumed here that arguments must ground in propositions with positive support. Thus $a_{0.00}$ is not an argument–at least not here. (When one performs hypothetical reasoning, e.g., for the sake of explanation or theory formation, then arguments that ground in zero support propositions are permitted.)

Once PRO has found all arguments that support a (line 2 above), it is time to employ them by bringing them into the discussion. Since he cannot throw them in all at once, PRO has to determine which argument shall be tried first. The criterion that will be used to order the arguments found is that of *expected success*: if an argument has a high chance of not being defeated in the rest of the debate, then it should be deployed first, for PRO may not get a second chance of putting forward a new argument.

However, it is difficult to determine the expected success of an argument, since we never know the effect of an argument until it has actually been tested in dispute. A measure of expected success that is normally used is simply the conclusive force, or *strength* of an argument, with the underlying idea that strong arguments are more likely to survive than weak arguments. This means that the problem of sorting arguments according to their expected success is reduced to the problem of sorting arguments by conclusive force.

PROBLEMS (Sec. 8.8)

1. Let

Proposition	DOB	*Proposition*	DOB
p	0.84	$p, q \dashv\!(0.86)\!\rightarrow s$	0.99
$p \dashv\!(0.78)\!\rightarrow q$	0.62	$p, r \dashv\!(0.93)\!\rightarrow s$	0.87
$p, q \dashv\!(0.91)\!\rightarrow r$	0.91	$r \dashv\!(0.95)\!\rightarrow t$	0.93
$q \dashv\!(0.98)\!\rightarrow r$	0.64	$s \dashv\!(0.97)\!\rightarrow t$	0.72

   (a) List all arguments for t (not necessarily with all the numbers). (Solution.)

(b) Suppose the strength of an argument is determined by the number of rules used, where more rules means a weaker argument. What is the strongest argument? Does this notion of strength correspond to your intuition? Explain your answer.

(c) Suppose the strength of an argument is determined by the strength of its weakest link, where a link is one of the following.

- A proposition with an explicit DOB. In that case, the strength of the link is the DOB of that proposition.
- A rule. In that case, the strength of the link is the DOB of the rule, multiplied by the strength of the rule.

For example, the strength of $r \leftarrow\!\!(0.98)\!\!- q \leftarrow\!\!(0.78)\!\!- p$ is equal to

$$\begin{aligned}&\min\{\text{DOB}(p), 0.78 * dos(p \text{ –}\!(0.78)\!\!\rightarrow q), 0.98 * dos(q \text{ –}\!(0.98)\!\!\rightarrow r)\}\\ &\quad = \min\{0.84, 0.78 * 0.62, 0.98 * 0.64\}\\ &\quad = \min\{0.84, 0.48, 0.63\}\\ &\quad = 0.48\end{aligned}$$

What is the strongest argument for s? Does this notion of strength correspond to your intuition? Explain your answer.

2. Design a datastructure for arguments. Write subroutines that return the conclusion of an argument, the premises of an argument and the strength of an argument. (Easy.) (Solution.)

3. An *immediate subargument* of an argument is a largest subargument that is not equal to the argument itself. For example, all immediate subarguments of the argument formed at Item 5 on page 166 are (in list-notation): $b \leftarrow\!\!(0.97)\!\!- [d, e \leftarrow\!\!(0.93)\!\!- f]$ and $c \leftarrow\!\!(0.98)\!\!- f, g$. Write a subroutine that returns all immediate subarguments of an argument. Write a subroutine that returns all (proper and improper) subarguments of an argument. (Easy.) (Solution.)

4. Write, or design, a subroutine that finds all arguments for a certain proposition. (Hard.)

- Does your subroutine allow circular arguments?
- Does it yield arguments of which all premises have an explicit DOS, or does it simply yield *all* arguments? If the latter is the case, try to modify your subroutine such that it yields only those arguments that ground in propositions with an explicit DOS.
- If a –$(x)\!\!\rightarrow b$ and b –$(y)\!\!\rightarrow c$ are rules, where both a and b have explicit DOSs, does your subroutine produces both arguments for c? I.e., does it produce $c \leftarrow\!\!(y)\!\!- b \leftarrow\!\!(x)\!\!- a$ as well as $c \leftarrow\!\!(y)\!\!- b$?

(Solution.)

5. A *counterargument* of an argument a is an argument with an opposite conclusion. Write a subroutine that finds all counterarguments of a given argument. [This should be easy, given the subroutine at (4).]

6. An argument a *interferes* with an argument b if a is a counterargument of b and b is not stronger than a. Write a subroutine that returns all arguments that interfere with a given argument.

7. An argument a *attacks* an argument b if it interferes with a subargument of b. Write a subroutine that finds all arguments that attack a given argument.

8. (Considerable amount of work.) Write a program that does the following.

- *Part I.* Given a proposition, find all arguments for that proposition.

- *Part II.* Verify, for every argument found at Part I, whether it is undefeated.

How would you qualify a conclusion if it is supported by an argument of strength 1.00? How about a conclusion that is supported by precisely one argument, and this argument has strength 0.60, while the negation has three counterarguments of strength 0.59?

8.9 Argument systems

Above, we have studied notions such as argument, counterargument, interference, and attack. From now on, we abstract away from the internal structure of arguments (and the reasons as to why these arguments attack each other) and assume that an attack relation between arguments is already defined. We are thus left with a clear and simple structure [25]:

Definition 8.13 (Argument system). An *argument system* is a 1-digraph[4] $G = (V, E)$, where nodes are interpreted as arguments, and edges are interpreted as attack relations between arguments. If $b \to a$ we say "b attacks a," "b is a counterargument of a," or "b is an objection to a". A reversed arrow $a \leftarrow b$ is pronounced "a is attacked by b".

Once an argument system is defined, the objective is to determine which arguments remain undefeated. Intuitively, an argument remains undefeated is all its counterarguments (if any) are defeated. Else, an argument is defeated. In this light, a counterargument c of an argument a is an attempt to defeat a. This attempt may succeed or not, depending on whether c is defeated itself. If c is defeated, it cannot defeat a. If *all* counterarguments of a are defeated, then no counterargument can defeat a so that a remains undefeated. In the other case, if c remains undefeated, it defeats a, so that a is defeated.

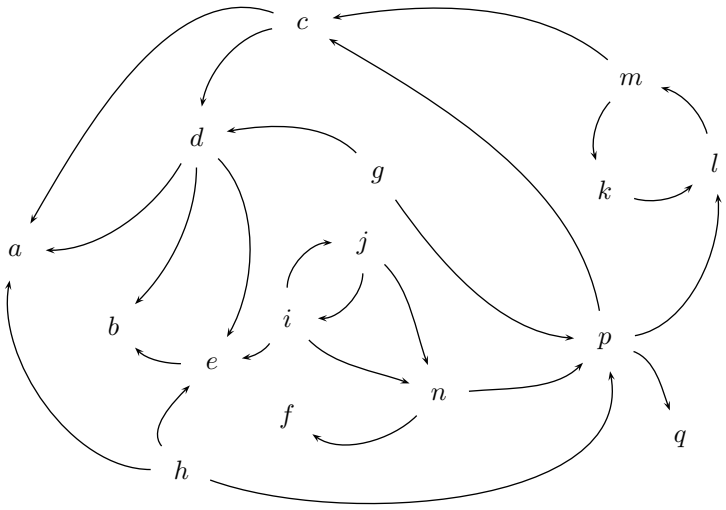


Figure 8.6: Attack relations in the running example.

[4] A digraph is a directed graph. A 1-digraph is a directed graph such that between every two points there is at most one edge. (But $a \to b$ and $b \to a$ is allowed.) If V are nodes, then a 1-digraph can be represented by a subset of the Cartesian product $V \times V$.

For example, $\mathcal{A} = \langle X, \leftarrow \rangle$ with arguments

$$X = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, p, q\}$$

and $\leftarrow$ as indicated in Fig. 8.6 is an (abstract) example of an argument system.

PROBLEMS (Sec. 8.9)

1. Consider the argument system displayed in Fig. 8.6 on the previous page. Which of the following statements are true? (Solution.)

- a is attacked by d	- n is attacked by i and j
- p has 3 attackers	- a is attacked by $\{d, l\}$†
- h has no attackers	- every attacker of q is attacked by n
- j is not attacked by $\{c, l\}$	- all attackers of e are attacked themselves
- m attacks one argument	- m is attacked by $\{a, l, f, g\}$

   †A set of arguments is said to *attack* an argument a if at least one element of that set attacks a.

2. Which arguments in Fig. 8.6 are defeated and which remain undefeated? (Solution.)

3. Consider the following naive definition of defeat among arguments:

   (a) The *attackers* of an argument a consist of all arguments that attack a.

   (b) An argument is *undefeated,* iff all its attackers are defeated. An argument is *defeated,* iff at least one of its attackers is undefeated.

   The following Prolog fragment suggests how we might compute arguments that are undefeated.

   ```
   undefeated(A) :-
      not defeated(A).

   defeated(A) :-
      defeats(_, A).

   defeats(A, B) :-
      attacks(A, B),
      undefeated(A).
   ```

   (a) Give an argument system for which every query of the form

   ```
   ?- undefeated(X).
   ```

   yields a correct answer. Which property must an argument system possess so that the above Prolog program halts?

   (b) Give an argument system for which the above Prolog program is caught in an endless loop.

   (c) (Considerable amount of work.) Rewrite the above code such that it is able to handle argument systems as given in the previous item. Can you program tell the difference between odd and even loops? Does it yield the right answer for floating arguments?

4. A set of arguments is said to be

   - *consistent,* or *conflict-free,* if that set does not contain arguments that attack each other;

- *complete* if every non-member is attacked by a member of that set;
- *stable* if it is conflict-free and complete.

(a) Prove: subsets of conflict-free sets are conflict-free; supersets of complete sets are complete.

(b) Let S be a stable set of arguments. Show that it satisfies the conditions of naive defeat, as mentioned in Exercise 3 on the facing page. (Solution.)

(c) Construct an argument system for which it is impossible to find a stable set of arguments. (Solution.)

(d) Construct an argument system for which it is possible to find more than one stable set of arguments.

5. An argument b is said to be attacked by a set of arguments A, if $b \leftarrow a$ for some $a \in A$. Let $\mathbf{E}(A) =_{Def} \{b \text{ is an argument} \mid b \text{ is not attacked by } A\}$, where A is a set of arguments. Prove:

   (a) The $\mathbf{E}$ operator is anti-monotonic: if $A \subseteq B$, then $\mathbf{E}(B) \subseteq \mathbf{E}(A)$.

   (b) If A is conflict-free, then $\mathbf{E}(A)$ is complete; if A is complete, then $\mathbf{E}(A)$ is conflict-free.

   (c) A is conflict-free if and only if $A \subseteq \mathbf{E}(A)$.

   (d) A is complete if and only if $\mathbf{E}(A) \subseteq A$.

   (e) (This should now be easy.) A is stable if and only if it is a fixed point of $\mathbf{E}$, i.e., if and only if $\mathbf{E}(A) = A$.

6. Stable extensions need not always exist, even with realistic argument systems. (Think of arguments pro and con in a legal case.) A much bigger problem with stable sets, however, is that they are extremely difficult to compute. Chvátal proved in 1973 that verifying whether a 1-digraph possesses a stable set (or *kernel*, in graph-theoretic terms), is NP-complete [15]. The cause of the computational intractability of stable sets is that, to verify whether a set of arguments S is stable, the algorithm has to find arguments in S against every possible argument outside S. However, the complement of S may be extremely large and typically contains arguments that are irrelevant to S. Such arguments need not be constructed, let alone that an algorithm must compute adequate defense for such arguments.

   A set of arguments D is said to *defend* an argument, a, if all attackers of a are attacked by an element of D. A set of arguments is said to be

   - *self-defending* if it defends its own members
   - *saturated* if it contains all arguments it defends
   - *admissible* if it is consistent and defends its own members
   - *preferred* if it is $\subseteq$-maximally admissible
   - *grounded* if it is $\subseteq$-minimally saturated
   - *perfect* if it is conflict-free, self-defending, and saturated.

   Let A and B be sets of arguments. Prove:

   (a) Stable sets are $\subseteq$-maximally compatible (but not conversely). Grounded, preferred and perfect sets are not always $\subseteq$-maximally compatible.

   (b) If a complete set is contained in a sound set, then these sets are equal.

   (c) If A is admissible, $A \subseteq \mathbf{E}^2(A) \subseteq \mathbf{E}^4(A) \subseteq \ldots \subseteq \mathbf{E}^5(A) \subseteq \mathbf{E}^3(A) \subseteq \mathbf{E}^2(A)$. (Solution.)

   (d) Each 2-stable set is clamped in between all terms of (6c). The same holds for stable, grounded, preferred and perfect sets.

   (e) The grounded set is equal to the lim inf of (6c).

(f) The concept of saturatedness in closed under finite intersection.

(g) Each stable set is preferred; each preferred set is perfect.

(h) Each grounded set is admissible and contained in a preferred set.

(i) A set is grounded if and only if it is $\subseteq$-minimally perfect; a set is preferred if and only if it is $\subseteq$-maximally perfect.

(j) Stable sets do not always exist; grounded, preferred and perfect sets do always exist.

(k) Stable, preferred and perfect sets generally are not unique. A grounded set is unique.

(l) A conflict-free union of two admissible sets is admissible.

(m) If an admissible set is not contained in a preferred extension, it is inconsistent with that preferred extension.

7. (Medium difficulty.) An argument is said to be *credulously preferred* if it is contained in at least one preferred set. Write an algorithm that computes for an arbitrary argument whether it is credulously preferred. Hint: it suffices to search for admissible sets. It even suffices to search for minimally admissible sets.

8. An argument is said to be *skeptically preferred* if it is contained in all preferred sets. Prove: an argument, a, is not skeptically preferred if and only if there is an admissible set, B, such that every minimally admissible set A that contains a is inconsistent with B. (NB: A is inconsistent with B iff $A \cup B$ is not conflict-free.) (Solution.)

9. (Downright difficult.) Use (8) to write an algorithm that computes for an arbitrary argument whether it is skeptically preferred.

Chapter 9

Rule-based reasoning

An essential aspect of argument-based reasoning is that we first form the arguments, and then see which arguments remain undefeated,—in that order. In other words, an essential aspect of argument-based reasoning is that we never reason with rules directly.

However, in the previous chapter it was argued that the assumption that argument strength is locally determined does not impose serious restrictions on our conception of conclusive force. From this assumption it follows that, in a dialogue, it is not necessary to let PRO and CON generate arguments in their entirety. Instead, it suffices for both parties to generate one rule (i.e., one elementary argumentation step) at a time and then gradually work backwards to the grounds on which the support ultimately rests. This non-argument based approach to reasoning with defeasible rules, is called rule-based reasoning.

9.1 Propagation of support

The flow of support through rules is determined by a principle that says how the DOS-values of the individual rules and propositions aggregate into a single DOS for the rule consequent.

A fairly standard principle is the following

Definition 9.1 (Propagation of support) Let

$$r \;=\; \text{“}p_1, \ldots, p_n \dashv(s)\!\rightarrow q\text{”}$$

be a rule with consequent q. If $\textsc{dos}(r)$, $\textsc{dos}(p_1)$, … , $\textsc{dos}(p_n)$ are known, then *q's support from* r, written $\textsc{dos}(r;q)$, is given by

$$\textsc{dos}(r;q) =_{Def} s * \min\{\textsc{dos}(p_1), \ldots, \textsc{dos}(p_n), \textsc{dos}(r)\} \tag{9.1}$$

Often, $\textsc{dos}(r) = 1$, so that $\textsc{dos}(r;q)$ is simply s times $\min\{\textsc{dos}(p_1), \ldots, \textsc{dos}(p_n)\}$. Often, one or all of $\textsc{dos}(r)$, $\textsc{dos}(p_1)$, … , $\textsc{dos}(p_n)$ are unknown, so that $\textsc{dos}(r;q)$ must be computed recursively.

This propagation formula is, since the the MYCIN experiments of the Stanford Heuristic Programming Project [11] and the concept of certainty-factors that ensued from this project, somewhat of a standard in the knowledge-engineering community [35, 109]. (But we are not going to treat certainty factors here.)

DOS-values often propagate through more than one rule.

Example 9.2 Suppose

Proposition	DOB	
r_1	1.00	where r_1 = "$b, c \dashv(0.89)\rightarrow a$"
r_2	1.00	where r_2 = "$d \dashv(0.94)\rightarrow c$"
a	0.00	
b	0.97	
c	0.00	
d	0.98	

Let us compute $\text{DOS}(r_1; a)$. I.e., let us compute the degree to which r_1 supports a. According to Eq. (9.1),

$$\text{DOS}(r_1; a) = s * \min\{\text{DOS}(r_1), \text{DOS}(b), \text{DOS}(c)\}.$$

Since there are no rules that further support b and r_1, we have $\text{DOS}(b) = \text{DOB}(b) = 0.97$, and $\text{DOS}(r_1) = \text{DOB}(r_1) = 1.00$. Hence,

$$\text{DOS}(r_1; a) = 0.89 * \min\{1.00, 0.97, \text{DOS}(c)\}.$$

$\text{DOS}(c)$ cannot be given immediately, because there are rules that further support c, viz. r_2. We now compute $\text{DOS}(c)$:

$$\text{DOS}(r_2; c) = 0.94 * \min\{\text{DOS}(r_2), \text{DOS}(d)\}$$

Since there are no rules that further support r_2 and d, we have $\text{DOS}(d) = \text{DOB}(d) = 0.98$, and $\text{DOS}(r_2) = \text{DOB}(r_2) = 1.00$. Hence,

$$\begin{aligned}\text{DOS}(r_2; c) &= 0.94 * \min\{\text{DOS}(r_2), \text{DOS}(d)\} \\ &= 0.94 * \min\{1.00, 0.98\} \\ &= 0.92\end{aligned}$$

Thus, we have

$$\begin{aligned}\text{DOS}(r_1; a) &= 0.89 * \min\{1.00, 0.97, 0.92\} \\ &= 0.82\end{aligned}$$

This example was arranged such that multiple DOS-values did not occur. This made it legal for us to assume that $\text{DOS}(a) = \text{DOS}(r_1; a)$ and $\text{DOS}(c) = \text{DOS}(r_2; c)$. But what if $\text{DOB}(c) > 0$? For example, what if $\text{DOB}(c) = 0.50$? Then how do values such as $\text{DOB}(c) = 0.50$ and $\text{DOS}(c) = 0.92$ combine? Do we take the maximum, do we add them up, or do we combine these values in some other way? The next section is devoted to this question.

Difference between rules and reasons

Definition 9.1 on the previous page provides the means to make a practical distinction between rules on the one hand and reasons on the other hand. We will now make this distinction.

In rule-based reasoning, a *rule* for q is a formula of the form r = "$p_1, \ldots, p_n \dashv(s)\rightarrow q$". Thus, we typically say that "r is a rule of strength s with antecedent $p_1, \ldots, p_n$ and consequent q". We also typically say that "r has support $\text{DOS}(r)$". Often $\text{DOS}(r) < s$.

A *reason* for q, on the other hand, is a rule for which $\text{DOS}(r; q)$ is known. For example, if $\text{DOS}(r; q) = 0.98$, we may say that "r is a strong reason for q," or that "r strongly supports q". Another example is when $s = 0.99$ but $\text{DOS}(r; q) = 0.01$. In that case, we would say that r is a strong rule but a weak reason for q. Thus, the distinction between a rule and a reason is that a reason for q is a supporter of q, while a rule with consequent q is not a supporter of q, because the support that r provides to q is unknown. Sec. 11.6 on page 229, discusses this distinction further.

PROBLEMS (Sec. 9.1)

1. Let $r =$ "$a, b \dashv(s)\mapsto c$".

   (a) What is the difference between $\text{DOS}(r)$, s, and $\text{DOS}(r; q)$? (Solution.)

   (b) Find values of $\text{DOS}(a)$, $\text{DOS}(b)$, $\text{DOS}(r)$, and s such that r is a strong rule but a weak reason for c. (Solution.)

   (c) Find values of $\text{DOS}(a)$, $\text{DOS}(b)$, $\text{DOS}(r)$, and s such that r is a weak rule but a strong reason for c. (Solution.)

   (d) Let $r =$ "$a, b \dashv(s)\mapsto c$", let $\text{DOS}(a) =$ and $\text{DOS}(b) = 0.5$. Draw the cube $[0, 1]^3$, and write $s \rightarrow$, $\text{DOS}(r) \rightarrow$ and $\text{DOS}(r; c) \rightarrow$ along the x, y and z-axis, respectively. Draw a graph of $\text{DOS}(r; c)$. [To obtain really accurate results you may use a graph-drawing program, such as `gnuplot`[1] to draw $\text{DOS}(r; c)$. I hasten to add that the use of software is not necessary to complete the problem.]

2. Let

Proposition	DOB	
r_1	1.00	where $r_1 =$ "$a, b \dashv(0.10)\mapsto c$"
r_2	1.00	where $r_2 =$ "$d, e \dashv(0.50)\mapsto f$"
r_3	1.00	where $r_3 =$ "$c, f \dashv(0.50)\mapsto g$"
a	0.50	
b	0.10	
d	0.50	
e	0.50	

   and $\text{DOB}(x) = 0.00$ for all other propositions.

   (a) Compute $\text{DOS}(g)$. (Solution.)

   (b) Same problem, but now also $\text{DOB}(f) = 1.0$ (Solution.)

   (c) Same, with $\text{DOB}(f) = 1.0$ and $\text{DOB}(g) = 0.2$. (Solution.)

3. This exercise is meant to emphasize the difference between argument-based reasoning and rule-based reasoning.

   (a) Give all arguments for a, based on the rules and propositions given in Example 9.2 on page 175. (Solution.)

   (b) Attach numbers to subconclusions of the arguments found under (3a). These numbers should indicate the support of these subconclusions (Solution.)

9.2 Independence and accrual of reasons

Two reasons accrue if their support accumulates. The idea is that support from independent sources accumulates, while support from dependent sources does not. Thus, the concept of accrual depends on the concept of independence.

Two (or more) reasons are said to be *independent* if they are based on unrelated evidence. For example: if Alice, Betty and Carol are three witnesses of a car-accident, and testify that they have seen a red Mercedes driving less than 80 miles per hour, then their testimony becomes more credible if they stood on three different points on the street at the time the accident happened. As an example of dependent testimonies we could imagine a scenario in which Alice, Betty and

[1] `gnuplot` is in the public domain and is able to produce various output formats, of which `*.ps` and `*.eps` are relevant to LaTeX, and `*.png` is relevant to HTML pages.

Carol drove the Mercedes themselves and are sisters. In the latter case, the testimonies are dependent and therefore do not strengthen each other.

It is often difficult to determine when two reasons are independent, and therefore the accrual of reasons is a controversial subject (Cf., e.g., [85, 91, 89] and [121].) The problem is that reasons sometimes accrue and sometimes do not, depending on the semantics of the rules. Consider the following example:

Proposition	DOS	where
$h \dashv(0.83)\mapsto \neg j$	1.00	$h =$ it is hot
$r \dashv(0.87)\mapsto \neg j$	1.00	$r =$ it rains
$h, r \dashv(0.91)\mapsto j$	1.00	$j =$ jogging is a pleasant activity
h	1.00	
r	1.00	

This example is due to Henry Prakken. Thus, in Prakken's perception, jogging is an unpleasant activity when it is hot or rains, but becomes pleasant again when it rains *and* it is hot. Probably the idea here is that, when it is hot, the rain works as some sort of water cooling. Considering the logical form of the problem itself, the trouble is that the first two reasons $h \dashv(0.83)\mapsto \neg j$ and $r \dashv(0.87)\mapsto \neg j$ may accrue (strength $(0.83 + 0.87)/(1 + 0.83 * 0.87) = 0.99$) thus unjustly defeating the more specific reason $h, r \dashv(0.91)\mapsto j$ for j. This is obviously not what we want, since $h, r \dashv(0.91)\mapsto j$ is more specific than its competitors and should therefore prevail.

The line of approach that we follow here is that reasons accrue unless they share variables in their antecedents. This is a conservative but safe criterion of accrual.

Definition 9.3 (Independent support) Let $R = \{r_1, \ldots, r_m\}$ be independent reasons for q. The *support of R for q*, written $\textsc{dos}(R; q)$, is given by

$$\textsc{dos}(R; q) =_{Def} \textsc{dos}(r_1; q) \oplus \cdots \oplus \textsc{dos}(r_m; q) \tag{9.2}$$

where "$\oplus$" is the so-called *hyperbolic sum*, defined by $x \oplus y =_{Def} (x + y)/(1 + xy)$. This sum behaves like ordinary addition (it is commutative and associative, for example) except that the outcome stays within $[-1, 1]$.

Thus, evidence that is supplied by independent reasons accumulates. The reason to use the hyperbolic sum rather than plain addition is that otherwise we would overshoot 1 in aggregating individual support. The above definition is, since the introduction of certainty-factors [11] somewhat of a standard in the knowledge-engineering community [35, 109]. However, there are differences. (See Exercise 3 on the facing page.)

Gross support

Gross support is all support a proposition receives, without taking into account the degree of belief in the proposition itself. Thus, gross support is all support a proposition may obtain through inference. Technically, gross support is determined by an independent set R of rules for q, and the guarantee that the accumulated support of R is as large as the support of any other set of independent rules for q. This idea is expressed in the next definition.

Definition 9.4 (Gross support) Let R be all reasons for q. Then

$$\textsc{gross}(q) =_{Def} \max\{\, \textsc{dos}(R; q) \mid R \text{ is an independent set of reasons for } q \,\}. \tag{9.3}$$

Although easy from a conceptual point of view, the above definition is difficult if you want to write an algorithm for it. First, you will have to find an independent set of rules for q, and then

you will have to determine whether the accumulated support of this set of rules is as large as the support of every other set of independent rules for q. Excercise 6 on the next page further addresses the problem how to compute the gross support.

Algorithm 3 Gross support (outline)

Input: Proposition p
Output: GROSS(p)

- return GROSS(p) **if** its value is cached
- $R = \varnothing$
- **foreach** rule r with consequent p **do**
  - next r **if** r is less specific than a rule r' with consequent $\neg p$ that has been used by the opponent in a previous move
  - compute DOS($r; p$)
  - determine which members of R do not depend on r
  - put r in R

  # R is now a weighted graph, where node r_1 is linked to node r_2 iff r_1 and r_2 are independent,
  # and where the weight of each node r is equal to DOS($r; p$).
- $\{r_1, \ldots, r_h\}$ = `heaviest_independent_subset_of`(R) # not a trivial function
- GROSS(p) = DOS($r_1; p$) $\oplus \cdots \oplus$ DOS($r_h; p$)
- cache GROSS(p), and return its value

Finally, to compute DOS(q) we will have to combine q's gross support with q's degree of belief and

Definition 9.5 (Degree of support)

$$\text{DOS}(q) =_{Def} \max\{\text{GROSS}(q), \text{DOB}(q)\} \tag{9.4}$$

In most cases, the computation of (9.4) is not that hard because the number of possible configurations is limited due to, e.g., the absence of a DOB, the absence of rule-dependencies, or the absence of rule-independencies.

PROBLEMS (Sec. 9.2)

1. Let $a, b \dashv(0.50)\rightarrow d$ and $a \dashv(0.40)\rightarrow d$ and $c \dashv(0.10)\rightarrow d$ and a and b and c be propositions with a DOB equal to 1.00. Let DOB(d) = 0.00. Compute GROSS(d), under the assumption that rules that share variables in their antecedent are dependent. (Solution.)

2. Let

Proposition	DOB	
r_1	1.00	where r_1 = "$b, c \dashv(0.89)\rightarrow a$"
r_2	1.00	where r_2 = "$d \dashv(0.94)\rightarrow c$"
a	0.15	
b	0.97	
c	0.63	
d	0.98	

   Compute GROSS(a).

3. Let $x \oplus y$ be the hyperbolic sum defined by $(x + y)/(1 + xy)$. For which values of $-1 \leq x, y \leq 1$ is $x \oplus y$ undefined? For which values is the hyperbolic sum equal to 1? Show that $x + 0 = x$, for every $x \in [-1, 1]$. Show that the hyperbolic sum is commutative, i.e. $x \oplus y = y \oplus x$. Show that the hyperbolic sum is associative, i.e. $x \oplus (y \oplus z) = (x \oplus y) \oplus z$. Write out $a \oplus b \oplus c \oplus d$ with "+" and "/". (Solution.)

4. Gross support is computed by first determining a heaviest subset of independent reasons, and then accumulating the DOS's of those reasons. The converse is also possible. An *independent partition* of R is a set $\{X_1, \ldots, X_n\}$ of subsets of R, such that

   - $R = X_1 \cup \ldots \cup X_n$,
   - $X_i \cap X_j = \varnothing$ for every i, j, and
   - every dependent pair of rules belongs to the same X_i.

   An independent partition always exists: $\{R\}$ is an independent partition of R. A set of *respresentatives* of an independent partition is a set of rules $x_1, \ldots, x_n$ such that $x_i \in X_i$ and $\text{DOS}(x_i) \geq \text{DOS}(x)$ for every other $x \in X_i$. An alternative way to define gross support would be

   $$\text{GROSS}'(q) =_{Def} \max\{\, x_1 \oplus \cdots \oplus x_n \mid x_1, \ldots, x_n \text{ are representatives of an independent partition of rules for } q \,\} \quad (9.5)$$

   Show that $\text{GROSS}'(q) \leq \text{GROSS}(q)$. Show that in some cases the inequality is strict. (Solution.)

5. In the MYCIN project [11],

   $$x \oplus y =_{Def} x + y - xy \quad (9.6)$$

   for positive x and y. Show that this definition can also be used as a pseudo-addition to compute the accrual of independent support. In particular, prove the following five properties: commutativity, associativity, unity element zero $(\forall x \in [0,1])(x \oplus 0 = x)$, closed $(\forall x, y \in [0,1])(x \oplus y \in [0,1])$, and monotonically increasing $(\forall x, y \in (0,1])(x \oplus y > \max\{x, y\})$.

   Later, we will see why (9.6) is (or was) not such a good idea after all.

6. Let

Proposition	DOB	
r_1	1.00	where r_1 = "$a, b \text{–}(0.60)\!\rightarrow g$"
r_2	1.00	where r_2 = "$b, c \text{–}(0.20)\!\rightarrow g$"
r_3	1.00	where r_3 = "$d, e \text{–}(0.40)\!\rightarrow g$"
r_4	1.00	where r_4 = "$e, f \text{–}(0.80)\!\rightarrow g$"

   Let $\text{DOB}(x) = 0.50$ for all other propositions.

   (a) Draw a graph with nodes r_i, $1 \leq i \leq 4$ and an edge from node r_i to r_j, $i \neq j$, if r_i and r_j share variables.

   (b) How many independent sets $R = \{r_1, r_2, r_3, r_4\}$ do exist? Give all of them.

   (c) Determine $\text{GROSS}(g)$.

7. Let R be all rules for a proposition q.

   (a) Give a greedy algorithm that approximates $\text{GROSS}(q)$ in time linear to the size of R. Hint: the inductive step runs as follows. Let $R = \{r_1, \ldots, r_k\}$, and let the first l elements of R be put in $R_1, ..., R_i$ such that rules from different R_j, $1 \leq j \leq i$, are independent. Then put r_{l+1} in R_j, such that $R_j \cup \{r_{l+1}\}$ is independent and $\text{DOS}(R_j \cup \{r_{l+1}\})$ is maximized. If there is no such j, then $R_{i+1} =_{Def} \{r_{l+1}\}$.

   (b) (Difficult.) Give an algorithm that computes $\text{GROSS}(q)$. Hint 1: most likely, your algorithm will be recursive. Hint 2: it is almost certain that your algorithm uses time exponential in the size of R. Hint 3: do a websearch on `"weighted independent set"`.

9.3 Support for contradictory propositions

When there is support for contradictory propositions, the situation becomes more complex. For example, if

Proposition	DOS	Proposition	DOS
$b, c \dashv(0.89)\rightarrow a$	1.00	b	0.97
$d \dashv(0.94)\rightarrow c$	1.00	c	0.00
$d \dashv(0.93)\rightarrow \neg a$	1.00	d	0.98
$a, \neg a \dashv(1.00)\rightarrow \bot$	1.00	$\bot$	0.00,

it is no longer clear which DOS-value a should assume. In particular, a DOS that reaches, or would reach $\bot$ through

$$\begin{array}{l} \bot_{0.99} \leftarrow\!(1.00)\!- a_{0.85} \leftarrow\!(0.89)\!- b_{0.97} \\ \qquad\qquad\qquad\qquad\qquad\quad\; c_{0.95} \leftarrow\!(0.97)\!- d_{0.98} \\ \qquad\qquad\quad \neg a_{0.91} \leftarrow\!(0.93)\!- d_{0.98} \end{array}$$

must be brought back to 0, since $\bot$ represents a contradictory proposition for which the DOS must remain zero. The problem is that it is not immediately clear how we can formulate criteria that ensure that DOS($\bot$) always remains clamped at zero. There are several solutions possible. A typical solution from the direction of computational dialectic is to let one party, PRO, seek and gather support for a and to let a second party, CON, seek and gather support PRO finds against a, i.e., for $\neg a$. In doing so, there are three possible outcomes:

1. PRO has found support for a that is stronger than all support CON could find against a. In that case, DOS(a) is equal to the DOS that has propagated through PRO's winning argument, while DOS($\neg a$) is then set equal to zero.
2. Not (1), but PRO has found as much support for a as CON has found against a. One option in this case is to let DOS(a) be equal to the DOS that has propagated thorough PRO's winning argument. (Motivation: CON's counterargument must be really stronger than PRO's argument.) Another option is to let both a and $\neg a$ share half of the strength of one of the two (equivalent) arguments that lie at the basis of the equilibrium.
3. Not (1) and not (2), i.e., PRO was unable to find a reason that could stand up to counterarguments put forward by CON. Suppose the conclusion of CON's winning reason is e (e could be equal to $\neg a$ but not necessarily so.) Then DOS(a) is set equal to zero and DOS(e) is set equal to the DOS that has propagated through CON's winning argument.

Thus, the computational dialectic paradigm ensures that, in all cases, the inferred DOS of a proposition (here a) and its negation (here $\neg a$) cannot be both positive, which is a solution to the problem mentioned above. Recent work in this area has proven that this solution is satisfactory, insofar that it prevents positive support from propagating to propositions with low DOSs, notably $\bot$.

Nett support

If there are no contradictory propositions, then nett and gross support amounts to the same. In such cases, NETT(a) = GROSS(a) = DOS(a), for every proposition a.

Else:

Definition 9.6 (Nett support)

$$\textsc{nett}(p) = \begin{cases} \textsc{gross}(p) & \text{if } \textsc{gross}(p) \geq \textsc{gross}(\neg p) \\ 0 & \text{otherwise} \end{cases} \tag{9.7}$$

$$\textsc{nett}(\neg p) = \begin{cases} \textsc{gross}(\neg p) & \text{if } \textsc{nett}(p) = 0 \\ 0 & \text{otherwise} \end{cases}$$

Thus, if $\textsc{gross}(\neg a)$ is close to one but $\textsc{gross}(a)$ is even closer to one, then (9.7) says that the a "gets all," and $\neg a$ "gets nothing". This all-or-nothing approach to contradictory evidence is typical to computational dialectic, as opposed to other approaches in which contradictory evidence weakens each other.

Finally,

$$\textsc{dos}(a) =_{Def} \textsc{nett}(a), \tag{9.8}$$

since $\textsc{nett}(a)$ is the final outcome of weighing all pros and cons.

Example 9.7 Let

Prop.	DOB	Prop.	DOB	
a	0.91	f	0.00	
b	0.63	r_1	1.00	where r_1 = "a, b –(0.89)→ f"
c	0.74	r_2	1.00	where r_2 = "b, c –(0.94)→ f"
d	0.86	r_3	1.00	where r_3 = "d –(0.88)→ $\neg f$"
e	0.98	r_4	1.00	where r_4 = "e –(0.73)→ $\neg f$"

and $\textsc{dob}(x) = 0.00$ for all other propositions. We compute $\textsc{dos}(f)$ and $\textsc{dos}(\neg f)$. This amounts to computing $\textsc{nett}(f)$ and $\textsc{nett}(\neg f)$. To this end, we need to know $\textsc{gross}(f)$ and $\textsc{gross}(\neg f)$. Let us start with $\textsc{gross}(f)$. To compute $\textsc{gross}(f)$, we need to know which rules support f. These are r_1 and r_2. Since r_1 and r_2 share variables, they are dependent, so that

$$\begin{aligned} \textsc{gross}(f) &= \max\{\textsc{dos}(r_1; f), \textsc{dos}(r_2; f)\} \\ &= \max\{\min\{0.91, 0.63\} \times 0.89, \min\{0.63, 0.74\} \times 0.94\} \\ &= 0.59. \end{aligned}$$

Since r_3 and r_4 are independent,

$$\begin{aligned} \textsc{gross}(\neg f) &= \textsc{dos}(r_1; f) \oplus \textsc{dos}(r_2; f) \\ &= (0.86 \times 0.88) \oplus (0.98 \times 0.73) \\ &= 0.96. \end{aligned}$$

Hence, $\textsc{dos}(f) = \textsc{nett}(f) = 0.00$ so that $\textsc{dos}(\neg f) = \textsc{nett}(\neg f) = 0.96$.

Thus, we obtain an algorithm, or actually two algorithms: one copy per party, each with its own storage of remembering previous results (Algorithm 4 on the facing page, Algorithm 3 on page 179).

PROBLEMS (Sec. 9.3)

1. Let

Prop.	DOB	Prop.	DOB	
a	0.61	f	0.00	
b	0.93	r_1	1.00	where r_1 = "a, b –(0.84)→ f"
c	0.84	r_2	1.00	where r_2 = "c, d –(0.85)→ f"
d	0.76	r_3	1.00	where r_3 = "d –(0.91)→ $\neg f$"
e	0.38	r_4	1.00	where r_4 = "a, b, e –(0.97)→ $\neg f$"

Algorithm 4 Nett support

Input: Proposition p
Output: NETT(p)
1: - return NETT(p) **if** its value is cached
2: - compute GROSS(p)
3: - let opponent compute GROSS($\neg p$)
4: - NETT(p) = GROSS(p) $\geq$ GROSS($\neg p$) ? GROSS(p) : 0.00
5: - cache NETT(p), and return its value

and DOB(x) = 0.00 for all other propositions. Compute DOS(f) and DOS($\neg f$).

2. Let

	DOB		DOB
p	1.00	p –(0.6)→ $\neg q$	1.00
p –(0.5)→ q	1.00	q –(1.0)→ $\neg r$	1.00
p –(0.4)→ r	1.00	r –(0.8)→ s	1.00

be propositions.

   (a) Give all arguments for s. Give all arguments for $\neg r$. Give all arguments for $\neg q$. (Solution.)

   (b) Compute the nett support of s. (To compute the nett support of s, it is necessary to compute the gross support and nett support of all propositions.) (Solution.)

   (c) Suppose the rule p –(0.5)→ q is replaced by the rule p –(0.7)→ q. What effects does this change have on the nett support of s? (Solution.)

   (d) Suppose p –(0.5)→ q [or p –(0.7)→ q] is replaced by p –(0.6)→ q. What are the effects for the nett support of s? (Solution.)

   (e) What is DOS(s)? (Solution.)

3. Let a and $\neg a$ be two contradictory propositions, such that GROSS(a) = x and GROSS($\neg a$) = y. The "hard way" of dealing with contradictory propositions is the following:

   NETT(a) =$_{Def}$ x and NETT($\neg a$) =$_{Def}$ 0 if $x \geq y$,
   NETT(a) =$_{Def}$ 0 and NETT($\neg a$) =$_{Def}$ y, otherwise.

   (a) Draw the cube $[0,1]^3$, and write DOS(a) →, DOS($\neg a$) → and NETT(a) → along the x, y and z-axis, respectively. Draw a graph of NETT(a). Use a graph-drawing program, such as `gnuplot` to draw NETT(a) more accurately. (Solution.)

4. Let a and $\neg a$ be two contradictory propositions, such that GROSS(a) = x and GROSS($\neg a$) = y. The "soft" way of dealing with contradictory propositions is the following. Let $z = x \oplus (-y) = (x - y)/(1 - xy)$.

   SOFT_NETT(a) =$_{Def}$ z and SOFT_NETT($\neg a$) =$_{Def}$ 0 if $z \geq 0$,
   SOFT_NETT(a) =$_{Def}$ 0 and SOFT_NETT($\neg a$) =$_{Def}$ $-z$, otherwise.

   (a) Prove that SOFT_NETT(a) ≥ 0 and SOFT_NETT($\neg a$) ≥ 0. Now argue that SOFT_NETT(x) ≥ 0, for every proposition x.

   (b) Draw the cube $[0,1]^3$, and write DOS(a) →, DOS($\neg a$) → and SOFT_NETT(a) → along the x, y and z-axis, respectively. Draw a graph of SOFT_NETT(a). Use a graph-drawing program, such as `gnuplot` to draw SOFT_NETT(a) more accurately. (Solution.)

(c) (See also Exercise 5 on page 180.) The certainty factor concept [11] coincides with our notion of support, but deals differently with contradictory information. In particular,

$$x \oplus' y =_{Def} \begin{cases} x + y - xy, & x \geq 0, y \geq 0 \\ \dfrac{x+y}{1-\min\{|x|,|y|\}}, & xy \leq 0 \\ x + y + xy, & x \leq 0, y \leq 0 \end{cases} \tag{9.9}$$

Draw a graph of $g : [-1,1]^2 \to [-1,1] : (x,y) \to x \oplus' y$, where $\oplus'$ is defined as in (9.9). Describe the similarities and differences between both graphs. (Solution.)

9.4 Sample runs

Here are some examples of Algorithm 4 and 3. To become acquainted with the format of the dialogues presented here, let us start looking at the most simple case we can think of.

Case 9.8 (No grounds) In this case a dispute is conducted on the basis of no grounds. PRO tries to establish a claim A

1 0,0 pro: I claim that A holds.
2 1,0 con: Why?
3 0,0 pro: Um... upon closer inspection
4 0,0 pro: I see I have no grounds for A.

Instead of discussing what happens here (not much, actually) I explain the format of the dispute. The first row of numbers are line numbers. Of the second row of numbers (separated by a comma) the first row indicates the *level* of the dispute, i.e., the number of times that the burden of proof has alternated. For example, if PRO starts a dispute on A and, within this dispute, CON starts a dispute on B and, within CON's dispute, PRO starts a dispute on C, then the dispute is at level three. The second half of the row of numbers that are separated by a comma indicates the *depth* of the dispute, i.e., the number of times that the defending party justifies his claim via regression through rules. Both parties follow the rules as defined in the algorithms on page 183, but because of the absence of material, the dispute quickly ends. □

We now move on to a case in which PRO has grounds to believe A:

Case 9.9 (Basic belief) Suppose that both parties agree on the fact that A has a DOS of 0.8:

1 0,0 pro: I claim that A holds.
2 1,0 con: Why?
3 0,0 pro: Simply because A is the case with DOS=0.80
4 1,0 con: Frankly, I am willing to contest A:
5 1,0 con: I claim that $\neg A$ holds.
6 2,0 pro: Why?
7 1,0 con: Um... upon closer inspection
8 1,0 con: I see I have no grounds for $\neg A$.
9 0,0 pro: That leaves me with A, obviously.

We see that the burden of proof changes two times (cf. line 1-6). The outcome of the dispute at level 2 then "bubbles up" rapidly at lines 6, 7 and 8. □

Now for a case in which both parties have arguments.

Case 9.10 (Arguments pro and con) PRO and CON have arguments of the most simple sort for and against A, respectively.

Proposition	DOS	Proposition	DOS
B	1.00	B –(0.97)→ A	1.00
C	1.00	C –(0.91)→ $\neg A$	1.00

Accordingly, the dispute then evolves as follows:

1	0,0 pro:	I claim that A holds.
2	1,0 con:	Why?
3	0,0 pro:	Because it follows from B with strength 0.97, and
4	0,1 pro:	I claim that B holds.
5	1,1 con:	Why?
6	0,1 pro:	Simply because B is the case with DOS=1
7	1,1 con:	Frankly, I am willing to contest B:
8	1,1 con:	I claim that $\neg B$ holds.
9	2,1 pro:	Why?
10	1,1 con:	Um... upon closer inspection
11	1,1 con:	I see I have no grounds for $\neg B$.
12	0,1 pro:	That leaves me with B, obviously.
13	0,0 pro:	And via B my A is supported 0.97
14	1,0 con:	Sure, but now this:
15	1,0 con:	I claim that $\neg A$ holds.
16	2,0 pro:	Why?
17	1,0 con:	Because it follows from C with strength 0.91, and
18	1,1 con:	I claim that C holds.
19	2,1 pro:	Why?
20	1,1 con:	Simply because C is the case with DOS=1
21	2,1 pro:	Frankly, I am willing to contest C:
22	2,1 pro:	I claim that $\neg C$ holds.
23	3,1 con:	Why?
24	2,1 pro:	Um... upon closer inspection
25	2,1 pro:	I see I have no grounds for $\neg C$.
26	1,1 con:	That leaves me with C, obviously.
27	1,0 con:	And via C my $\neg A$ is supported 0.91
28	1,0 con:	Nevertheless, I grant that earlier on (line 13), you were
29	1,0 con:	able to support A more strongly. Therefore I drop $\neg A$.
30	0,0 pro:	Combining my support (line 1-13) and your counter-
31	0,0 pro:	support (line 15-27) yields 0.97 for A,
32	0,0 pro:	which means that I am right on A.

We see that the localist protocol (page 183) forces both parties to enter and explore every nook and cranny of the search space immediately, i.e., on the rule level. To us (humans) this may look tedious, but it is the only way to explore the dialectic search space exhaustively.

If C –(0.98)→ $\neg A$ (instead of 0.91), then CON wins:

1	0,0 pro:	I claim that A holds.
⋮	⋮	⋮ [approximately the same lines as in the previous dispute]
27	1,0 con:	And via C my $\neg A$ is supported 0.98
28	2,0 pro:	Sure, but now this:
29	2,0 pro:	I claim that A holds.
30	3,0 con:	Why?
31	2,0 pro:	Don't you remember? I already argued this earlier on:

32 2,0 pro: At line 13 I concluded that A is supported 0.97
33 2,0 pro: Nevertheless, I grant that earlier on (line 27), you were
34 2,0 pro: able to support $\neg A$ more strongly. Therefore I drop A.
35 1,0 con: Combining my support (line 15-27) and your counter-
36 1,0 con: support (line 1-13) yields 0.98 for $\neg A$,
37 1,0 con: which means that I am right on $\neg A$, after all.
38 0,0 pro: Seems I have to give up on A (line 1). Fair enough.

Note how this time PRO is sent back into the defense by CON (line 28-29), since this time CON has found better support for $\neg A$ (0.98) than PRO has found for A (0.97). □

The next case is known as the Nixon diamond, because it is the canonical example of symmetry in nonmonotonic reasoning [33].

Case 9.11 (Nixon diamond) As with Case 9.10, but now with

Proposition	DOS	Proposition	DOS
B	1.00	$B \dashv(0.97)\rightarrow A$	1.00
C	1.00	$C \dashv(0.97)\rightarrow \neg A$	1.00

To obtain the Nixon diamond, let A = "Nixon is a Pacifist", B = "Nixon is a Quaker" and C = "Nixon is a Republican". Thus: Quakers tend to be Pacifists, Republicans tend to be non-Pacifists, Nixon is both a Republican and a member of the Quakers – is Nixon a Pacifist?

1 0,0 pro: I claim that A holds.
⋮ ⋮ ⋮ [approximately the same lines as in the dispute in Case 9.10]
22 1,1 con: That leaves me with C, obviously.
23 1,0 con: And via C my $\neg A$ is supported 0.97
24 1,0 con: I recall that, earlier on, you were able to support A as strong as I now support $\neg A$.
25 0,0 pro: Combining my support (line 1-11) and your countersupport (line 13-23) yields 0.97 for A. Since I put forward A first you had the burden to outweigh my support for A. You merely equaled it. That is not enough, so A holds.

Thus if PRO and CON have equivalent arguments, then Definition 9.6 on page 181 part I, is applied. □

Case 9.12 (Accrual of reasons) As with Case 9.10, but now with a second rule $D \dashv(0.93)\rightarrow A$, thus providing additional support against A:

Proposition	DOS	Proposition	DOS
B	1.00	$B \dashv(0.97)\rightarrow A$	1.00
C	1.00	$C \dashv(0.91)\rightarrow \neg A$	1.00
D	1.00	$D \dashv(0.93)\rightarrow \neg A$	1.00

1 0,0 pro: I claim that A holds.
⋮ ⋮ ⋮ [approximately the same lines as in the previous dispute]
26 1,1 con: That leaves me with C, obviously.
27 1,0 con: And via C my $\neg A$ is supported 0.91
28 1,0 con: Further, $\neg A$ follows from D with strength 0.93, and

29 1,1 con: I claim that D holds.
30 2,1 pro: Why?
31 1,1 con: Simply because D is the case with DOS=1
32 2,1 pro: Frankly, I am willing to contest D:
33 2,1 pro: I claim that $\neg D$ holds.
34 3,1 con: Why?
35 2,1 pro: Um... upon closer inspection
36 2,1 pro: I see I have no grounds for $\neg D$.
37 1,1 con: That leaves me with D, obviously.
38 1,0 con: And via D my $\neg A$ is supported 0.93
39 1,0 con: Combining support for $\neg A$ (at 17 and 28)
40 1,0 con: makes 0.99 in total.
⋮ ⋮ ⋮ [uninteresting and fruitless attempt of pro]
47 1,0 con: Combining my support (line 15-39) and your
48 1,0 con: countersupport (line 1-13) yields 0.99 for $\neg A$,
49 1,0 con: which means that I am right on $\neg A$, after all.
50 0,0 pro: Seems I have to give up on A (line 1). Fair enough.

Lines 39-40 are crucial here, because CON applies the principle of accrual of reasons to create "synergy" between $C \dashrightarrow\!\!\!\!\!(0.91)\!\!\!\!\!\rightarrow \neg A$ and $D \dashrightarrow\!\!\!\!\!(0.93)\!\!\!\!\!\rightarrow \neg A$. □

The next case is like the previous case, except that C and D are now combined into one rule, which means that C and D are dependent.

Case 9.13 (Two grounds) Consider

Proposition	DOS	Proposition	DOS
B	1.00	$B \dashrightarrow\!\!\!\!\!(0.97)\!\!\!\!\!\rightarrow A$	1.00
C	1.00	$C, D \dashrightarrow\!\!\!\!\!(0.99)\!\!\!\!\!\rightarrow \neg A$	1.00
D	1.00		

Again, the deeper levels are omitted.

1 0,0 pro: I claim that A holds.
⋮ ⋮ ⋮
15 1,0 con: I claim that $\neg A$ holds.
16 2,0 pro: Why?
17 1,0 con: Because it follows from C and D with strength 0.99,
18 1,0 con: and I claim C and D. In particular,
19 1,1 con: I claim that C holds.
⋮ ⋮ ⋮
27 1,1 con: That leaves me with C, obviously.
28 1,1 con: I claim that D holds.
⋮ ⋮ ⋮
36 1,1 con: That leaves me with D, obviously.
37 1,0 con: And via C and D my $\neg A$ is supported 0.99
⋮ ⋮ ⋮ [same uninteresting and fruitless attempt of pro as before]
45 1,0 con: Combining my support (line 15-37) and your counter-
46 1,0 con: support (line 1-13) yields 0.99 for $\neg A$,
47 1,0 con: which means that I am right on $\neg A$, after all.
48 0,0 pro: Seems I have to give up on A (line 1). Fair enough.

The evidence available is used differently this time. To see this, compare line 37 of the present dispute with lines 39-40 of the previous dispute. □

Next, we move on to a critical case.

Case 9.14 (Accrual of dependent reasons) Consider

Proposition	DOS	Proposition	DOS
C	1.00	C –(0.91)→ $\neg A$	1.00
D	1.00	D –(0.93)→ $\neg A$	1.00
		C, D –(0.97)→ A	1.00

C –(0.91)→ $\neg A$ and D –(0.93)→ $\neg A$ accrue with strength $0.91 + 0.93 - 0.91 * 0.93 = 0.99$ so in principle could defeat C, D –(0.97)→ A. However, the first two rules depend on the third rule, because the third rule is more specific than the first two rules.

1	0,0 pro:	I claim that A holds.
2	1,0 con:	Why?
3	0,0 pro:	Because it follows from C and D with strength 0.97,
4	0,0 pro:	and I claim C and D. In particular,
5	0,1 pro:	I claim that C holds.
⋮	⋮	⋮
13	0,1 pro:	That leaves me with C, obviously.
14	0,1 pro:	I claim that D holds.
⋮	⋮	⋮
22	0,1 pro:	That leaves me with D, obviously.
23	0,0 pro:	And via C and D my A is supported 0.97
24	1,0 con:	Sure, but now this:
25	1,0 con:	I claim that $\neg A$ holds.
26	2,0 pro:	Why?
27	1,0 con:	Because it follows from C. But I see you have
28	1,0 con:	a more specific reason for the opposite at line 3,
29	1,0 con:	which prevents me from supporting $\neg A$ with C.
30	1,0 con:	Moreover, to further underpin what I said at 25,
31	1,0 con:	I claim that $\neg A$ holds
32	1,0 con:	because it follows from D. But I see you have
33	1,0 con:	a more specific reason for the opposite at line 3,
34	1,0 con:	which prevents me from supporting $\neg A$ with D.
35	1,0 con:	Um... upon closer inspection
36	1,0 con:	I see I have no grounds for $\neg A$.
37	0,0 pro:	That leaves me with A, obviously.

We see that C –(0.91)→ $\neg A$ and D –(0.93)→ $\neg A$ are pre-empted by C, D –(0.97)→ A, as it should. □

Finally, I conclude the case parade with a reworked example from [35]. This is done to offer the reader an impression of how a localist type of dispute is conducted in situations that are more complex than the basic cases discussed above. The dispute can well be followed without further contextual information. For completeness' sake its background is included as an appendix.

Case 9.15 (Flood warning) In this example, two experts discuss whether a flood warning must be issued to the residents of a little town near the bank of a large river.

1	0,0 pro:	I claim that "flood_warning" holds.

2 1,0 con: Why?
3 0,0 pro: Because it follows from “higher_change”, “normal_level” and “heavy_rain” with strength 0.9, and
4 0,0 pro: I claim “higher_change”, “normal_level” and “heavy_rain”. In particular,
5 0,1 pro: I claim that “higher_change” holds.
6 1,1 con: Why?
7 0,1 pro: Because it follows from “heavy_upstream” with strength 0.8, and
8 0,2 pro: I claim that “heavy_upstream” holds.
⋮ ⋮ ⋮ [fruitless attempt of con to refute “heavy_upstream”]
28 0,2 pro: That leaves me with “heavy_upstream”, obviously.
29 0,1 pro: And via “heavy_upstream” my “higher_change” is supported 0.8
⋮ ⋮ ⋮ [pro establishes a second necessary condition:]
367 6,7 pro: That leaves me with “higher_change”, obviously.
368 6,6 pro: And via “higher_change” my “not lower_change” is supported 0.8
369 6,6 pro: Combining support for “not lower_change” (at 352 and 359) makes 0.8 in total.
⋮ ⋮ ⋮ [con has its occasional successes, too:]
681 3,5 con: That leaves me with “height_16”, obviously.
682 3,4 con: And via “height_16” my “normal_level” is supported 0.8
⋮ ⋮ ⋮ [deepest level:]
1078 7,6 con: I claim that “light_rain” holds.
1079 8,6 pro: Why?
1080 7,6 con: Don’t you remember? I already argued this earlier on:
1081 7,6 con: At line 1051 I concluded that “light_rain” is supported 0.3
1082 7,6 con: Nevertheless, I grant that earlier on (line 1076), you were able
1083 7,6 con: to support “not light_rain” more strongly. Therefore I drop “light_rain”.
1084 6,6 pro: Combining my support (line 1053-1076) and your countersupport (line 1039-1051)
1085 6,6 pro: yields 0.84 for “not light_rain”, which means that I am right on “not light_rain”.
1086 5,6 con: Seems I have to give up on “light_rain” (line 1039). Fair enough.
⋮ ⋮ ⋮ [the dispute draws to a close, with back references:]
1148 0,0 pro: Because it follows from “higher_change”, “high_level” and “no_rain” with strength 0.8, and
1149 0,0 pro: I claim “higher_change”, “high_level” and “no_rain”. In particular,
1150 0,1 pro: Earlier on we debated “higher_change”. The nett outcome then was 0.8,
1151 0,1 pro: as it will be this time.
1152 0,1 pro: Earlier on we debated “high_level”. The nett outcome then was 0.8,
1153 0,1 pro: as it will be this time.

1154 0,1 pro: Earlier on we debated "no_rain". The nett outcome then was 0.8,
1155 0,1 pro: as it will be this time.
1156 0,0 pro: And via "higher_change", "high_level" and "no_rain" my "flood_warning" is supported 0.8
⋮ ⋮ ⋮ [pro gathers his conclusive support on "flood_warning"]
1173 0,0 pro: Combining support for "flood_warning" (at 3, 1148, 1157 and 1165) makes 0.8 in total.
⋮ ⋮ ⋮
1212 1,0 con: I admit that none of the 6 reasons for "not flood_warning" (line 1175)
1213 1,0 con: provided any support for "not flood_warning".
1214 1,0 con: Having tried "not flood_warning", I admit that, earlier on (line 1173), you were
1215 1,0 con: able to establish support for "flood_warning" (0.8). Therefore I drop "not flood_warning".
1216 0,0 pro: That leaves me with "flood_warning", obviously.

What is worth mentioning here is that, from line 1148 on, almost all moves are back references to earlier results, indicating that PRO uses "old" results to close pending attempts of CON to block PRO's establishment of "flood_warning". Without back references the algorithm would be simpler, but its execution would take (much) more time. Thus, in general there seems to be a tradeoff between the complexity of the procedure and the length of the debate. This was already noted by [56]. □

9.5 Relation with Bayesian belief networks

Although the work presented in this chapter fits within the school of dialogical logics, it lies closer to research in Bayesian Belief Nets (BBNs) [79] and Qualitative Probabilistic Networks (QPNs) [127] than other work in dialogical logics, which is due to its local character. For example, in logic-based (including dialectic) approaches, it may be enough to specify that, e.g., $A, B \mathrel{-\!(0,98)\!\rightarrow} C$ while saying nothing about related rules (Equation 9.10). BBNs require considerable more input, however. With BBNs, however, the problem is that if one has chosen to say something about the implicational strength between A, B and C, one is forced to specify x, y and z in

$$\begin{array}{rr} A, \neg B \mathrel{-\!(x)\!\rightarrow} C, & A, \neg B \mathrel{-\!(1\text{-}x)\!\rightarrow} \neg C, \\ \neg A, B \mathrel{-\!(y)\!\rightarrow} C, & \neg A, B \mathrel{-\!(1\text{-}y)\!\rightarrow} \neg C, \\ \neg A, \neg B \mathrel{-\!(z)\!\rightarrow} C, & \neg A, \neg B \mathrel{-\!(1\text{-}z)\!\rightarrow} \neg C. \end{array} \tag{9.10}$$

An infamous disadvantage of BBNs is that too many rules, or rule strengths, must be specified, even if the knowledge-engineer does not have the faintest idea what the corresponding numbers should be like. Machine learning techniques (Chapter 11 on page 217) partially cure this problem, but if the numbers do not exist, cannot be acquired, or cannot be provided for, then even the most sophisticated machine learning techniques will fail to bring them out.

BBN theory also has other solutions to the problem of dealing with missing numerical data. These solutions are known under the headers *Noisy-OR*, *Noisy-AND*, and *divorcing*. They interpolate x, y, and z based on information that certain rules for C accrue, and certain other rules for C explain each other away. *Noisy-OR*, *Noisy-AND*, and *divorcing* are solution concepts that head in the direction of computational dialectics, in the sense that they are no longer probabilistically sound methods of inference.

QPNs are an answer to the number-acquisition problem, in the sense that implicational strengths are replaced by signed links (+ or −), and accrual is replaced by so-called positive (or negative) synergy. Still, a disadvantage of the QPN approach is that its semantics is based on complex logical tables that combine standard logical connectives and connectives for different sorts of non-material implication with connectives for positive and negative synergy. This makes QPN a rather baroque attempt to capture simple forms of commonsense reasoning. Although the latter may be an unjust point of critique, the work presented here might help to provide additional insight in QPNs, among others by looking at it from a dialogical perspective. In general it might bring together work in dialogue logics and the more technical approaches such as QPNs.

9.6 Benchmark problems

In this section, we list some of the benchmark problems that appeared in the literature thus far.

Tweety

Strict defeat (traditional)

Penguins are birds
Birds tend to fly
Penguins don't fly
Tweety is a penguin

Does Tweety fly?

Nixon diamond

Ambiguity (traditional)

Quakers tend to be pacifists
Republicans tend to be non-pacifists
Nixon is a republican quaker

Is Nixon a pacifist?

Nixon's political motivation

Cases (Ginsberg)

Republicans tend to be hawks
Quakers tend to be doves
Nobody is both a hawk and a dove
Hawks tend to be politically motivated
Doves tend to be politically motivated
Nixon is a republican quaker

Is Nixon politically motivated?

Nixon's anti-militarism

Cascaded ambiguity (Touretzky)

Republicians tend to be non-pacifist and football fans
Quakers tend to be pacifist
Pacifists tend to be anti-military
Football fans tend to be non-anti-military
Nixon is a republican quaker

Ambiguous?

Royal African Elephants

Locus of preemption (Sandewall)

Elephants tend to be grey
Royal elephants tend to be non-gray
Clyde is a royal elephant and an african elephant

Is clyde non-gray?

Garfield and People

Irrelevance (Baker)

Cats tend to be aloof
Aloofness tends to indicate dislike of people
Cats tend to like people
Garfield is a cat

Does Garfield like people?

University Students

Strong specificity (Delgrande)

Adults tend to be employed
University students tend to be unemployed
University students tend to be adults
Fred is a university student and an adult

Is Fred unemployed?

Adults under 22

(Geffner-Pearl)

Adults tend to be employed
University students tend to be unemployed
University students tend to be adults
Adults under 22 tend to be university students
Tom is an adult under 22

Is Tom unemployed?

Bankrupt Conservatives

(Henry Prakken)

Conservatives tend to be selfish
Poor people tend to be unselfish
Bankrupt conservatives tend to be poor
Rob is a bankrupt conservative

Is Rob selfish?

Dancers and Ballerinas

Specificity versus directness (Loui)

Dancers tend not to be ballerinas
Dancers tend to be graceful
Graceful dancers tend to be ballerinas
Noemi is a dancer

Ambiguous?

Gullible Citizens

Relations (Touretzky)

Citizens tend to dislike crooks
Gullible citizens tend to like elected crooks
Fred is a gullible citizen
Dick is an elected crook

Is it the case that Fred likes Dick?

Unrefined Big Blocks

Monotonic chaining, Closure (Poole, Loui)

Noisy cars tend to be highly revved
Highly revved cars tend to be small
Noisy cars tend to have big block engines
Wide-tyred cars tend to handle well
Good handling tends to indicate refinement
Wide-tyred cars tend to be unrefined cars
All big block unrefined cars are muscle cars
All small block refined cars are non-muscle cars
Guido's car is noisy with wide tyres

Is Guido's car a muscle car?

Say Randy flies

Specificity without contradiction (Poole)

The usual response for "bird" is "flies"
The usual response for "emu" is "runs"
Things at the emu farms tend to be emus
If soothsaying and the response for an x is y then say(x, y)
Presume soothsaying
Randy is at the emu farm

Say "Randy runs"?

Lottery Paradox

Probablistic inconsistency, collective defeat (Kyburg, Pollock)

Probably, ticket 1 will not win
⋮
Probably, ticket 100 will not win
Ticket 1 or ... or ticket 100 will win

Any conclusion?

Billboard's hot 100

Specificity versus cases, Simpson's paradox (Loui, Neufeld, Poole)

Records on the dance chart tend to be on the Hot100 chart
Records on the soul chart tend to be on the Hot100 chart
⋮
Records on the Aussie40 chart tend to be on the Hot100 chart
Records on one of the dance or soul or ... or Aussie40 chart tend to be *not* on the Hot100 chart
Randy bought a record on the dance or soul or ... or Aussie40 chart

Is Randy's record on the Hot100 chart?

Doctors are Medical?

Strong specificity versus specificity: statistics and implicature (Loui)

Doctors tend to be medical
Most Doctors have the PhD or JD
Persons with the PhD or JD tend to be non-medical
Fred as a PhD or JD

Is Fred non-medical?

Backwater CS PhD's

(Cross, Nute)

Backwater teachers tend to be poor
Backwater Cs teachers tend to have CS PhD's
CS PhD's tend to be rich
Legal pro-bono people tend to be poor
Legal pro-bono people tend to have JD's
Legal pro-bono JD's tend to be rich
Fread teaches CS at Backwater and does legal pro-bono work

Ambiguous?

Tom's waking

Hidden reasons for presumptions (Geffner)

People usually wake before noon
Tom usually wakes after noon

Does Tom wake in the afternoon?

Genetically altered pigs

Evidence versus directness (Loui)

Pigs tend to be fat and flabby
Genetically altered pigs tend to be non-fat
Non-fat animals tend to be non-flabby
Snowball is a genetically altered pig

Is snowbal non-flabby?

Hernandez tries to homer

Argument versus subargument (Loui)

Good hitters tend not to try to homer
Good hitters which have homered lately tend to try to homer
Trying to homer tends to result in not getting a hit
Batters who have homered lately and aren't trying to homer tend to get good hits
Hernandez is a good hitter who has homered lately

Does Hernandez fail to get a hit?

Hermann's Pennsylvania Dutch

Mixing strict and defeasible inheritance (Horty, Thomason)

Native speakers of German tend not to be born in America
All native speakers of Pennsylvanian Dutch are native speakers of German
Native speakers of Pennsylvanian Dutch tend to be born in Pennsylvania
Hermann is a native speaker of Pennsylvanian Dutch

Is Hermann born in America?

Fiats are fast?

Indirect strong specificity (Loui)

Fiats are North Italian cars
North Italian cars tend to be fast
Sports cars tend to be fast
Fiats tend to be slow
Fred's Fiat is a sports car

Ambiguous?

Nice guy lawyers

Exception to class-based strong specificity (Ginsberg)

Lawyers tend to be republican and nice
Republicans tend to be conservative
Conservatives tend to be not-nice
Dave is a conservative republican lawyer
Fred is a conservative non-republican lawyer

Both ambiguous?

Party for friends

Implicature (Geffner, Myers, Baker)

By default, all my friends will show
Tom and Mary are my friends
Tom says that if Mary shows, he won't

Does Tom fail to show?

Chapter 10

Abduction

Both deduction and argumentation have in common that they both apply general rules to particular observations with the purpose to support existing claims (e.g., legal reasoning) or to produce new conclusions (e.g., reasoning about action).

Both forms of reasoning also have in common that they presuppose the existence of rules, and that rules of inference are applied in a direction that goes from antecedent to consequent. Deduction as well as argumentation are modes of reasoning that "follow the rules," in the most literal sense of the word.

Figure 10.1: Charles Sanders Peirce (1839-1914).

The aim of the next few chapters is to study two other forms of reasoning, namely, abduction and induction. Like deduction, abductive reasoning presupposes the existence of rules and, also like in deduction is rule-following (rather than rule-creating). With abduction, rules are followed, be it in "the wrong direction," i.e., from conclusion to premises. The third mode of reasoning, induction, differs from deduction and abduction in the sense that it does not presuppose the existence of rules. Rather, induction is about creating rules of inference, based on background knowledge in the form of facts and cases.

This chapter is about abduction. We first sketch a global picture, and then focus on abduction. First, as a means to explanatory reasoning, then as a more general tool to search in explanatory dialectics.

Peirce's triple

The American lawyer, logician and philosopher Charles Sanders Peirce proposed, besides deduction and induction, a third mode of reasoning, viz. *abduction*. The idea is that, in many patterns of reasoning, you have a concrete case at hand (sometimes called: "the facts"), a general principle that abstracts from concrete cases, i.e. a rule, and a conclusion in the form of an observation or result, that would follow from the concrete case and the general principle. In

some cases, we have a case and a rule, so that we may infer with a certain amount of certainty a conclusion. But in other situations, there is only a rule and an observation, or two observations in succession. In such situations, the third component can be inferred from the other two with abductive or inductive inference, respectively.

Peirce's argument went by means of the following example [CP, 2.623]:

Deduction		All the beans from this bag are white.	(rule)
		These beans are from this bag.	(case)
	So	These beans are white.	(result)
Induction		These beans are from this bag.	(case)
		These beans are white.	(result)
	So	All the beans from this bag are white.	(rule)
Abduction		All the beans from this bag are white.	(rule)
		These beans are white.	(result)
	So	These beans are from this bag.	(case)

Of these, deduction is the only reasoning which is completely certain, inferring its result as a necessary conclusion. Induction produces a rule validated only in the long run [CP, 5.170], and abduction merely suggests that something may be the case [CP, 5.171]. Later on, Peirce proposed these types of reasoning as the stages composing a method for logical inquiry, of which abduction is the beginning:

> "From its [abductive] suggestion deduction can draw a prediction which can be tested by induction." [CP, 5.171].

Peirce's triangle is an attractive starting point to make a distinction between deduction, induction and abduction: deduction is rule-following, induction is rule-making, and abduction is applying rules in the "wrong" (that is, opposite) direction.

Still, despite its conceptual simplicity and elegance, it is fair to state that Peirce's scheme does not square with recent developments in logic and artificial intelligence. One complication, for instance, is that it neglects the fact that deduction is only one out of many "modes of reasoning," namely, that mode in which rules are absolutely certain. In the previous chapter, however, we have seen that it is also possible to reason with rules of inference that are not absolutely certain. Peirce acknowledged this type of inference as well, and called it *ampliative reasoning*. Ampliative reasoning is the kind of reasoning in which more is concluded than what is contained in the premises, and hence amplify the scope of our beliefs. Ampliative arguments are not deductively valid, but may yield credible conclusions. Thus, although Peirce certainly acknowledged the existence of non-deductive forms of reasoning, he apparently did not incorporated it in his deduction / induction / abduction scheme.

Both abduction and induction may be formulated with the help of deductive means (such as propositional logic and predicate logic), but never turn into forms of deductive inference themselves (Fig. 10.2 on the next page).

An additional complication in the distinction between deduction, induction, and abduction is that both abduction and induction may be translated into heuristic rules with which it is possible to reason forward, from premises to conclusion (Fig. 10.2 on the facing page, two small ellipses). Thus, it is possible to mold abductive (or inductive) logics into the from-antecedent-to-consequent paradigm, which brings them within the category of logics in which one is supposed to follow the rules. This phenomenon will be discussed in the next section.

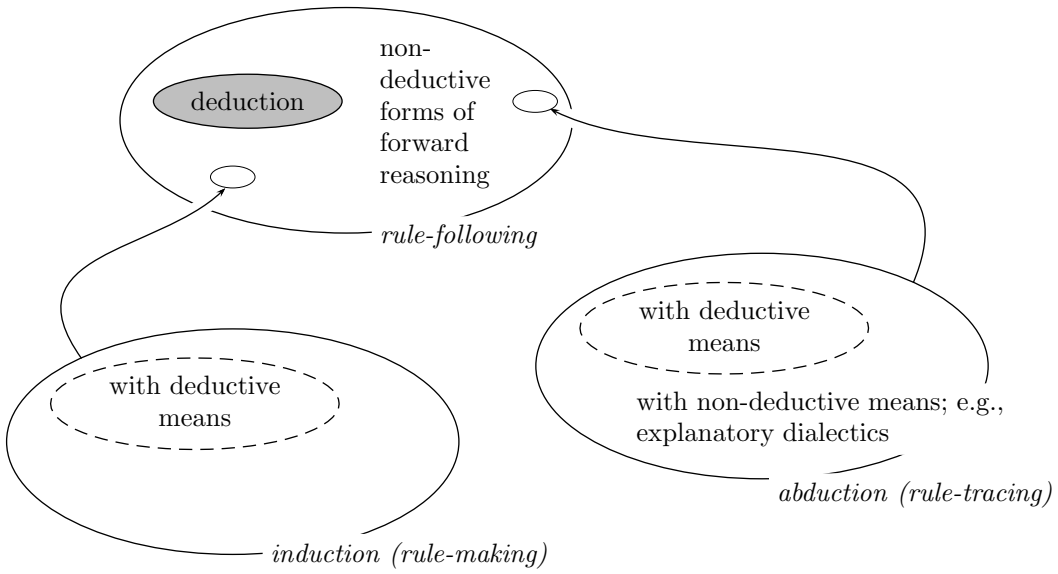


Figure 10.2: Contemporary version of Peirce's triangle.

10.1 Explanatory reasoning

When surprising events occur, people naturally try to generate explanations of them. Such explanations usually involve hypothesizing causes that have the events as effects. Accordingly, the most important application of abduction is *explanatory reasoning*.

Often, the terms "abduction" and "explanatory reasoning" are used interchangeably, but there is a distinction.

- Explanatory reasoning is reasoning in which explanations are produced by logical means. Any logical method could be used for this purpose, including abduction.
- Abduction is the type of reasoning in which rules of inference are traversed backwards, "in the wrong direction". Abduction can be used for different purposes, including the purpose to produce explanations.
- If rules represent causal relations between propositions in such a way that causality "flows" from antecedent to consequent, and if explanations are produced by means of abductive inference, then abduction and explanatory reasoning amount to the same activity.

Reasoning from effects to prior causes is found in many domains, including sociology, legal theory, medicine, science and manufacturing. When friends are acting strange, we conjecture about what might be bothering them; when a crime has been committed, jurors must decide whether the prosecution's case gives a convincing explanation of the evidence. In medical diagnosis, a physician tries to decide what disease or diseases produced them, given a set of symptoms. In scientific theory evaluation, scientists seek an acceptable theory to explain experimental evidence. With fault diagnosis in manufacturing, when a piece of equipment breaks down, a trouble shooter tries to determine the cause of the breakdown. Explanations are needed when devices break down which leads to circumstances in which faults must be diagnosed. These patterns of reasoning are reflected in questions for explanation, such as: "why did X break?," "how does X work?," "what if ...?," and "what if not ...?"

With explanatory reasoning, rules represent causal relations between propositions, in such a way

that causality runs in the same direction as the (arrows of the) rules. The following example from medical diagnosis show that this observation is far from trivial and needs to be addressed explicitly.[1]

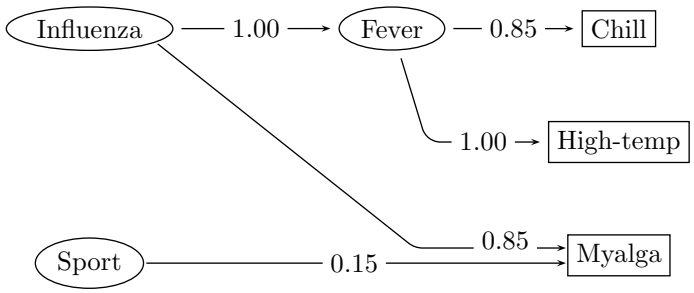


Figure 10.3: Causal relations in medical diagnosis

Example 10.1 (Whether muscle pain is caused by influenza or sporting) Influenza causes fever and influenza often (but not always) causes myalga (painful muscles). Fever causes a high body temperature and often (but not always) makes the patient complain about being cold (chill). Sporting sometimes causes painful muscles (Fig. 10.3). When a friend of yours complains about painful muscles, you may reason backwards through the rules to conclude that this might be caused by influenza or by intensive sporting. Conclusions may also be explained away: when a someone does not complain about having it cold, it becomes more plausible to conclude that he (or she) must have been sporting. □

Causal rules may be contrasted with *heuristic rules*. Consider, for example, the symptom myalga in Fig. 10.3. The fact that this symptom can be caused by influenza as well as sport, can be represented by causal rules (Fig. 10.4, left) or by heuristic rules (Fig. 10.4, right). The first

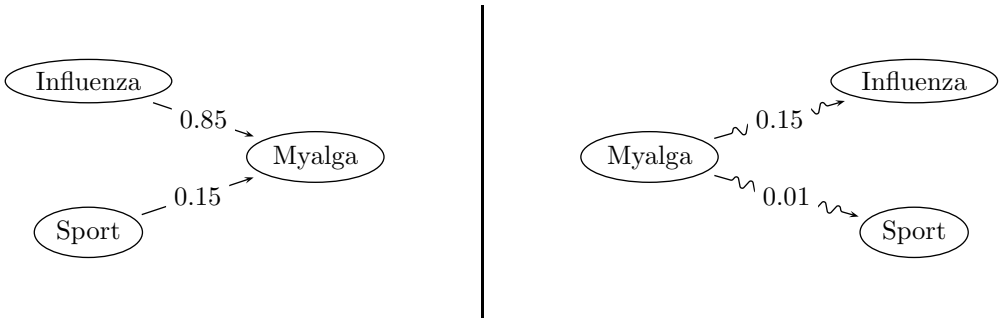


Figure 10.4: Causal (left), vs. heuristic representation (right).

representation respects causality, in the sense that the rules run from left to right, and so does causality. The second representation does not respect causality, but uses heuristic rules (some would say: heuristics) to infer possible causes. With the first representation, painful muscles may be explained by means of abduction, that is, by reasoning backwards. With the second representation, painful muscles may be explained by reasoning forwards.

[1] This example is inspired on (but not identical to) a diagram displayed in [59].

1. Causal and heuristic rules are rules of a different type and may not be mixed. If we *do* mix them, we may create a so-called *belief pump*. A belief pump is a cycle of rules with which it is possible to inflate the inferred degree of support of propositions artificially. This phenomenon is already suggested in Fig. 10.4: if you have sported intensively, I may use a causal rule to increase my belief that you are suffering from myalga. I then use a heuristic rule to increase my belief that you have sported, after which I use a causal rule to increase my belief that you are suffering from myalga, and so forth. More on the inflation of belief on page 167.
2. If the rule strengths in Fig. 10.4 were probabilities, then some probabilities would follow from others with Bayes' rule. E.g.,

$$Pr(\text{Sport} \mid \text{Myalga}) = \frac{Pr(\text{Myalga} \mid \text{Sport})Pr(\text{Sport})}{Pr(\text{Myalga})} \tag{10.1}$$

   Thus, if we know $Pr(\text{Myalga})$ we could compute $Pr(\text{Sport})$, and if we know $Pr(\text{Sport})$, we could compute $Pr(\text{Myalga})$. However, the numbers here are not supposed to stand for probabilities. There are merely used to indicate the strength of a rule of inference. Thus, we may not apply principles of probability without further notice.

10.2 The hypothetico-deductive method

In practice, many things pass as explanations. Somewhere around 1930, the Austrian born British philosopher Karl Popper argued that it is impossible to explain something by means of induction, because no amount of evidence assures us that contrary evidence will not be found.

Instead, he proposed a scientific methodology which he termed the hypothetico-deductive method of science [87, 86].
"Hypothetico" means "based on hypotheses".
For Popper, the reliability of an explanation, or scientific theory, is determined by the ability to develop and clearly state hypotheses that can be tested in some systematic way. Since then, Popper's theory has always been extremely influential in logic and the philosophy of science, not in the last place because many philosophers of science think that the hypothetico-deductive method is actually a highly idealized and unfaithful representation of scientific practice.

Figure 10.5: Karl Popper (1902-1994).

Like abduction or "inference to the best explanation" the HDM is a form of inference that follows a pattern like this:

D is a collection of data (findings, observations),

H would, if true, explain D,

No other hypothesis explains D as well as H does.

Therefore, H is probably the case.

where D and H are sets of logical formulas.
With a little more nuance, it goes like this:

B is a (typically large) repository of rules and facts that we assume to be true; this collection is referred to as *background knowledge*.

D is a (small) collection of data (findings, observations) that we accept as true, but in addition would like to have explained,

H is a collection of hypotheses that would, if true, help B to explain D,

No other set of hypothesis explains D as well as H does.

Therefore, H is probably the case.

It is possible to make this scheme more realistic
by further refining it, for example, by not making an distinction a priori between B, H, and D. However, the current level of abstraction suffices for our purposes.

Not every set H from which D follows counts as an explanation:

1. *Completeness.* H must explain (cover) all the data. More specifically, there must exist proofs, or arguments, based on the combination of sentences in B and H that explain D.
2. *Likelihood.* H should make D more likely relative to B, than D is likely relative to B of itself. (In probability theory we would say $Pr(D|B,H) > Pr(D|B)$.) In particular, B should not logically imply D. Otherwise, any explanation H would be useless, as it fails to raise, and sometimes even lowers, the plausibility of D.
3. *Consistency.* From B and H we may not derive statements that we know are highly unlikely. In particular, $B \cup H$ should be consistent. Otherwise, $B \cup H$ would represent an implausible state of affairs. For example, if $b \in B$, $\text{DOS}(b) = 0.98$ and there is an argument a based on $B \cup H$ such that a supports $\neg b$ with a DOS of 0.98, then H contradicts some element in the background knowledge. This contradiction makes H implausible.
4. *Minimality.* H must be as small as possible. More specifically, if H' is another collection of hypotheses such that $H \subset H'$ (where "$\subset$" means proper subset), then a necessary condition for H' to be an explanation of D, is that D is more likely relative to $B \cup H'$ than D is likely relative to $B \cup H$. Otherwise, H' is considered to water down H with superfluous hypotheses. Notice that $Pr(D|B,H') > Pr(D|B,H)$ is not a sufficient condition since H' might violate one of the conditions above.
5. *Generality.* H must be such that if H' is another collection of hypotheses such that H assumes as much as H' ($H \vdash H'$), then a necessary condition for H to be an explanation of D, is that D must be as likely relative to $B \cup H$ as D is likely relative to $B \cup H'$. (Otherwise, H would make make unnecessary assumptions, compared to H'.)

   Generality is a logical variant of the minimality criterion. Nothing is lost by assuming that all least presumptive explanations are minimal. (Exercise 3 on page 204.)

The plausibility, credibility, or strength of an abductive conclusion will, or should, in general depend on several factors, such as:

1. The credibility of H (independent of alternative collections of hypotheses).
2. How decisively H surpasses the alternatives.
3. How thorough the search was for alternative explanations, and
4. Pragmatic considerations, such as:
   - The costs of being wrong and the benefits of being right.
   - How strong the need is to come to a conclusion at all, especially considering the possibility of seeking further evidence before deciding.

In the next sections, we will formulate algorithms that comply with the above rationality postulates, and uses specific heuristics to measure the plausibility of sets of hypotheses.

Propositional abduction

In this section, we will develop an abductive algorithm in the framework of propositional logic that is able to solve cases such as displayed in Fig. 10.3 on page 198. More in general, the abductive algorithm that we will present here is able to deal with problems of the type $\mathcal{C} = \langle B, H, D\rangle$ where B, H and D are finite sets of propositions with the meaning as explained in Sec. 10.2 on page 199. The goal, then, is to find all best explanations $H' \subseteq H$. (This is a set of subsets of H.)

In the previous section it was explained why explanations in propositional logic, i.e., proofs that connect observations back to hypotheses, can be identified with the hypotheses themselves. Hence, the abductive algorithm that we will deal with does not focus on proofs (or arguments), but rather on the existence of proofs, i.e., on provability.

The algorithm consists of three steps. The first step is to find a weakest formula X (unrelated to H) such that B and X together explain D: $B, X \vdash D$. The second step is to find subsets of H that imply X. This is done by "growing" the empty set into different candidate explanations with elements of H, adding one hypothesis at a time, until these candidate-explanations have been extended to sets that become either worthless (i.e., inconsistent) or useful (i.e., explain D). The third step, then, is to remove explanations that imply other candidate-explanations, since we are searching for explanations that are least presumptive (Condition 5 on the facing page).

We now look at the three steps in somewhat more detail. Step 1 is to find a weakest (or most general) formula X, unrelated to H, such that $B, X \vdash D$. Such a formula X can be found by trying to refute $B, X \circ D$ in a refutation tree, where X is unknown. The branches that remain open indicate what is missing in X to make $B \cup X$ explain D, and can be closed by putting the missing literals in X.

Example 10.2 Suppose

$$B = \{p \supset q, (p \wedge q) \supset r\} \text{ and } D = \{r\}.$$

We are interested in the most general clause set X that bridges the gap between background knowledge B and data D. To produce X, we construct a refutation tree for $B, X \vdash D$, where X acts like a variable. If $X = \varnothing$, the refutation would succeed (Fig. 10.6 on the next page).

This refutation succeeds because

1. The letters p and r do not occur on the LHS of (1);
2. The letters p, q and r do not occur on the LHS of (2);
3. The letters p and r do not occur on the LHS, and the letter q does not occur on the RHS of (3).

To let the refutation fail—that is, to establish $B, X \vdash D$—it is necessary and sufficient to let $p \vee r \in X$, $p \vee q \vee r \in X$, and $p \vee \neg q \vee r \in X$, respectively. Hence,
$X = \{p \vee r, p \vee q \vee r, p \vee \neg q \vee r\}$. Thus, $X = (p \vee r) \wedge (p \vee q \vee r) \wedge (p \vee \neg q \vee r) \equiv p \vee r$ is the weakest formula such that $B, X \vdash D$. □

If B is inconsistent, then X remains empty. Hence the refutation of $B, X \circ D$ can at the same time be used to check B's consistency. Another way to see this is to observe that a consistency check of B amounts to a refutation of $B \circ$ which is a subtree of the refutation of $B, \varnothing \circ D$. B and D can now be dropped, since our goal is reduced to finding subsets $H' \subseteq H$ such that $H' \vdash X$. This is done in Step 2.

Step 2 of the algorithm is to define three sets, namely, *OPEN*, *NOGOODS*, and *EXPLANATIONS*, that contain subsets of H. (So you will have to think about a datastructure that is able to represent sets of sets.)

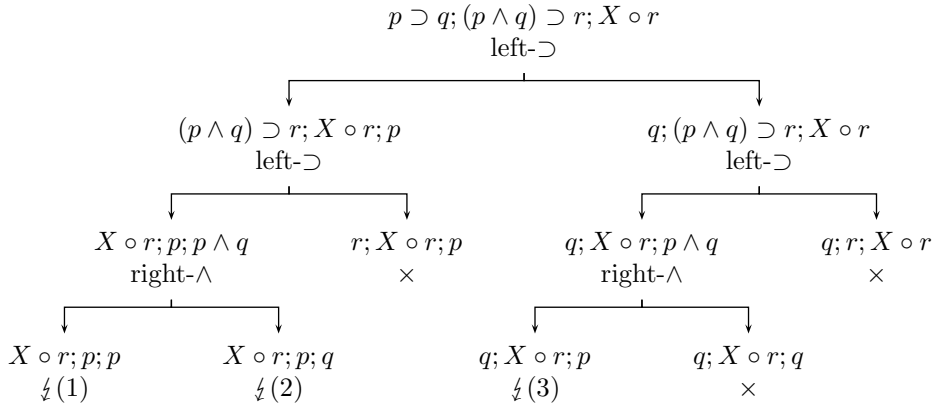


Figure 10.6: A refutation tree to find a most general explanation X

- *OPEN* is initialized to $\{\varnothing\}$ and is used to store candidate-explanations (subsets of H) that fail to cover X (Condition 1 on page 200). Elements of *OPEN* are called open, because it is not clear yet whether these sets are part of an explanation. They must be extended (i.e., specialized) to cover X.
- *NOGOODS* is initialized to $\varnothing$ and is used to store elements that were removed from *OPEN* because a specialization made them inconsistent. Since the abductive algorithm tries to find all explanations, *NOGOODS* eventually contains all minimally inconsistent subsets of H.
- *EXPLANATIONS* is initialized to $\varnothing$ and is used to store elements that were removed from *OPEN* because a specialization made them entail X. *EXPLANATIONS* eventually contains all explanations of X. Thus, the set *EXPLANATIONS* eventually contains all explanations of D, given B.

The idea is to specialize elements of *OPEN* and to divide them among *NOGOODS* and *EXPLANATIONS* until *OPEN* is empty. Specialization is done as follows: take one element O from *OPEN* and extend it in all possible ways with elements from H. This will given you $|H| - |O|$ one-step specializations $S_1, \ldots, S_{|H|-|O|}$, one for each $h \in H \backslash O$. Examine each S_i. If S_i is either in *OPEN*, *NOGOODS* or *EXPLANATIONS*, then drop it, since if it is in *NOGOODS* or *EXPLANATIONS* we have seen it, and if it is in *OPEN* it will be examined later. (Saves computation.) If S_i is inconsistent, put it in *NOGOODS*; if S_i implies X, put it in *EXPLANATIONS*. Else, put S_i back in *OPEN*, since it needs to be specialized further. This basically, is the entire algorithm (cf. on page 205).

Example 10.3 Let $\mathcal{A} = \langle B, H, D \rangle$ be a propositional abduction problem with background knowledge $B = \varnothing$, data $D = \{r\}$, and a pool of possible hypotheses $H = \{h_1, h_2, h_3, h_4\}$, such that

$h_1 = p \wedge \neg s$,
$h_2 = p \supset (r \wedge s)$,
$h_3 = q \supset (r \wedge \neg s)$, and
$h_4 = p \wedge q$.

Since B and H are such that $B, H \nvdash D$, it makes sense to ask which subsets of H explain D.

$\mathcal{A}$ generates a hierarchy (officially: a *lattice*) of candidate-explanations, as indicated in Figure 10.7. An example of such a candidate-explanation is $E = \{h_1, h_3\}$.

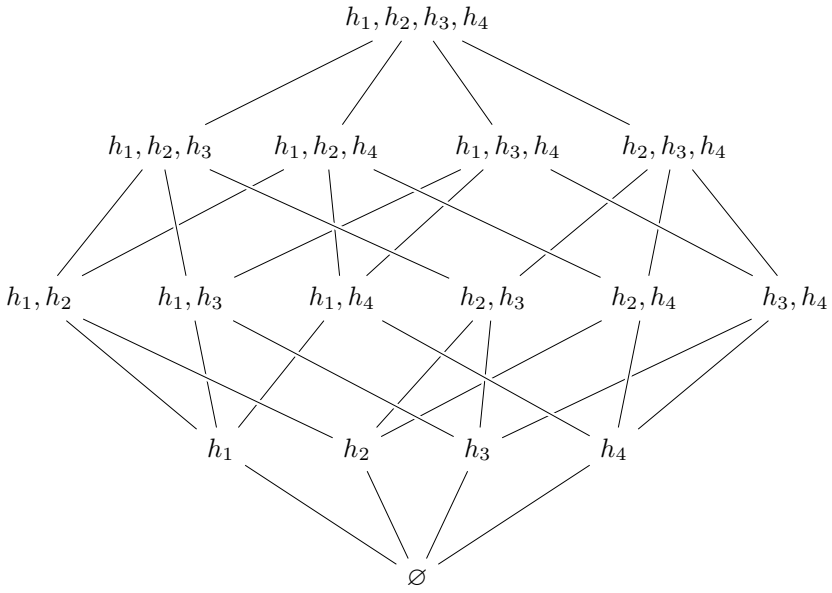


Figure 10.7: Hierarchy of candidate-explanations, in the form of sets of hypotheses

Not all candidate-explanations are explanations: $\{h_1, h_2\}$ and $\{h_2, h_3, h_4\}$ are inconsistent with B, while $\{h_2, h_4\}$, $\{h_3, h_4\}$, and $\{h_1, h_3, h_4\}$ are combinations that are consistent with B and cover D. Figure 10.8 on the following page is made with the intention to depict this situation. If we grow $E = \varnothing$ by adding elements of H one at a time, we encounter the nogoods $\{h_1, h_2\}$ and $\{h_2, h_3, h_4\}$. (Compare Figure 10.8.) Further, $\{h_2, h_4\}$, $\{h_3, h_4\}$, and $\{h_1, h_3, h_4\}$ are consistent sets of hypotheses that cover r. But since $\{h_1, h_3, h_4\}$ is not minimal, it does not qualify as an explanation. The remaining sets $\{h_2, h_4\}$, $\{h_3, h_4\}$ do qualify as explanations, because neither set is as presumptive as the other one, i.e., $h_2 \wedge h_4 \nvdash h_3 \wedge h_4$ and $h_3 \wedge h_4 \nvdash h_2 \wedge h_4$. □

Step 3 of the algorithm is to delete explanations that imply other explanations. You can slightly gain in efficiency if the third step is included in the second one. In that way, S is compared with each element of *EXPLANATIONS* before putting it into *EXPLANATIONS* too. The entire algorithm is listed on page 205.

Computational complexity

The algorithm for propositional abduction that is presented on page 205 uses two computationally expensive consistency and derivability checks. Two of these checks occur inside a for-loop, which means that they will be executed repeatedly. Consistency and derivability checks work well in cases with a relatively small number of proposition letters. However, from Chapter 2 we know that consistency checks are NP-complete in worst cases. This might become a problem when we want to produce explanations in large scenarios. Using refutation trees to check consistency is usually a bad idea in such cases, so that we will have to reside to consistency checkers that are more efficient. Such consistency checkers are discussed in Chapter 5, p. 131.

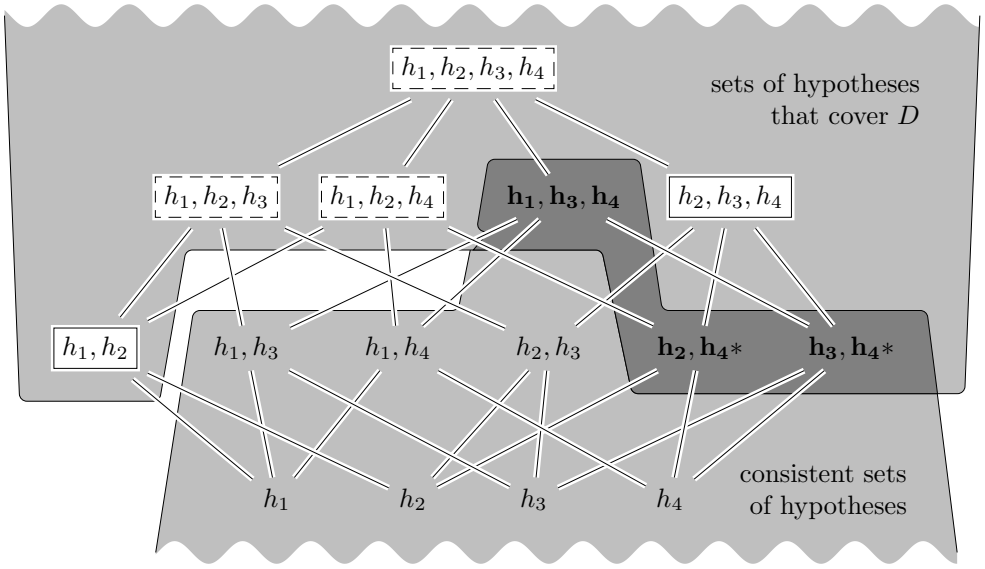


Figure 10.8: Different classes in the hierarchy of candidate-explanations.Lines, boxes and shaded regions have the following meaning:

	meaning
lines	subsets
shaded region below	consistent sets of hypotheses
shaded region above	complete sets of hypotheses
intersection of shaded regions	explanations
starred hypotheses	least presumptive explanations
boxes	minimally inconsistent sets of hypotheses
dashed boxes	inconsistent sets of hypotheses

PROBLEMS (Sec. 10.2)

1. Investigate the meaning of "Ockham's razor". Alternative spellings of the name of the intended person are, besides "Ockham", "Wilhelm von Occam," "W. of Occam," "William of Ockham," or combinations thereof. (Good to know if you do a web search.)

   How does it relate to the problem of finding good explanations?

2. Consider the following background information: fire causes smoke and causes the alarm to go off. The alarm may also go off because someone has tinkered with it. If the alarm goes off, the building is evacuated.

   (a) Formulate the above information in the language of propositional logic. (Solution.)

   (b) Suppose the following data is given: the building is evacuated but there is no smoke. Formulate a weakest proposition that, together with the background information, covers the data.

   (c) Suppose the only two possible hypothesis for an evacuation are fire and the alarm going off. Give all possible explanations for the data.

3. Prove that all least presumptive explanations are minimal. (Solution.)

Algorithm 5 Abduction in propositional logic

Input: A list of propositional formulas, B, representing the background knowledge, a list of candidate hypotheses, H, and a list of observations, D, that must be explained

let X be the weakest formula that "fills the gap" between B and D, i.e., let X be the weakest formula such that $B, X \vdash D$ # use a refutation tree to obtain the unknown X; works also if B is inconsistent; B and D can now be dropped
let $NOGOODS = EXPLANATIONS = \varnothing$
let $OPEN = \{\varnothing\}$ # ordered set, i.e. list in which every element occurs at most once
while *OPEN* has elements **do**
  let O be the first element from *OPEN* (thus, the latter drops one element)
  for each specialization S of O **do**
    if S is in $NOGOODS \cup OPEN \cup EXPLANATIONS$ **then**
      next S # we have seen it already
    else
      check whether S is consistent and entails X
      # both checks are expensive but can be can be combined in one refutation for which X is already in CNF
      if S is inconsistent **then**
        put S in *NOGOODS*
      else if S entails X **then** # S is minimal
        for each E in *EXPLANATIONS* **do** # this for-loop ensures that only the least presumptive explanations are kept
          next S **if** E is as weak as S # expensive test
          remove E from *EXPLANATIONS* **if** S is weaker than E
        put S in *EXPLANATIONS*
      else
        put S in *OPEN* # uniqueness is guaranteed, since we know $S \notin OPEN$
    # next S
  # next O
  # every candidate-explanation is now closed
return *EXPLANATIONS*

10.3 Explanatory dialectics

Propositional abduction possesses at least two major disadvantages. A first disadvantage is that it relies on propositional consistency/derivability checks, and we know that such checks are NP-complete. Thus, propositional abduction is computationally intractable. A second, and probably more important, disadvantage lies on the conceptual level. A drawback of propositional abduction is that it falls within the framework of propositional logic. In propositional logic, statements are either true or false. Facts hold or do not hold and rules of inference either do or do not imply their conclusions with absolute certainty. We have observed in previous chapters that this representation does not correspond to reality. Another disadvantage is that explanations-as-proofs do not tolerate contradictions. If someone or something has found an explanation, it is assumed that this explanation stands once and for all, and cannot be invalidated by new insights. But reality is often different. Sometimes, new incoming data invalidates existing explanations, and enables alternative and more plausible explanations. Explanatory dialectics tries to remedy these shortcomings by applying the principle of abduction in a dialectic context. Whether this approach really works is discussed at the end of this section.

The idea of explanatory dialectics is that explanations are arguments that support observations. Such arguments remain valid until they are refuted by other arguments. These other arguments need not necessarily be explanations, because the task of refutations is to criticize,—not to explain. For example, if a surprising event e occurs, then $\text{DOB}(e) > 0$. This is so, because e is

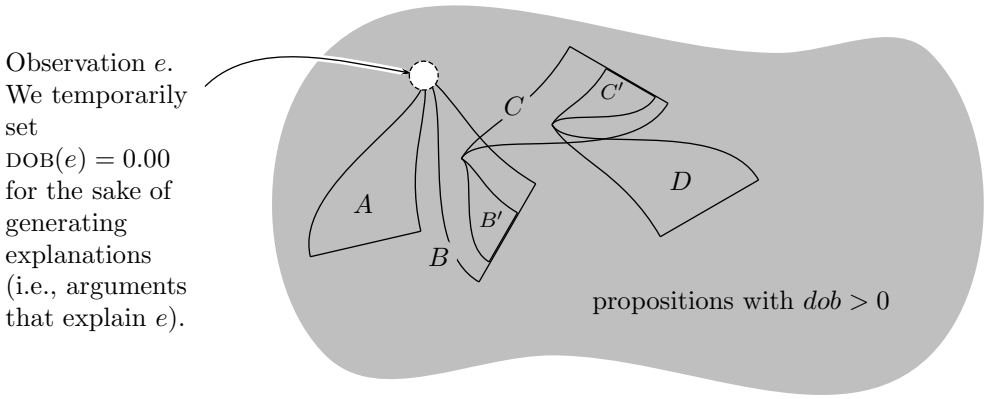


Figure 10.9: Explanatory dialectics

observed. To explain e, we temporarily set DOB(e) = 0 and try to infer support for e through rules and propositions $\neq e$ for which DOS > 0. This idea is depicted in Fig. 10.9: observation e is explained by two arguments, viz. A and B. Explanation B is attacked by argument C, since C interferes with a subargument B' of B. Note that C is not an explanation; it is just an argument. Explanation B is not defeated by C, since there is another argument, D, that defeats a sub-argument C' of C.

An alternative definition

The term "explanation" can be understood in different ways. In a number of frameworks for abduction, notably those frameworks that use an abstract notion of explanation, these interpretations coincide. For other frameworks for abduction, however, notably those frameworks that possess an expressive language, different interpretations of the term "explanation" do have different meanings. Therefore, we first analyse the meaning of the concept of explanation. A glance at the the dictionary[2] shows that the concept "explanation" can be understood in at least three different ways:

1. Statements or claims that explain or make clear. (E.g., "That was supposedly the explanation of her absence.")
2. The act or process of explaining. (E.g., "She had to explain why she was late.")
3. A mutual exposition of terms, meaning, or motives, with a view to adjust a misunderstanding, and reconcile differences.

The first interpretation describes a static notion of explanation, namely, the parts, or material, of which an explanation is constituted. The second interpretation describes a dynamic notion of interpretation, a process under which the static notion of explanation is formed. The third interpretation, finally, describes a social process to smooth out differences between parties to come to an agreement. Automated reasoning is concerned with the first two interpretations, which are of an analytical nature (which means that they are suitable for analysis and precise definition). The dynamic notion of explanation presupposes the static one, so let us define the static one first. For the moment, let us say that an *explanation* is an argument or proof that explains why a certain observation holds, and that an argument *a explains* an observation, or datum, d if a supports d, i.e., if d is the conclusion of a. (In Chapter 8 it is described how

[2]To avoid difficulties with copyright, entries from Collin's Cobuild, The American Heritage, Webster's, and WordNet were combined and edited.

arguments may look like.)

In practice an explanation is often identified with its premises. More precisely, in practice an explanation is often identified with the premises of the argument that is supposed to explain the observation. When this is the case, the reasons or rules that were used to infer the conclusion remain unnoticed, for instance because we have agreed on the fact that intermediate reasoning steps, or reasons, do not have to be mentioned, for example because rules are supposed to be common knowledge. If

$$\begin{array}{lllll} A \leftarrow\!\!(0.76)\!\!- & B \leftarrow\!\!(0.97)\!\!- & D & & \\ & & E \leftarrow\!\!(0.93)\!\!- & I & \\ & C \leftarrow\!\!(0.98)\!\!- & F & & \\ & & G \leftarrow\!\!(0.93)\!\!- & J & \\ & & & K & \\ & & & L & \\ & & H \leftarrow\!\!(0.93)\!\!- & M & \end{array}$$

is an argument explaining A, for instance, then the premises $\{D, I, F, J, K, L, M\}$ are often considered as the explanation of A. Thus, an explanation (i.e. argument) often reduces to a set of statements on the basis of which the argument can be constructed (or reconstructed). A problem with identifying explanations with their premises, however, is that the projection of an argument to premises need not be one-one. It may happen that one set of statements and one observation may enable more than one argument. Here is a simple example.

Example 10.4 Let $E_1 = h \to d$ and $E_2 = h \to x \to d$ be two possible explanations for the observation d. If explanations are identified with their premises, then both E_1 and E_2 are identified with $\{h\}$, so that it is impossible to distinguish between them. In particular, when it is claimed that h explains d, we do not know whether d is explained by way of $h \to d$ or by way of $h \to x \to d$. Later, we will see that this can sometimes be problematic. □

When rules of inference are strict (as is case in propositional logic, for instance) explanations-as-arguments may be identified with explanations-as-premises because, when rules have strength one, arguments are proofs and since proofs with equal premises provide equal support, it does not matter by which proof an observation actually is explained. Thus, deductive explanations may safely be identified with their premises without loss of information.

Example 10.5 Suppose $\textsc{dos}(D) = 0.97$, $\textsc{dos}(F) = 0.94$, $\textsc{dos}(G) = 0.93$. Then

$$\begin{array}{lllll} A \leftarrow\!\!(1.00)\!\!- & B \leftarrow\!\!(1.00)\!\!- & D_{0.97} & & \\ & & E \leftarrow\!\!(1.00)\!\!- & F_{0.94} & \\ & C \leftarrow\!\!(1.00)\!\!- & F_{0.94} & & \\ & & G_{0.93} & & \end{array}$$

and

$$\begin{array}{lllll} A \leftarrow\!\!(1.00)\!\!- & H \leftarrow\!\!(1.00)\!\!- & D_{0.97} & & \\ & & J \leftarrow\!\!(1.00)\!\!- & F_{0.94} & \\ & & & G_{0.93} & \\ & D_{0.97} & & & \end{array}$$

are two deductive explanations of A. For the first argument:

$$\begin{array}{l} \textsc{dos}(E) = \textsc{dos}(F) = 0.94, \\ \textsc{dos}(C) = \min\{\textsc{dos}(F), \textsc{dos}(G)\} = \min\{0.94, 0.93\} = 0.93, \\ \textsc{dos}(B) = \min\{\textsc{dos}(D), \textsc{dos}(E)\} = \min\{0.97, 0.94\} = 0.94, \\ \textsc{dos}(A) = \min\{\textsc{dos}(B), \textsc{dos}(C)\} = \min\{0.94, 0.93\} = 0.93. \end{array}$$

Similarly, for the second argument the DOS will come through as 0.93, albeit via other propositions, viz. J and H. □

When rules of inference are defeasible, however, it *does* matter which argument is used in an explanation, as the following example shows.

Example 10.6 Clearly, in this case it *does* matter to mention whether A is supported by

$$\begin{array}{lll} A \leftarrow\!\!(0.01)\!\!- & B \leftarrow\!\!(0.01)\!\!- & D_{0.97} \\ & & E \leftarrow\!\!(0.01)\!\!- \; F_{0.94} \\ & C \leftarrow\!\!(0.01)\!\!- & F_{0.94} \\ & & G_{0.93} \end{array}$$

or by

$$\begin{array}{lll} A \leftarrow\!\!(0.99)\!\!- & H \leftarrow\!\!(0.99)\!\!- & D_{0.97} \\ & & J \leftarrow\!\!(0.99)\!\!- \; F_{0.94} \\ & & \qquad\qquad G_{0.93} \\ & D_{0.97} & \end{array}$$

Although both arguments ground in equal premises, the first argument make use of rules that are weaker than the second one. □

Thus, in a context in which rules of inference may have different strengths, we have not said everything when we say that, e.g., "$E = \{h_1, h_2, h_3\}$ explain D". This only says that arguments for D on the basis of E exist. One could additionally say that "D is explained by a strong argument based on E," or "if E were true, then D would be a matter of course". Even more is said when we offer a specific argument a on the basis of E and supporting D, compute to what extent a supports D, and investigate the relation of a with other arguments, explanations, and counter-arguments to explanations, that may further be constructed.

10.4 The most plausible explanation

Explanatory dialectics amounts to what is done in the chapter on argumentation (Chapter 8 on page 149), but the goals as well as the terminology are different: arguments that support an observation are now considered as explanations, and claims that must be defended are now considered as observations that must be explained.

In Section 8.8 on page 168 it was described how to find all arguments for a particular proposition: begin with the conclusion and traverse backwards through the justifications until you encounter propositions that are able to function as premises in an argument. This algorithm is simple when premises are either true or false. However, the problem becomes less obvious when rules as well as propositions may possess different DOSs. How far do we have to go back in the justifications? Is there a point beyond which search for support makes no longer sense? If so, then where is that point? If not, then how do we control the search process?

To explain the essential aspects of finding a strongest argument, we make use of the following (somewhat contrived) combination of rules and propositions.

Example 10.7 (Find a strongest argument) Suppose p is a proposition for which we would like to find a strongest argument, and p_i are propositions that are connected via p in the following manner:

$$
\begin{array}{l}
\text{p} \leftarrow\!\!(0.83)\!-\, p_1\ [0.01] \\
\qquad \leftarrow\!\!(0.83)\!-\, p_2\ [0.65] \\
\qquad\qquad \leftarrow\!\!(0.83)\!-\, p_3\ [0.01] \\
\qquad\qquad\qquad \leftarrow\!\!(0.83)\!-\, p_4\ [0.01] \\
\qquad\qquad\qquad\qquad \leftarrow\!\!(0.83)\!-\, p_5\ [0.01] \\
\qquad\qquad\qquad\qquad\qquad \ldots
\end{array}
$$

The numbers in square brackets denote the DOB's of the corresponding propositions. To find a strongest argument for p we neglect $\text{DOB}(p)$ and try to compute $\text{DOS}(p)$. To this end, we adopt interest in the DOS of p_1. Since the DOS of p_1 is unknown, we adopt interest in the DOS of p_2 as well, and so forth. The regression does not continue indefinitely, and may stop for two reasons. The first reason presents itself when all incoming rules are weaker than the DOB of that proposition. E.g., if

$$
\begin{array}{lll}
\text{DOB}(x) = 0.98, \text{ and} & \text{rules for } x = \{\ x \leftarrow\!\!(0.78)\!-\ldots, & \\
& \qquad\qquad\quad x \leftarrow\!\!(0.97)\!-\ldots, & \\
& \qquad\qquad\quad x \leftarrow\!\!(0.94)\!-\ldots, & \\
& \qquad\qquad\quad x \leftarrow\!\!(0.83)\!-\ldots, \quad \ldots\ \} &
\end{array}
$$

In that case, no rule can surpass $\text{DOB}(x)$, so that $\text{DOS}(x) = \text{DOB}(x) = 0.98$. A second reason to stop is that we are about to pass a "point of no return" regarding support. Consider for example p_2. Since $\text{DOB}(p_2) = 0.65$, all incoming rules (in this case $p_2 \leftarrow\!\!(0.83)\!- p_3$) must provide a support of at least 0.65 to improve on p_2's DOB. Since $\text{strength}(p_2 \leftarrow\!\!(0.83)\!- p_3) = 0.83$, this means that p_3 needs a DOS of at least $0.65/0.83 = 0.78$ to supersede p_2's DOB. This means that p_4 needs a DOS of at least $0.78/0.83 = 0.94$ to fulfill p_3's requirement of having a DOS of at least 0.78. In turn, p_5 needs a DOS of at least $0.94/0.83 > 1.00$ to fulfill p_4's requirement of having a DOS of at least 0.94. The latter is clearly impossible, so that the regression stops at p_4. Thus, strong support weakens along the way if it must be fetched from remote places. □

Here is an (admittedly somewhat contrived) analogy: some of the Dutch refuel their car in Germany. They do this because German petrol is taxed less heavily than Dutch petrol. This operation makes sense only if you are living less than, say, 20 kilometers from the border. Otherwise, a drive to the pump would cost you more than getting the petrol.

A more systematic description of how to find a strongest argument is Algorithm 6.

Algorithm 6 : Finding a strongest argument for proposition p

Input: A proposition p, and a minimally required support $0 \leq s \leq 1$

```
 1: if DOB(p) ≥ s then # p by itself could be an argument
 2:    strongest_argument_thus_far := p
 3:    m := DOB(p)
 4: else # p needs support from other propositions to surpass s
 5:    strongest_argument_thus_far := nil
 6:    m := s
 7: for each rule R with consequent p do
 8:    next rule if strength(R) < m
 9:    arg := strongest argument through R for p, stronger than m # call Algorithm 7
10:    next rule if arg = nil
11:    if strength(arg) > strength(strongest_argument_thus_far) then
12:       strongest_argument_thus_far := arg
13: return strongest_argument_thus_far
```

Here is the output of the algorithm applied to Example 10.7 on the preceding page:

```
1. ==>SUPPORT p (dob 0).
```

Algorithm 7 : Finding a strongest argument through rule R

Input: A rule R, and a minimally required support $0 \leq s \leq 1$

1: $m := s/\text{strength}(R)$
2: **for** each p in R's antecedent **do**
3: strongest_arg_for$[p]$:= strongest argument for p, stronger than m # call Algorithm 6
4: **return** `nil` **if** strongest_arg_for$[p] =$ `nil`
at this point, all elements R's antecedent are supported by arguments stronger than m
5: **return** argument that is formed by R and all strongest arguments for all elements in R's antecedent

```
p1 [0.01]   /* numbers behind propositions represent dob */
p2 [0.65]
p3 [0.01]
p4 [0.01]
p5 [0.01]
p6 [0.01]
p7 [0.01]
p8 [0.01]
p9 [0.01]
.
.
.
Rule1:  p <-(0.83)- p1 [1]   /* numbers behind rules represent dob */
Rule2: p1 <-(0.83)- p2 [1]   /* numbers between parentheses represent rule strength */
Rule3: p2 <-(0.83)- p3 [1]
Rule4: p3 <-(0.83)- p4 [1]
Rule5: p4 <-(0.83)- p5 [1]
Rule6: p5 <-(0.83)- p6 [1]
Rule7: p6 <-(0.83)- p7 [1]
Rule8: p7 <-(0.83)- p8 [1]
Rule9: p8 <-(0.83)- p9 [1]
.
.
.
```

Table 10.1: Rules and propositions of Example 10.7 on page 208.

```
 2. P needs 0, gives 0, so rules need > 0.
 3. Rules for p: Rule1
 4. Strength of Rule1 is 0.83 > 0, so Rule1 deserves to be explored.
 5. | ==>EXPAND Rule1: p <-(0.83)- p1 [1]
 6. | Trying to find argument for p1 (which is in antecedent of Rule1).
 7. | | ==>SUPPORT p1 (dob 0.01).
 8. | | P1 needs 0, gives 0.01, so rules need > 0.01.
 9. | | Rules for p1: Rule2
10. | | Strength of Rule2 is 0.83 > 0.01, so Rule2 deserves to be explored.
11. | | | ==>EXPAND Rule2: p1 <-(0.83)- p2 [1]
12. | | | Trying to find argument for p2 (which is in antecedent of Rule2).
13. | | | | ==>SUPPORT p2 (dob 0.65).
14. | | | | P2 needs 0.01, gives 0.65, so rules need > 0.65.
15. | | | | Rules for p2: Rule3
16. | | | | Strength of Rule3 is 0.83 > 0.65, so Rule3 deserves to be explored.
17. | | | | | ==>EXPAND Rule3: p2 <-(0.83)- p3 [1]
18. | | | | | Trying to find argument for p3 (which is in antecedent of Rule3).
19. | | | | | | ==>SUPPORT p3 (dob 0.01).
20. | | | | | | P3 needs 0.78, gives 0.01, so rules need > 0.78.
```

```
21. | | | | | | Rules for p3: Rule4
22. | | | | | | Strength of Rule4 is 0.83 > 0.78, so Rule4 deserves to be explored.
23. | | | | | | | ==>EXPAND Rule4: p3 <-(0.83)- p4 [1]
24. | | | | | | | Trying to find argument for p4 (which is in antecedent of Rule4).
25. | | | | | | | | ==>SUPPORT p4 (dob 0.01).
26. | | | | | | | | P4 needs 0.94, gives 0.01, so rules need > 0.94.
27. | | | | | | | | Rules for p4: Rule5
28. | | | | | | | | Rule5 (strength 0.83) is skipped.
29. | | | | | | | | No argument for p4 was able to support p4 > 0.94,
30. | | | | | | | | Moreover, dob(p4)=0.01 < what is needed (0.94),
31. | | | | | | | | so there is no support for p4 > 0.94.
32. | | | | | | | | <==SUPPORT p4 returns (without an argument).
33. | | | | | | | <==EXPAND Rule4 returns without argument, because support
34. | | | | | | |           for element in antecedent, namely p4, is undefined.
35. | | | | | | Rule4 does not lead to an argument > 0.78.
36. | | | | | | No argument for p3 was able to support p3 > 0.78,
37. | | | | | | Moreover, dob(p3)=0.01 < what is needed (0.78),
38. | | | | | | so there is no support for p3 > 0.78.
39. | | | | | | <==SUPPORT p3 returns (without an argument).
40. | | | | | <==EXPAND Rule3 returns without argument, because support
41. | | | | |           for element in antecedent, namely p3, is undefined.
42. | | | | Rule3 does not lead to an argument > 0.65.
43. | | | | No argument for p2 was able to support p2 > 0.65,
44. | | | | but dob(p2) > what is needed (0.01), so we use the dob of p2.
45. | | | | ::> Thus, dos(p2) = dob(p2) = 0.65.
46. | | | | <==SUPPORT p2 returns 0.65
47. | | | Support for p2 is 0.65
48. | | | <==EXPAND Rule2 returns p1 <-(0.83)- p2 [1]
49. | | Best argument via Rule2 is now determined.
50. | | Trying next rule.
51. | | No more rules for p1.  Best arg. via Rule2.  Dos(p1) can now be computed.
52. | | ::> dos(p1) = strength(Rule2)*min{dos's of strongest sub-arguments} = 0.53.
53. | | <==SUPPORT p1 returns strongest supporter, viz. p1 <-(0.83)- p2 [1]
54. | Support for p1 is p1 <-(0.83)- p2 [1]
55. | <==EXPAND Rule1 returns the following argument:
56. | p <-(0.83)- p1 [1]
57. |   p1 <-(0.83)- p2 [1]
58. Best argument via Rule1 is now determined.
59. Trying next rule.
60. No more rules for p.  Best arg. via Rule1.  Dos(p) can now be computed.
61. ::> dos(p) = strength(Rule1)*min{dos's of strongest sub-arguments} = 0.44.
62. <==SUPPORT p returns strongest supporter, viz. the following argument:
63. p <-(0.83)- p1 [1]
64.   p1 <-(0.83)- p2 [1]
```

At this point, we know how explanatory dialectics works when precisely one observation must be explained; i.e., we know how explanatory dialectics works when $|D| = 1$. With propositional abduction, the case $|D| = 1$ can easily be generalized into the case $|D| > 1$, since $D = \{d_1, \ldots, d_n\}$ can be transformed into $D' = \{d_1 \wedge \ldots \wedge d_n\}$. A problem with explanatory dialectics (and Algorithm 6 on page 209 in particular), is that the case with one single observation does not immediately generalize to the case in which more than one observation must be explained. This problem is caused by the fact that conjunction is not a built-in feature in the restricted object-language of argumentation (Chapter 8). There are ways to deal with this situation, for example by introducing literals and extra rules of inference to represent conjunctions that are relevant to the problem. We stop here, however, since pursuing this approach would bring us outside the scope of the chapter.

Another problem with Algorithm 6 is that it returns only one argument, namely, one of the

strongest arguments. This is a problem, because in almost all argumentation processes, arguments are tried one after the other, and it may well happen that not the first, but the *last* argument turns out to be victorious. Fortunately, by a relatively minor change we can also let Algorithm 6 return the n strongest arguments. It is also possible to do this with lazy evaluation (Scheme, Haskell), so that arguments are computed on demand only.

Evaluation

Explanatory dialectics remedies two shortcomings of propositional abduction, namely, the presence of NP-complete consistency checks, and the lack of expressive power to represent non-deductive explanations. We must be honest: with explanatory dialectics the problems with computational complexity are not cleared out of the way, but they are avoided. The burden to make sure that the algorithm finds all the relevant arguments is shifted to the knowledge-engineer, who's task (among other things) is to transform unstructured knowledge into an ordered rule base that can be processed with simple inference techniques. This problem is further discussed in Section 8.4 on page 161.

PROBLEMS (Sec. 10.4)

1. Given

proposition	DOB	*proposition*	DOB
$p, \neg q \dashv(0.90)\!\rightarrow r$	1.00	r	0.20
$p \dashv(0.85)\!\rightarrow r$	1.00	$\neg q, r \dashv(0.90)\!\rightarrow s$	1.00
p	0.80	$r \dashv(0.80)\!\rightarrow s$	1.00
$\neg q$	0.70	$p, \neg r, s \dashv(0.71)\!\rightarrow \neg q$	1.00

   Compute all DOSs that are the result of searching the strongest argument for s. Give the strongest argument for s. (Solution.)

2. Let

proposition	DOB		
$p_{i+1} \dashv(x)\!\rightarrow p_i$	1.00	$1 \leq i$	
p_i	y	$1 < i$	
p_1	0.00		where $0.00 \leq x, y \leq 1.00$.

   (a) Compute $\text{DOS}(p_1)$ for $x, y = 0.90, 0.50$. What is the length of the strongest argument for p_1? Where does the search stop? (Solution.)

   (b) Determine how $\text{DOS}(p_1)$, the length of the strongest argument for p_1, and the search depth depend on (x, y). (Solution.)

3. The weakest link principle says that the strength of an argument is determined by its weakest link. The certainty-factor propagation principle says that the strength of an argument is determined by the strength of its immediate sub-arguments, times the strength of its top-rule.

   For example, the strength of

$$A = s \leftarrow\!(0.90)\!- \begin{array}{l} \neg q \ [0.97] \\ r \leftarrow\!(0.85)\!- p \ [0.98] \end{array}$$

   according to the weakest link principle is $\min\{0.97, 0.98, 0.85, 0.90\} = 0.85$. (The numbers 0.97 and 0.98 are the DOBs of $\neg q$ and p, respectively, and we assume that all rules have a DOB that is equal to one.) The strength of A according the certainty-factor propagation principle is equal to $0.90 * \min\{0.97, 0.85 \times 0.98\} = 0.81$. Find two arguments A and B,

such that B interferes with A (but not conversely) according to the weakest link principle, and A interferes with B (but not conversely) according to the certainty-factor propagation principle. (Solution.)

4. Consider the following situation: when the bridge is open, you'll have to wait. Waiting often (but not always) causes you to be late. You'll have to wait, unless there is a detour. When there is a detour, you usually don't have to wait. (Usually, but not always. The detour might be blocked, for example.) And so forth:

proposition	DOB
bridge-open –(0.90)→ have-to-wait	1.00
bridge-open, there-is-a-detour –(0.95)→ ¬have-to-wait	1.00
have-to-wait –(0.90)→ too-late	1.00
missed-the-bus –(0.80)→ have-to-wait	1.00
missed-the-bus, high-bus-frequency –(0.95)→ ¬have-to-wait	1.00
flat-tire –(1.00)→ have-to-walk	1.00
have-to-walk –(0.80)→ too-late	1.00
missed-the-bus –(1.00)→ ¬flat-tire	1.00
flat-tire –(1.00)→ ¬missed-the-bus	1.00
it-is-morning –(0.80)→ high-bus-frequency	1.00
it-is-morning	1.00

Suppose, in addition, that a student, Ronald, tries to explain to his (or her) teacher, John, that he is too late.

   (a) Suppose Ronald maintains that he missed the bus (DOB = 1.00), while John is skeptical (DOB = 0.50). Let us say they agree to middle their DOBs (implicitly so) and agree to ground arguments in "missed-the-bus" with DOB(missed-the-bus) = 0.75. How may Ronald explain his late arrival? (Solution.)

   (b) Do there exist arguments against the explanation(s) given at (4a)? If so, do these arguments defeat the explanation(s) given at (4a)? (Solution.)

   (c) Ronald presents additional information: the bridge was open. Let us suppose that both parties (implicitly), in addition to the earlier information, agree on DOB(bridge-open) = 0.75. How may Ronald explain his late arrival? Are there counterarguments? (Solution.)

   (d) Suppose John brings Ronald's attention to the fact that there are several bridges in town, so that there exists a detour, and suppose Ronald fully agrees to this fact: DOB(there-is-a-detour) = 1.00. What has changed relative to (4c)? (Solution.)

   (e) Ronald's arsenal of reasons has not exhausted, yet—he has additional information: his bike had a flat tire. Let us suppose that both parties, in addition to the earlier information, agree on DOB(flat-tire) = 0.50. (Ronald claims "flat-tire," but John does not believe it.) What has changed relative to (4d)? (Solution.)

5. (Jensen *et al.*, 1990.) An oil wildcatter must decide either to drill or not to drill. He is uncertain whether the hole is dry, wet or soaking in oil. The wildcatter can make seismic soundings that will help determine the geological structure of the site. The soundings will give a closed reflection pattern (indication for much oil), an open pattern (indication for some oil) or a diffuse pattern (almost no hope for oil).

   The wildcatter has two decisions to make, namely whether to test with seismic soundings costing $10,000 and whether to drill costing $70,000. The utility gained from drilling is determined by the state of the hole (dry, wet or soaking).

   The problem is modeled in Figure 10.10. In Bayesian network terminology, Figure 10.10 is called an *influence diagram*. The rectangles are so-called *decision nodes*. They represent

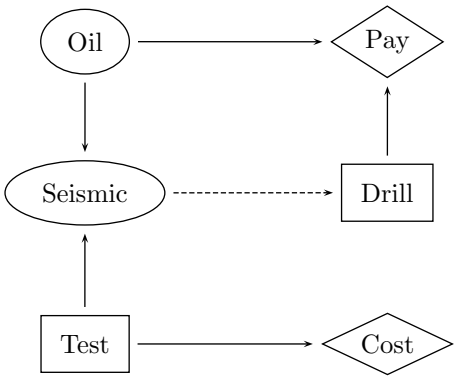


Figure 10.10: Influence diagram modeling the oil wildcatters decision problem.

decisions that can be taken by the oil wildcatter. For example, an oil wildcatter can decide to test or not to test. Independently, he can decide to dril or not to drill. The diamonds are so-called *utility nodes*. They describe how much money is earned (or lost), depending on predecessor nodes. For example, the amount that will be paid depends on whether there is oil and whether the oil wildcatter decides to drill. The ovals are so-called *chance nodes*. They describe certain events, and the likelihood that events happen, depending on predecessor nodes. For example, the form of the seismic pattern depends on whether there is oil, and whether the oil wildcatter decides to perform seismic tests at all.

The chance node *Oil* has states "dry," "wet," and "soaked," the chance node *Seismic* has states "closed," "open," and "diffuse," the decision nodes *Test* and *Drill* has actions "yes" and "no." Table 10.2 shows the conditional probability tables (CPTs) of all nodes. For example, the probability that a seismic test will return a closed reflection pattern in the presence of a soaked oil well, is equal to 0.50. A seismic test costs $10,000, the amount we earn by drilling a soaked well is $200,000, and the a priori probability on the presence of a soaked well is equal to 0.20. The arrow from *Seismic* to *Drill* does not represent a causal dependency. Instead it shows that when the decision of *Drill* has to be made, the state of *Seismic* must be known.

Oil	*Test*	***Seismic***		
		"closed"	"open"	"diffuse"
"dry"	"yes"	0.10	0.30	0.60
"dry"	"no"	0.33	0.33	0.33
"wet"	"yes"	0.30	0.40	0.30
"wet"	"no"	0.33	0.33	0.33
"soaked"	"yes"	0.50	0.40	0.10
"soaked"	"no"	0.33	0.33	0.33

Oil	*Drill*	***Pay***
"dry"	"yes"	-70
"dry"	"no"	0
"wet"	"yes"	50
"wet"	"no"	0
"soaked"	"yes"	200
"soaked"	"no"	0

Oil		
"dry"	"wet"	"soaked"
0.50	0.30	0.20

Test	***Cost***
"yes"	-10
"no"	0

Table 10.2: Conditional probability tables (left) and utility tables (right) for the influence diagram modeling the oil wildcatters decision problem.

(a) Translate the Bayesian network into defeasible rules of inference. (There are different solutions. One solution variant maintains the style of propositional logic, while

Test (decision)	
"yes"	"no"
22.50	20.00

Seismic (event)		
"closed"	"open"	"diffuse"
0.29	0.34	0.37

Drill (decision)	
"yes"	"no"
20.00	0.00

Oil (event)		
"dry"	"wet"	"soaked"
0.50	0.30	0.20

Table 10.3: Result of probabilistic propagation before anything is decided.

another variant follows the style of first-order logic.) (Solution.)

(b) Let $\mathcal{A} = \langle B, H, D \rangle$ be an abduction problem with background knowledge B equal to what is obtained at (5a), the collection of possible hypotheses equal to $H = \{$test, don't test, drill, don't drill$\}$ and $D = \{$cost is \$10,000, paid \$200,000, nett profit is \$190,000$\}$. Give an explanatory argument e for D on the basis of B and H. Which hypotheses does e use? Is e undefeated? Why? (Or why not?) (Solution.)

(c) In the context of Bayesian belief networks, Figure 10.10 on the preceding page and Table 10.2 on the facing page can be used to decide whether a test is profitable or not. This is done by calculating the probabilistic dependencies in the network by means of probability propagation algorithms. (These algorithms are non-trivial, and do not work on all Bayesian belief networks.) Initially, Table 10.3 shows the result of a propagation before anything is known about whether the wildcatter will test with seismic soundings or not. From Table 10.3 we see that the expected utility of testing is \$22,500 while the expected utility of not testing is only \$20,000. (Again, these numbers are computed with the help of non-trivial propagation algorithms.) This implies that it will be best for the wildcatter to test with seismic soundings which will give him the states of *Seismic* by Bayesian means.

Discuss in the context of an argumentation formalism the possibilities of solving the problem whether testing is profitable.

Chapter 11

Learning new rules

The existence of rules of inference is a necessary condition to reason with data. The previous chapters departed from the assumption that such rules exist, and can be used. But what if they do not exist? In that case we will have to produce, some would say *induce*, rules of inference from raw data. In this chapter, a number of algorithms will be formulated to learn rules of inference from data.

11.1 Cases

The idea behind rule learning is that rules can be learnt from cases. The notion of case is based on the more elementary notion of situation description. A *situation description* is a description of a situation at a particular moment in time consisting of a enumeration of properties that hold at that point in time, and/or several events that happened during that point in time. Think of it like a snapshot. An example of (a fragment of) a list of situation descriptions is

Timepoints	*Cases*
⋮	⋮
t2006:	Tour de France, World Championship Football, DeNiro Movie
t2007:	¬New president, DeNiro Movie, Eclipse
t2008:	Tour de France, Olympic Games, European Championship Football, New president, DeNiro Movie
t2009:	Tour de France, DeNiro Movie
t2010:	Tour de France, World Championship Football, DeNiro Movie
t2011:	Tour de France, DeNiro Movie
t2012:	Olympic Games, Tour de France, European Championship Football, DeNiro Movie
⋮	⋮

Here, time points are years, and each case is a (more or less arbitrary) enumeration of events that happened during that year. Thus, in 2006 there was the Tour de France, there was the World Championship Football, and in that year, a Robert DeNiro Movie came out. Then, in 2007, there was the Eclipse, we had the same president throughout that year and, again, a Robert DeNiro Movie came out. And so forth.

A more abstract example of a (chronological) list of situation descriptions is

Timepoint	*Situation description*
⋮	⋮
t2006:	$\neg a, b, \neg d, g, \neg j, \neg k, m$
t2007:	$b, \neg c, d, \neg e, f, h, \neg i, j$
t2008:	$a, c, \neg d, \neg e, f, \neg k, l, \neg m$
t2009:	$a, b, d, \neg g, h, j, k, \neg m$
t2010:	$\neg b, c, e, \neg f, g, \neg i, \neg j, n$
t2011:	$a, \neg c, e, \neg f, \neg k, l, \neg m$
⋮	⋮

A typical list of situation descriptions from which sensible rules can be distilled from, consists of about $1,000$ or more situations, where each situation consists of, say, 20 or more features. The more situation descriptions and the more features, the better.

Rules can be extracted from data in different ways. In this chapter, we will confine ourselves to discovering patterns in situations per sé, rather than discovering patterns in situations through time. For example, if c occurs every time a or b occurs, and if c is absent every time a and b are absent, then there is an algorithm in this chapter that will produce the rules $a \rightarrow c$ and $b \rightarrow c$.[1] However, this chapter does not provide methodes to discover causal patterns that may occur trough time. For example, if c happens at t_n every time a happens at t_{n-1}, then we might say that a causes c and write the rule $a \text{–}(causes)\!\!\rightarrowtail c$. Learning causal rules, however, is beyond the scope of this chapter. Be reassured: most if not all treatises on rule learning are aimed at discovering patterns in situations per sé and not at discovering causal patterns through time [77].

Now for cases. A *case* is a situation description in which one event is singled out, usually because we want to know under which particular circumstances this single event has happened. An example of a case list for f is

Timepoint	*Case*
⋮	⋮
t2006:	$\neg a, b, \neg d, g, \neg j, \neg k, m$
t2007:	$b, \neg c, d, \neg e, h, \neg i, j \succ f$
t2008:	$a, c, \neg d, \neg e, \neg k, l, \neg m \succ f$
t2009:	$a, b, d, \neg g, h, j, k, \neg m$
t2010:	$\neg b, c, e, g, \neg i, \neg j, n \succ \neg f$
t2011:	$a, \neg c, e, \neg k, l, \neg m \succ \neg f$
⋮	⋮

This list is made by by taking f out of the situation description and putting it at the RHS behind a $\succ$ -sign. Cases ending with an f are sometimes called *positive examples for* f, or *positive instances for* f, because they describe situations in which f occurs. Cases ending with $\neg f$ are sometimes called *negative examples for* f, or *negative instances for* f, because they describe situations in which $\neg f$ occurs. There are also cases that are neither positive nor negative. Case lists for other literals such as a, $\neg a$, b and $\neg b$, can be made in a similar fashion.

Representing cases by using the $\succ$ -notation demonstrates nicely that examples play the role of cases in machine learning. On the other hand, a disadvantage of the use of $\succ$ is that it is redundant and not really essential to the rule learning algorithms that we present in this chapter. If an algorithm is dedicated to learning rules on f, it simply singles out f without having to rely on a $\succ$ -symbol of some sort. Therefore, this chapter further works with the earlier mentioned and more general notion of situation description.

[1] Algorithm 10 on page 227.

11.2 Some concepts from machine learning

Machine learning terminology is the best vehicle to understand how rules can be learnt from cases. We therefore introduce some basic notions and concepts.

In machine learning a rule, or set of rules, would be called a *hypothesis*, and a case would be called an example, observation, or *instance*. A typical machine learning task, then, is to formulate a hypothesis, H, that explains the classification of observed instances $I_1, \ldots, I_n$ and predicts the classification of unobserved (unseen) instances $I_{n+1}, \ldots$. A typical machine learning task, for example, is to explain why an insurance company accepts certain insurance applications and rejects others. Another example is to discover which customers tend to buy a certain commercial product, based on customer profiles.

As a running example in this section we try to find an hypothesis H that explains the following series:

$$\begin{array}{lllll} (22,25) \to + & (76,54) \to - & (37,23) \to + & (37,37) \to + & (25,80) \to - \\ (34,75) \to - & (85,78) \to - & (22,38) \to + & (90,10) \to - & (50,50) \to - \end{array} \tag{11.1}$$

Thus, the first instance is the pair $(22, 25)$ that is classified positive, the second instance is the pair $(76, 54)$ that is classified negative, and so forth. The objective is to find, or rather produce, a simple hypothesis that explains all positive instances, and rules out (does not explain) all negative instances. An example of a hypothesis is "the set of positive points is formed by the disk that consists of all points with a distance less than 5 from $(25, 25)$". Another example of a hypothesis is "the set of positive points is formed by the set of all points (x, y) such that $x + y$ is even".

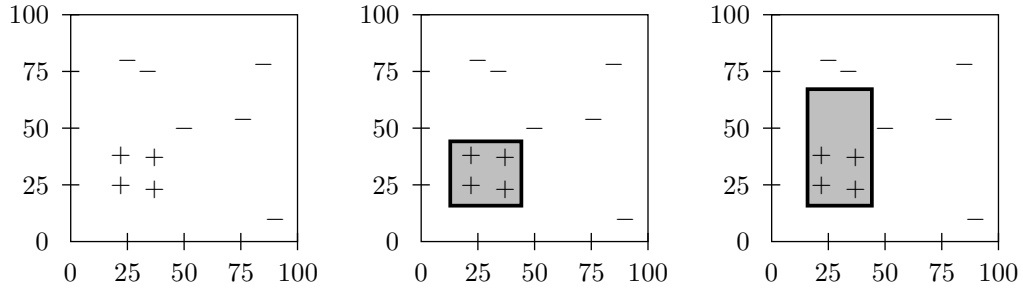


Figure 11.1: Left: positive and negative instances. Middle: a hypothesis that is consistent with the data. Right: another consistent hypothesis.

In machine learning, the hypotheses have a certain format. This format often follows from the data structure that is selected to represent the hypothesis. Let us say here that something is a hypothesis, if it is a closed rectangle within the hundred-field $[0, 100] \times [0, 100]$. More in particular, let us say that H is a hypothesis if and only if $H = [a, b] \times [c, d]$, where $a < b$, $c < d$ are integers between (or equal to) 0 and 100. Thus, a rectangle qualifies as a hypothesis only if its boundary lies on the grid of integers. The set of all possible hypotheses is called the *hypothesis space*, and is written $\mathcal{H}$. Thus,

$$\mathcal{H} = \{[a, b] \times [c, d] \mid a, b, c, d \in \{0, 1, \ldots, 100\}, \text{ and } a < b, c < d\} \tag{11.2}$$

Some hypotheses perform better than others. Performance is expressed in terms of coverage, matching and accuracy.

Classification Every hypothesis classifies instances either positive or negative. The hypothesis (rectangle) $H = [20, 30] \times [20, 30]$, for instance, classifies eight examples as negative and two examples as positive.

Match A hypothesis matches the data for better or for worse. The match indicates how well a hypothesis matches, or classifies, the data.

$$match(H) =_{Def} \frac{\text{instances correctly classified}}{\text{total number of instances (in the data)}} \tag{11.3}$$

If $match(H) = 1$, then H classifies all examples correctly; if $match(H) = 0$, then H classifies all examples incorrectly. (Also useful.) Two out of ten examples are classified incorrectly by $H = [20, 30]^2$, because $(37, 23)$ and $(37, 37)$ are classified incorrectly as negative. Hence, $match(H) = 8/10$.

Consistency A hypothesis is *consistent* with the data if it classifies all instances correctly, i.e, if $match(H) = 1$.

Coverage An instance I is said to be *covered* by a hypothesis if that hypothesis classifies I as positive.

$$coverage(H) =_{Def} \text{ total number of data-instances covered} \tag{11.4}$$

The coverage of a hypothesis may change, depending on whether new instances are classified positive.

Range The range of a hypothesis is equal to the total number of instances that are covered by it, regardless whether the instances occur in the data or not.

$$range(H) =_{Def} \text{ total number of instances (seen and unseen) covered} \tag{11.5}$$

The range of a hypothesis never changes, even in the presence of additional instances.

Accuracy A hypothesis may be more or less accurate for the instances it covers.

$$accuracy(H) =_{Def} \frac{\text{number of positive data-instances covered}}{\text{total number of data-instances covered}} \tag{11.6}$$

The accuracy of a hypothesis indicates its performance on the data it covers. Both examples covered by $H = [20, 30]^2$ are classified correctly. Hence, $accuracy(H) = 2/2 = 1$.

There is always a tradeoff between accuracy and coverage. Accuracy can be improved by reducing the range (= coverage) of an hypothesis. An extremely accurate hypothesis can always be obtained by identifying it with a particular instance. However, such an hypothesis can only be applied to that particular instance and will be of little use to classify future instances. Conversely, if we enlarge the range of an hypothesis, it can be applied to a larger number of unseen instances, but the downside of such hypotheses is that their accuracy often decreases to a degree below of what is acceptable.

Specificity Hypothesis H_1 is said to be *as specific as* hypothesis H_2 if all instances that covered by H_1 are covered by H_2 as well. In that case, H_2 is said to be *as general as* hypothesis H_1. Often, the notion of specificity is used to bring some order in the hypothesis space so that the search for a hypothesis proceeds from general to specific or vice versa.

The above terminology suffices to proceed with the main objective of this chapter, namely, to formulate algorithms that learn rules from cases.

PROBLEMS (Sec. 11.2)

1. How many hypothesis does $\mathcal{H}$ specified by Eq. (11.2) contain? (Solution.)
2. What is the most general hypothesis in $\mathcal{H}$? (Solution.)

(a) Is it unique? (Solution.)
(b) How many instances does it cover? (Solution.)
(c) How many instances are classified correctly? (Solution.)
(d) How well are the instances matched by the most general hypothesis? (Solution.)
(e) What is the accuracy of the most general hypothesis? (Solution.)

3. What is the most specific hypothesis in $\mathcal{H}$? Same questions as with (2). (Solution.)
4. Let H be the hypothesis $[20, 60]^2$. Same questions as with (2). (Solution.)
5. Formulate a hypothesis H that is consistent with the data as provided by (11.1). (Solution.)
6. Formulate a most general hypothesis H that is consistent with the data. Is it unique? (Solution.)
7. Formulate a most specific hypothesis H that is consistent with the data. Is it unique? (Solution.)
8. Explain why a hypothesis H_1 with $match(H_1) = 0$ can be more useful than a hypothesis H_2 with $match(H_2) = 1/2$.
9. Usually, the accuracy of a hypothesis increases if it is made more specific. Show with an example that, in some cases, the accuracy of a hypothesis may decrease even if it is made more specific.

11.3 Learning one rule

In this section we describe two ways of learning a rule. The first algorithm is simple but inefficient. The second algorithm is more sophisticated and solves all mentioned shortcomings of the first algorithm.

An exhaustive algorithm

Suppose the following five cases:

1.	a, c, b, d	4.	$\neg a, \neg b$
2.	b, a, d	5.	$a, \neg d$
3.	$b, c, \neg d$		

and suppose that we would like to learn a rule for d on the basis of these five cases. The idea is to begin with a rule with an empty antecedent, and gradually make the antecedent more specific until the rule covers all positive instances (Case 1,2) and no negative instances (Case 3,4,5). (Case 4 is considered negative here as well, since d is absent.) Other approaches abstain in such cases, i.e., other approaches do not classify cases in which d and $\neg d$ do not occur.) Thus, the algorithm moves from general to specific.

The rule for d with the empty antecedent is

$$\rightarrow d.$$

This rule covers all positive examples, but also three negative examples, viz. 3,4 and 5. One way to avoid the negative examples is to make the antecedent more specific. This can be done in five different ways, since five other relevant literals occur in the data, viz. a, b, c, $\neg a$ and $\neg b$. (Literal $\neg c$ does not occur in the data and, hence, does not need to be taken into account.) Thus, the five possible one-step specializations of the rule "$\rightarrow d$" are

Specialization	*Performance*	*Action*
$a \rightarrow d$	covers negative example nr. 5	make antecedent more specific
$b \rightarrow d$	covers negative example nr. 3	make antecedent more specific
$c \rightarrow d$	violates positive example nr. 2	remove rule
$\neg a \rightarrow d$	violates positive example nr. 1	remove rule
$\neg b \rightarrow d$	violates positive example nr. 1	remove rule

Of the specializations thus obtained, some must be improved (viz. the first two), while others become useless (viz. the last four). The antecedents of the rules that perform better are further specialized. For the moment, let us neglect all specializations of "$b \rightarrow d$," and focus on all specializations of "$a \rightarrow d$":

Specialization	*Performance*	*Action*
$a, b \rightarrow d$	covers all positive examples and avoids all negative examples	keep rule
$a, c \rightarrow d$	violates positive example nr. 2	remove rule
$a, \neg b \rightarrow d$	violates positive example nr. 1	remove rule

The (antecedent of the) rule "$a, b \rightarrow d$" is general enough to cover the positive instances 1 and 2, yet specific enough to miss the negative instances 3,4 and 5. So this is a good hypothesis. The remaining two rules are dropped because they miss a positive instance. The algorithm goes on (eventually producing another consistent hypothesis $b, a \rightarrow d$) but we stop here. The entire algorithm is listed as Algorithm 8. When this algorithm is implemented and applied to the above

Algorithm 8 Exhaustive algorithm to learn all most-general consistent rule antecedents

Input: P, a list of positive instances
Input: N, a list of negative instances
- set O to [null-hyp]; # list of open hypotheses, i.e., hypotheses that can be improved
- set C to [ ] # will contain closed hypotheses, i.e., hypotheses that cannot be improved
while O has elements **do** # there are hypothesis that can be improved
  - replace each h in O by all one-step specializations of h
  for each h in O **do**
    if h misses (i.e., fails to cover) some member of P **then** # h is too specific
      - remove h from O, and move on to the next element in O
    else if h misses every member of N **then** # h is consistent
      - remove h from O; # since it does not need to be further specialized
      - add h to C, provided h is not more specific than some element in C # otherwise, h would not be a real contribution
  # if, at this point, O contains hypothesis, these hypothesis are too general and must be specialized
O is now empty
- **return** C;

five instances, it produces an output as displayed in Fig. 11.2 on the facing page.

A pleasant property of Algorithm 8 is that it is complete: it finds all most-general hypotheses that are consistent with the data. But the algorithm also possesses a number of less pleasant properties:

1. It explores all possible specializations of all rule antecedents, which leads to a combinatorial explosion of the search space. (Exercise 11.3.)
2. The algorithm produces no hypotheses if the data is inconsistent, e.g., if the data contains errors. See Fig. 11.3 on page 224, 2nd diagram.
3. The algorithm produces bad hypotheses if the data is noisy. See Fig. 11.3 on page 224, 3rd diagram.

```
== Positive for d: ====================================================
Sit1: a, b, c, d
Sit2: a, b, d
== Negative for d: ====================================================
Sit3: ~d, b, c
Sit4: ~b, ~d, ~a
Sit5: a, ~d
========================================================================
Specializing Hyp0: (the null hyp) => 4 new hypotheses (total 5).
Hyp1: a (from Hyp0) covers -Sit5: a, ~d => make it more specific.
Hyp2: b (from Hyp0) covers -Sit3: ~d, b, c => make it more specific.
Hyp3: c (from Hyp0) violates +Sit2: a, b, d => remove it.
Hyp4: ~a (from Hyp0) violates +Sit1: a, b, c, d => remove it.
Hyp5: ~b (from Hyp0) violates +Sit1: a, b, c, d => remove it.
Specializing Hyp1: a (from Hyp0) => 3 new hypotheses (total 9).
Specializing Hyp2: b (from Hyp0) => 3 new hypotheses (total 13).
Hyp10: a, b (from Hyp2) is consistent with all examples
Remember Hyp10 as a good hypothesis.
Hyp11: b, c (from Hyp2) violates +Sit2: a, b, d => remove it.
Hyp12: b, ~a (from Hyp2) violates +Sit1: a, b, c, d => remove it.
Hyp13: ~b, b (from Hyp2) violates +Sit1: a, b, c, d => remove it.
Hyp6: a, b (from Hyp1) is consistent with all examples
However, Hyp10 was discovered earlier and is as general as Hyp6,
so it makes no sense to further specialize Hyp6.
Hyp7: a, c (from Hyp1) violates +Sit2: a, b, d => remove it.
Hyp8: a, ~a (from Hyp1) violates +Sit1: a, b, c, d => remove it.
Hyp9: ~b, a (from Hyp1) violates +Sit1: a, b, c, d => remove it.
All hypothesis are now made as specific as possible.
=== Final hypothesis list: ============================================
Hyp10: a, b (from Hyp2)
========================================================================
```

Figure 11.2: Output of exhaustive general-to-specific (EGS)

4. Algorithm 8 produces all hypothesis that are consistent with the data where in practice we often want the *best* hypothesis, whether or not it is consistent with the data. (If it is inconsistent, this is due to the data and not to the hypothesis itself, since it is impossible to produce a consistent hypothesis on the basis of inconsistent data.)

These disadvantages make that the exhaustive algorithm is merely of theoretical interest. The next algorithm (presented below) is heuristic, and meets all shortcomings mentioned above.

PROBLEMS (Sec. 11.3)

1. Suppose 50 different literals occur in a data file, an suppose we would like to learn a rule for the 50th literal z, say.

   (a) How many different specializations of the rule "$\rightarrow z$" are possible? (Solution.)

   (b) What is the size of the search tree if every antecedent is specialized to 48 literals eventually? (Solution.)

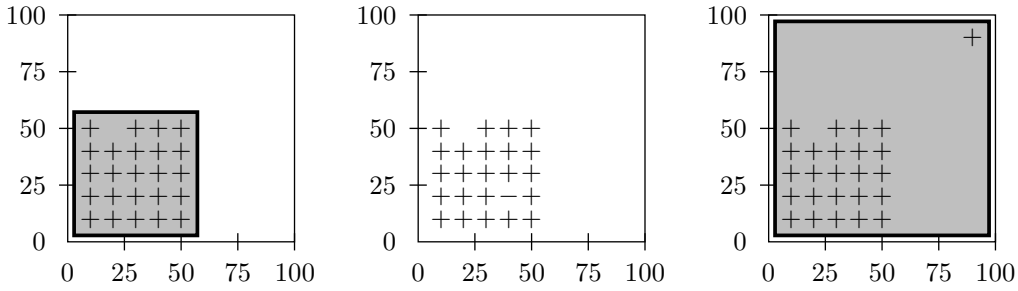


Figure 11.3: Left: data that can be covered by a consistent hypothesis; Middle: data "polluted" with a noisy instance (possibly an error); hence, no consistent hypothesis; Right: noisy positive instance (error?) → forces a general hypothesis. (Hypothesis space 11.2 on page 219 is assumed.)

A heuristic algorithm

The second algorithm does roughly the the same as the first algorithm, with the following differences:

1. Rather than checking for consistency, a *score* is computed for every open hypothesis. There are different score definitions for different purposes, but for the moment the definition

   $$score(H) =_{Def} match(H)$$

   (i.e. the ratio of observed instances that are classified correctly) is fine.
2. Not all open hypothesis are further specialized but only the b best scoring open hypothesis are further specialized. The others are dropped. The number b is fixed number, and is called the *beam-size*. This strategy ensures that the collection of open hypotheses does not grow exponentially, but is always of size b.

The heuristic algorithm begins with the most general rule antecedent as well. This time we do not check whether individual data-instance is classified correctly, but simply compute the score of that particular rule antecedent. We then specialize all antecedents that are open for improvement, and compute their score. An example of this process is displayed in Fig. 11.4 on the facing page. Like Alg. 8, the search starts with the rule with the empty antecedent "$\rightarrow z$". A score is computed and from the figure we can see that the most general hypothesis "$\rightarrow z$" matches 40% of the instances. Then "$\rightarrow z$" is specialized in the same manner as with Alg. 8 on page 222, namely, by adding one literal (does not matter which literal). We see that all (visible) specializations of "$\rightarrow z$" score better than their parent, so search is continued on these hypotheses, provided that the beam-size ≥ 3. (If the beam-size is 2, for instance, then the branch starting with "$c \rightarrow z$" would not be searched.) If every specialization scores less than the parent hypothesis, then the parent apparently cannot be improved, and is put in the list of closed hypothesis. This is the case with "$\neg a, c \rightarrow z$". Search continues until all open hypothesis are closed. Then the best-scoring hypothesis is returned. The entire algorithm is described on page 226.

Algorithm 9 uses a *heuristic* to guide the search. This means that the algorithm assumes that the best hypothesis is a specialization of one of the b best-scoring hypothesis. This assumption is sometimes false. It may happen, for instance, that a dropped hypothesis could have been specialized into a more specific one that is as general but better scoring than the ultimate hypothesis returned by Algorithm 9. This is the price to be paid for carrying out a memory-friendly heuristic search rather than a memory-exhaustive complete search. (Compare this to inviting b applicants for a job interview on the basis of l letters of application ($b < l$). There is a chance you miss out on the best candidate because he/she is not on the top-b list

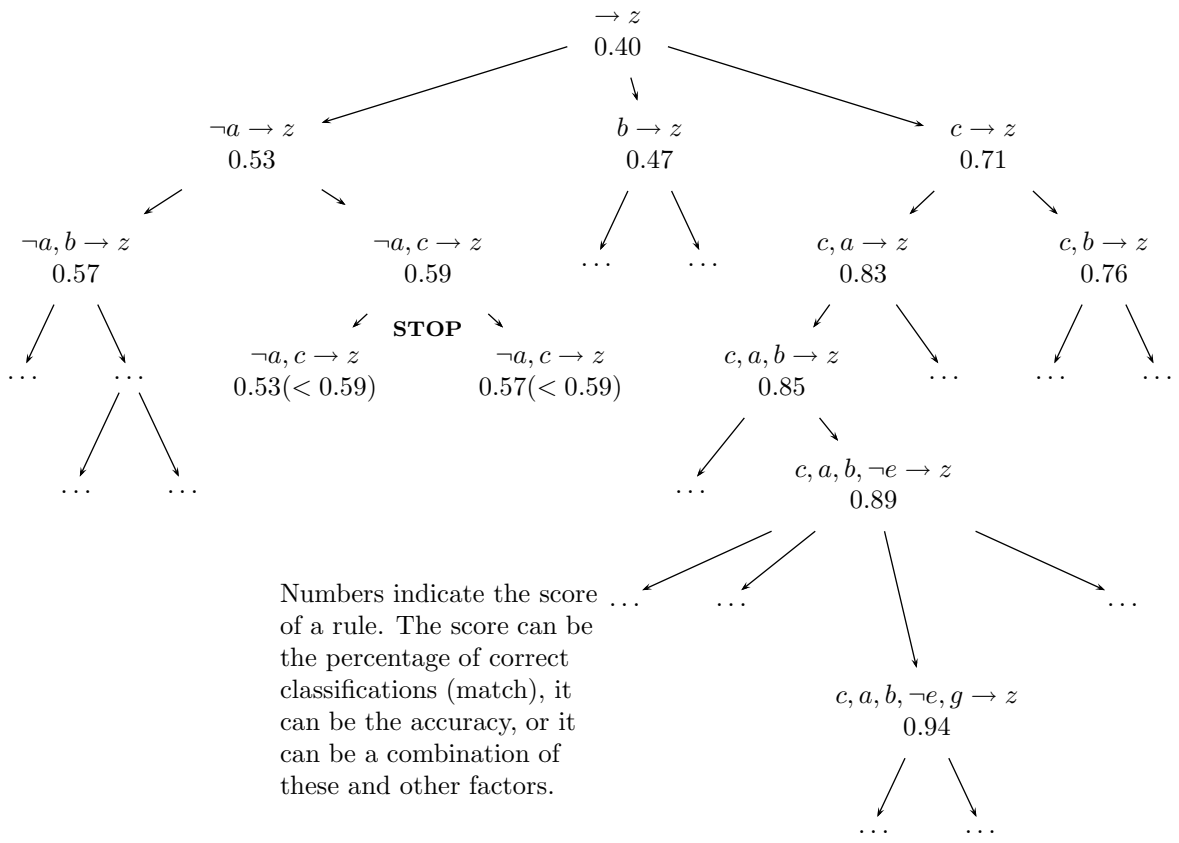


Figure 11.4: Learning one rule by making the most promising antecedent more specific.

because he/she wrote a bad letter.) Experimental studies fortunately suggest that heuristic search almost always leads to the right hypothesis.

Another property of Algorithm 9 is that the final hypothesis does not need to be consistent with the data. This is not necessarily a disadvantage, since it is impossible to produce a consistent hypothesis on the basis of inconsistent data (Fig. 11.3 on the preceding page, 2nd diagram). A best-scoring hypothesis always exists, and the heuristic algorithm will almost always find such a hypothesis. Algorithm 9 also performs well if the data contains noise (Fig. 11.3 on the facing page, 3rd diagram), especially if covering a noisy positive instance would enforce the inclusion of many negative instances. (Why?)

11.4 Learning sets of rules

A disadvantage of the hypotheses space used by Algorithm 8 (page 222) and Algorithm 9 (page 226), is that a hypothesis is identical to precisely one rule, or rule antecedent. In many situations, however, merely one rule antecedent does not suffice to classify all instances accurately. To show why, let us get back to the situation outlined in Section 11.2 on page 219, where instances are points in the plane, and hypotheses are closed rectangles. If the instances are distributed as displayed in Fig. 11.6 on page 229, then the exhaustive algorithm (page 222) would produce no hypothesis, while the heuristic algorithm (page 226) would produce an hypothesis that matches at best 69% of the instances. The second diagram in Fig. 11.6 on page 229 suggests what to do: use a more refined hypothesis representation, in which a hypothesis can be a disjunction (or union) of rule antecedents.

Algorithm 9 Batch-processing of instances to learn the best matching rule antecedent. Search is heuristically guided by the score of a hypothesis.

Input: P, a list of positive instances
Input: N, a list of negative instances
Input: *SCORE*, a subroutine that computes the quality or performance of an hypothesis
Input: b, an integer ≥ 1, representing the beam-size
1: - set O to [null-hyp];
2: - set C to [ ]; # no hypothesis closed, yet
3: **while** O has elements **do**
4: - set S to [ ]; # will contain specializations of O that score better than their parent
5: **for** each h in O **do**
6: S_h := all better-scoring one-step specializations of h
7: **if** S_h is empty **then** # h cannot be improved
8: - put h in C if $h \notin C$
9: **else**
10: - augment S with S_h (remove duplicates, if necessary)
each h in O has now been either closed or specialized
11: - set O to the b best-scoring elements of S # so that O is reduced to a manageable size
O is now empty
12: - **return** the best-scoring element of C;

Learning disjunctions does not require much additional programming effort. Any algorithm can be used to generate the elementary hypotheses (i.e., rule antecedents). Since we have Algorithm 9 (the heuristic algorithm), we use that one. The only thing we will have to beware of is that elementary hypotheses (i.e. rectangles) must be assessed differently if they are used as parts of disjunctive hypothesis. The value of an elementary hypothesis is no longer determined by its performance on all data (Eq. 11.3 on page 220), but rather by its accuracy on the data it covers (Eq. 11.6 on page 220). Other hypotheses, then, are responsible for covering the remaining positive instances. From this observation it would follow that an appropriate score function for hypotheses that are going to be used in a disjunction is their accuracy (Eq. 11.6 on page 220). However, this is too simple an assumption. The problem of this approach is that a hypothesis can be always made more accurate by specializing it, downright to the case in which every rule covers precisely one positive instance. This is not the way to go, however, since such a set of rules is of no use classifying future instances. The problem is that every future instance will be classified as negative, unless it equals a positive instance. This phenomenon is called *overfitting*.

Overfitting is a term from statistics, adopted by the machine learning community. Examples of overfitting:

1. *Given:* A teacher draws a set of points on a piece of paper, approximately lying on a straight line
2. *General concept that lies at the basis of what is manifest:* A straight line.
3. *Overfitting:* Student draws a polygon (line with bends) that goes through all points. This relatively complex solution neglects the more simple notion of straight line.

a. *Given:* Three red objects, shown by parent to 2-year old child.
b. *General concept that lies at the basis of what is given:* The concept "red".
c. *Overfitting:* Child rejects all other red objects as being red, because it believes that "redness" only holds for the three objects that were indicated by the parent.

What happens with overfitting, is that the pupil (or, more generally, the learning algorithm) focuses too much on the examples given, and neglects that there might be a general principle that lies underneath given the examples. The examples are learnt "too well," at the expense of the general structure that is suggested by the hypothesis-representation. Thus, in machine

learning there is always a tradeoff between specializing on the examples on the one hand, and enforcing a more general hypothesis representation on the other hand.

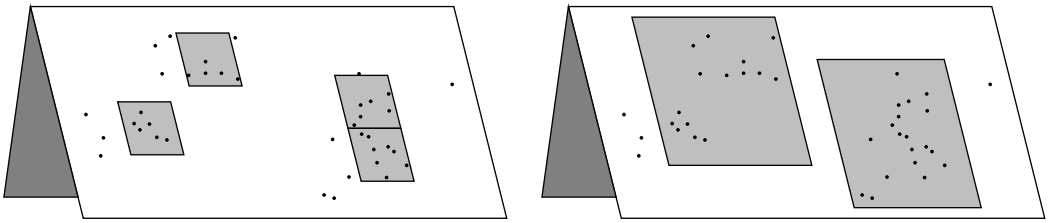


Figure 11.5: Tradeoff in choosing a disjunctive hypothesis. Left: precise but complex covering; Right: simple but imprecise covering.

Here is a practical analogy: Suppose an electric sun-collector has been damaged by hail. In fact, hail stones have damaged some of the collector cells. The sun-collector has about 100,000 of such cells. To avoid a short-circuit in times of rain, most (but not necessarily all) cells must be covered by plastic tissue which is cut in the form of rectangles. A technician is willing to repair the sun-collector. His rate is 50,= Euro for the first patch plus 10,= Euro for every additional patch. This suggests that the application of a small number of large tissues is the best solution. However, spots where tissue is applied do no longer produce electricity. Every cell that is covered but otherwise undamaged, costs 4.68 Euro on a yearly basis. Thus, from the viewpoint of generating electricity, applying several small tissues would be better. This indicates that there is a tradeoff between simplicity and precision. A simple solution with one or two tissues is cheap but diminishes the output of electricity. A complex solution with many small applications maintains electricity output but is expensive. The same holds for formulating a disjunctive hypothesis. There is a tradeoff between an exact covering with many specific (and complex) hypothesis and global covering with only a few simple general hypotheses. The first solution has the advantage of exactness, but the price that has to be paid is overfitting: the hypotheses are tailored at the data, do not represent a global concept, and have little predictive power. The second solution has the advantage of simplicity, but misclassifies a considerable number of observed instances. As so often, the best solution lies somewhere in the middle of two extremes.

Algorithm 10 Batch-processing of instances to learn a set of rules — sequential covering

Input: P, a list of positive instances
Input: N, a list of negative instances
Input: $SCORE'$, a subroutine that computes the performance of an elementary hypothesis
Input: l, a lower bound for what is acceptable as a score
1: - set R to [ ];
2: **while** P has elements **do**
3: - let r be the rule that is produced by Algorithm 9 with parameters $SCORE'$, P and N
4: - leave the while-loop if $accuracy(r) \geq l$;
5: - put r in R;
6: - remove all members of P that are covered by r;
7: - **return** (R, P); # P contains all instances not covered by R

To construct a solution, let

$$accuracy'(H) =_{Def} \begin{cases} accuracy(H) & \text{if } accuracy(H) \text{ is defined,} \\ 0 & \text{otherwise.} \end{cases} \tag{11.7}$$

This ensures that the accuracy of a hypothesis is always defined, so that the algorithm never

stops because of a zero-division. Further, let

$$score(H) = a * accuracy'(H) + b * range(H) + c * coverage(H) \tag{11.8}$$

where $0 \leq a, b, c$ are parameters to regulate the importance, or weight, of aspects of hypotheses such as **a**ccuracy, range (**b**earing), and **c**overage. The definition of range and coverage are given at page 220.

The range of a rule (Definition 11.5 on page 220) may be estimated as follows. With n different proposition letters, for instance $n = 4$ and $L = \{a, \neg a, b, \neg b, c, \neg c, d, \neg d\}$, it is possible to form 3^n different cases, since every proposition letter offers three choices: include the proposition letter in the situation description, include its negation, or include none of them. (For example, L gives rise to $3^4 = 81$ possible cases.) All rules with an empty antecedent cover all cases. All rules with one literal in the antecedent cover one third of all cases, which is 3^{n-1}. In general, all rules with k literals in the antecedent cover 3^{n-k} cases, which means that the range of a rule is divided by three every time the antecedent is specialized by means of one literal. Thus, the range of a rule is proportionally equal to

$$1/3^{|\text{length of the antecedent}|}. \tag{11.9}$$

Now the parameter b can be used to regulate the impact of this quantity on the overall score of a rule antecedent.

Tuning a, b, and c has different effects, which is summarized in the following table:

a	b	c	*rule set produced*
large	small	small	overfitting: many rules, most of them with large (i.e., specific) antecedents; hence, many future instances will be classified negative
small	large	small	few rules, most of them with small (i.e., general) antecedents; many positive instances from the data remain uncovered
small	small	large	few rules, some of which are specific; most of the positive instances from the data are covered

Algorithm 10 on the preceding page works as follows: apply Algorithm 9 (the heuristic general-to-specific algorithm) with the new score function (given by Eq. 11.8) to the set of positive instances P. This yields a rule r_1. Remove all elements that are covered by r_1 from P, and apply Algorithm 9 again. This yields a rule r_2. Repeat this process, until the accuracy of rule r_i falls below some lower bound l. If $l = 0$, then Algorithm 10 goes all the way and may end with an inaccurate rule of inference. (which is not necessarily bad, since all positive examples are covered, be it that some positive instances are covered with inaccurate and/or extremely specific rules). If $l > 0$, then the algorithm ends with $R = \{r_1, \ldots, r_{i-1}\}$ and P containing all positive instances not covered by R. The indexes run to $i - 1$ instead of i, because rule r_i is the last rule that is made by the algorithm. It is not included in R, though, because it is the first rule for which the accuracy is less than l. If $l \rightsquigarrow 1$, R will contain fewer rules.

11.5 Learning a complete set of rules

If you do not know which rules you want, for example, if you try to discover general patterns in the data, then generally you do not know which features exhibit regularities and which not. In such cases it might be a good idea compute rules for *all* literals.

Algorithm 11 on the facing page describes how to produce rules for all literals that possess a certain accuracy.

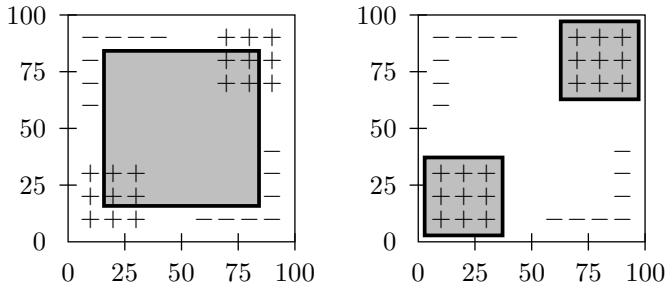


Figure 11.6: Left: best-scoring hypothesis (from hypothesis space defined at Eq. 11.2 on page 219), scores $(8 + 14)/(18 + 14) = 0.69$; Right: better hypothesis but outside hypothesis space, scores 1.00.

Distilling patterns, rules and and regularities from data without knowing what is important is a form of *unsupervised learning*: only with hindsight we can tell which (combinations of) literals exhibit regularities, and which (combinations of) literals do not. For example, for a large data set with literals $\{a, \ldots, z\neg a, \ldots \neg z\}$, the last algorithm may produce

Algorithm 11 Batch-processing of instances to learn all rules for all literals

Input: P, a list of positive instances
Input: N, a list of negative instances
Input: l, a lower bound for rule accuracy
1: - set R_{all} to [ ];
2: **for** each literal L that occurs in the data **do**
3: - augment R_{all} with all rules produced by Algorithm 10 with target literal L;
4: - **return** R_{all}; # all rules are as accurate as l

$$\begin{aligned}
t, \neg h, i, \neg s &\quad \text{–(0.98)}\rightarrow \quad b \\
\neg i, \neg s &\quad \text{–(0.87)}\rightarrow \quad y \\
t, \neg h, e &\quad \text{–(1.00)}\rightarrow \quad e \\
\neg e, n, d &\quad \text{–(0.94)}\rightarrow \quad b \\
\neg o, \neg f &\quad \text{–(0.91)}\rightarrow \quad y \\
\neg t, \neg h, e &\quad \text{–(1.00)}\rightarrow \quad e \\
\neg s, e, \neg c, t, \neg i, \neg o, n &\quad \text{–(0.96)}\rightarrow \quad b,
\end{aligned}$$

assuming that $l = 0.80$. It is only *after* the application of this algorithm, however, that we learn that b, y, and e are the only literals for which learning made sense.

11.6 Rule parametrization

Rule strength and degree of belief (DOB) are important rule properties when a rule is used (rather than created).[2] For example, if

$$\text{DOB}(r_1) = 0.45 \text{ where } r_1 = \text{“}x, y \text{ –(0.91)}\rightarrow z\text{”}$$

then r_1 has DOB 0.45 and strength 0.91. Roughly, the rule strength indicates the certainty with which a rule antecedent implies the consequent, while the support of a rule indicates the

[2] Chapter 8 on page 149.

credibility of a rule in its entirety. (See also Table 11.1.) In this section it is explained how, once that rule is learnt, its strength and support can be defined in terms of accuracy and coverage, respectively.

Strict rules with strict support. Strict rules a-priori, such as: "all circles are round," "the law is the law," "every bachelor is unmarried," as well as inductive rules a-posteriori with massive affirmation: "all restaurants are open between 7-9 PM".	*Strict rules with defeasible support.* Strict rules a-posteriori, for which no counterexamples have been found: "all clerks are nice" (in the sense: "every clerk I've met, was nice"), "all apples are round" (in the sense: "until now, I didn't see apples that were not round).
Defeasible rules with strict support. Defeasible rules a-priori: "rolling a dice usually doesn't produce a six," "most rectangles are not squares (if length and breadth are random variables that are uniformly distributed over $\{1, 2, 3, 4, 5\}$)," as well as inductive rules a-posteriori with massive affirmation: "most birds fly".	*Defeasible rules with defeasible support.* Typically rules a-posteriori: "I've met some boxers, and, apart from one exception, most of them were kind". With enough confirmation, such rules may become strictly supported.

Table 11.1: Rule modalities

How accuracy determines, or at least *indicates*, rule strength can be seen by observing that accuracy is defined as the correct classification ratio among observed instances that are covered. The latter does not say much about the correct classification ratio among observed *and* unobserved instances that are covered by the rule, but since observed instances are by definition all the instances we have, it is reasonable to expect that the rule accuracy will be about the same on unseen instances. (In addition, probability theory is able to indicate the standard deviation of such an estimation, but that would carry us too far here—all we need is one number to fill the slot "rule strength".) Thus, after rules have been learnt, it is reasonable to set rule strength equal to the accuracy of a rule.

How coverage relates to support is less obvious, since coverage is concerned with the range of a rule on seen instances, while support is concerned with the overall credibility of a rule. So what makes a rule more credible than other rules? A simple answer is: by the number of instances that were used to define, or create, the rule. If many instances were used to create a rule, then it reasonable to attach a firm belief to this rule. [Seeing $9,000,000$ people and observing that 49% is male, for example, justifies a high DOB for the rule $human(X) \dashrightarrow_{(0.49)} male(X)$.] Conversely, if few instances were used to create a rule, then it is reasonable to attach a small belief, or degree of support, to this rule in its entirety. However, this approach comes with at least two problems. The first problem is that "the number of instances that were used to create a rule" is an ill-defined concept. Is it the total number of positive and negative instances that were used in the process to form the rule, *or* is it the total number of positive and negative instances that are covered by the rule once it is created?

As long as R covers unseen instances, we have $coverage(R)/range(R) < 1$. In such cases, we cannot be sure about R's true accuracy, because unseen instances in the range of R may change the sampled accuracy, and the latter is the only indication we have for the true rule accuracy.

We may only know the true accuracy of R if all instances in R's range are seen, i.e., if $coverage(R)/range(R) = 1$. In that case, we may read the true accuracy from the sampled accuracy. In all intermediate cases, $coverage(R)/range(R)$ is a fair indicator of the credibility of R's accuracy. Probability theory is able to indicate the confidence interval of the true accuracy, given the coverage and given the sampled accuracy. However, that would carry us too far here—all we need is a number to indicate the credibility of the rule. For the moment, let us say that the credibility of the sampled accuracy—the DOB of R—is equal to the square root of $coverage(R)/range(R)$.

On page 228 it was indicated that $range(R) = 3^{n-k}$, where k is the length of the antecedent of R, n the number of different proposition letters. Thus

$$\begin{aligned}\text{DOB}(R) &= \text{credibility of}(R)\\ &= \text{credibility of sampled accuracy of}(R)\\ &= \sqrt{coverage(R)/range(R)}\\ &= \sqrt{coverage(R)/3^{n-k}}\\ &= 3^{(k-n)/2}coverage(R)\end{aligned}$$

where $coverage(R)$ is the number of data elements that are covered by R.

PROBLEMS (Sec. 11.6)

1. (Learning one rule.) Consider the following fifteen instances:

$$\begin{array}{lll} a,b,c_1,x & a,b,c_6,x & \neg a,b,c_{11},x\\ a,b,c_2,x & a,b,c_7,x & \neg a,b,c_{12},x\\ a,b,c_3,x & a,b,c_8,x & a,\neg b,c_{13},x\\ a,b,c_4,x & a,b,c_9,y & a,\neg b,c_{14},y\\ a,b,c_5,x & a,b,c_{10},\neg x & \neg a,\neg b,c_{15},\neg x \end{array} \tag{11.10}$$

   We use rules to classify cases for x. The score of a rule R is determined by the following formula:

$$score(R) = a * accuracy'(R) + b * range(R) + c * coverage(R) \tag{11.11}$$

   (a) Suppose $a = 1$, $b = 0$ and $c = 0$. Apply the heuristic general-to-specific (HGS) algorithm to learn the best scoring rule for x. (Solution.)

   (b) Same question with $a = 1$, $b = 0$ and $c = 0.023$. (Solution.)

   (c) Suppose $a = 1$ and $b = 0$. For which values of c does the search yield $a, b \rightarrow x$ and $b, a \rightarrow x$? (Solution.)

11.7 Learning first-order rules

In the previous sections we discussed algorithms for learning sets of propositional (i.e., variable-free) rules. In this section, we consider learning rules that contain variables. Our motivation for considering such rules is that they are more expressive than propositional rules. Inductive learning of first-order rules is often referred to as *inductive logic programming* (ILP), because this process can be seen as automatically inferring Prolog programs from instances.

Learning rules from instances that are formulated in the language of first-order logic is not much harder than learning rules from in the language of propositional logic. All methods and

techniques from propositional rule learning carry through to the first-order case. In particular, the exhaustive general-to-specific algorithm (EGS) and the heuristic general-to-specific algorithm (HGS) remain applicable.

A method from propositional rule learning that does *not* carry through to the first-order case, is specializing the antecedent. In the propositional case, a rule antecedent can be specialized by a literal that occurs in the data but does not occur in the antecedent. Since there are a restricted number of such literals, the number of alternative specializations remains within reasonable bounds. In the first-order case, the situation is different. Here, literals are predicates or equalities that may contain terms. Since terms may by arbitrarily large (think of $f(x), f(f(x)), f(f(f(x)))$, the number of alternative specializations may increase exponentially and worse. Therefore, almost all practical algorithms for learning sets of first-order rules works with a first-order language without function symbols. In this way, the number of alternative specializations again remains within reasonable bounds.

When predicates may contain variables, an antecedent can be specialized by predicates and equalities containing variables or constants (but no compound terms). More precisely, suppose the current rule being considered is

$$R: \ l_1, \ldots, l_n \rightarrow px_1, \ldots, x_k$$

where $l_1, \ldots, l_n$ are literals that form the rule antecedent (body) and where $px_1, \ldots, x_k$ is the literal that forms the consequent (head). There are different ways to specialize the antecedent:

1. Add a predicate $qy_1, \ldots, y_m$, where q occurs in the data, and $y_1, \ldots, y_m$ are new variables, or variables that are already present in the rule, such that at least one of the y-variables already occurs in R.
2. Add an equality of the form $x = y$, where both x and y already occur in R.
3. Add the negation of either (1) or (2).

To select the most promising such literal, we will have to compute the score of the extended, or specialized, rule over the test data. This is done by instantiating R in all possible ways with all possible variable bindings for all variables in R. This normally would give a huge amount of rule instantiations, but since we are working with a first-order language without function symbols, and since the data set contains a limited number of constants, this is doable. The next step, then, is to compute the score of all instantiations of R. The score of R itself is then equal to the average of the score of all instantiations.

11.8 Other ways to learn rules

In this chapter, we have presented a number of specific algorithms, that differ in some aspects from other rule learning algorithms. But there are other ways to learn rules.

Some of the choices are:

1. *Batch vs. real-time processing of instances.* Also known as off-line vs. on-line, or as non-incremental vs. incremental processing of instances. In this chapter, we have chosen for batch processing of instances.

2. *General-to-specific vs. specific-to-general development of hypotheses.* In this chapter, we have chosen for moving from general to specific hypotheses. A combination of both search methodes is also possible.

3. *Sequential covering vs. exception-to-exception.* In this chapter, we have chosen for sequential covering.

Chapter 12

Coherence

There are different ways to look at logical data. One is to maintain that not all data make an equal contribution to the justification of beliefs and that observations deserve a special, if not completely privileged, role. This approach is traditionally called the *foundationalist* approach. According to the foundationalist approach, knowledge is a "pyramid" that rests on a foundation, on which further conclusions are built with the help of rules of inference. Most classical logics follow the "foundationalist" approach (but perhaps not intentionally so).

Another approach is to say that there are no fundamental truths and that beliefs are justified as to how far they fit in with other beliefs. This approach is traditionally called the *coherentist* approach [116, 114, 5]. According to this approach, "a body of knowledge is a free-floating raft every plank of which helps directly or indirectly to keep all the others in place, and no plank of which would retain its status with no help from the others" [108].

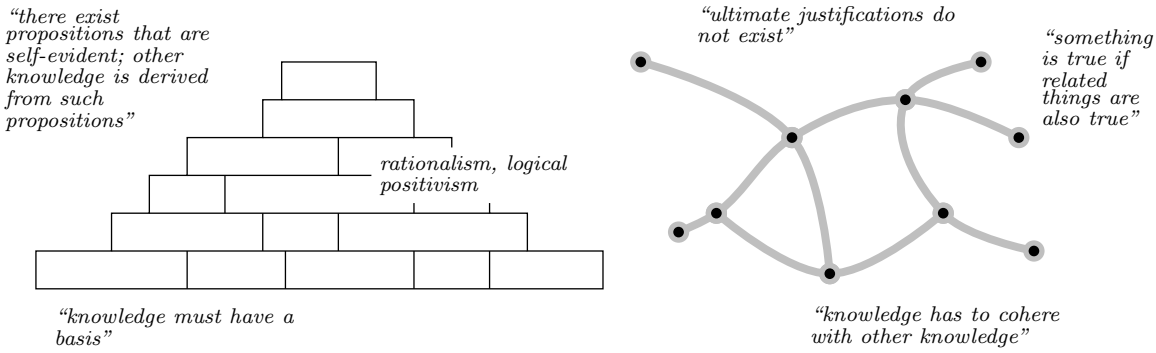


Figure 12.1: The pyramid and the raft.

Great contemporary philosophers have presented compelling arguments in favor of coherentism, and the approach has many followers. During the last ten or twenty years coherentism has become a respected, established and substantiated way of looking at knowledge and belief. Accordingly, epistemologists (philosophers that are concerned with the nature and scope of knowledge) often speak of "the pyramid and the raft" [26, 107, 108].[1]

Realistic data contains inconsistencies. A problem with the foundationalist approach is that inconsistencies are not resolved by the beliefs that are responsible for it. Existing

[1] Sosa (1970): "Repairs must be made afloat, and though no part is untouchable, we must stand on some in order to replace or repair others. Not every part can go at once." (Sosa ascribes the raft metaphor to Neurath, who is ironically enough known to be logical positivist.)

foundationalist-type logics, such as belief revision and argumentation, resolve inconsistencies by giving up existing beliefs or by not accepting all possible lines of argumentation. Thus, existing logics solve the problem either at the beginning or at the end of a line of reasoning. But none of the foundationalist-type logics is designed with the objective to distribute inconsistencies evenly throughout the rules and propositions that are responsible for it. Coherentist-type logics are designed to do just that.

12.1 Programming coherence

Many contemporary programmers of AI systems have expressed sympathy for incorporating ideas of coherence in automated reasoning. The problem is (or rather, was) that, until recently, the nature of coherence was usually left vague, with no method provided for determining whether a belief should be accepted or rejected on the basis of its coherence or incoherence with other beliefs. For a long time, the coherentist view was not amenable to automated reasoning and rather belonged to the domain of armchair philosophers.

Recently, however, the Canadian philosopher of science Paul Thagard proposed how an apparently vague notion of coherence can be specified to such an extent that it can be implemented on a computer. Thagard managed to explicate logical coherence by arguing that an implication $P \rightarrow Q$ establishes a relation of coherency between P and Q: it is reasonable to believe Q once we believe in P, and it is reasonable to believe P once we believe in Q. It is obvious that the second inference is generally much weaker than the first, but if there are no other explanations, or causes, for Q, then it is justifiable, or at least plausible, to accept P. In a similar way we might argue that two contradictory propositions Q and R establish a relation of *in*coherency between Q and R: it is reasonable not to believe R once we believe in Q, and it is reasonable not to believe in Q once we believe in R. Thus, pairs of propositions can cohere (fit together) or incohere (resist fitting together).

Section 12.4 (page 237 and further) deals with Thagard's notion of coherence. The material below is written as a preparation for these sections.

Terminology

Coherence-speak differs somewhat from the logical vocabulary: logic deals with rules and propositions, while coherence programmers speak in terms of networks (Table 12.1).

Logical vocabulary	*Network vocabulary*		
proposition	node, or unit		
accepted proposition	activated node	activation value:	near 1
rejected proposition	de-activated node	activation value:	near −1
two coherent propositions	excitatory link	weight:	near 1
two incoherent propositions	inhibitory link	weight:	near −1
two unrelated propositions	no link, or	link with weight:	0

Table 12.1: Logical vs. network terminology.

Given that a large number of propositions are coherent or incoherent with each other in various ways, the problem is: how we can accept or believe some of these elements and reject or disbelieve others, in a way that maximizes coherence? To answer this question, we have to be more precise about the notions of coherence and acceptability. Further, we must know how to maximize coherence in a network of propositions.

The basic ideas behind maximizing coherence can best be studied by means of the most simple type of coherence, namely, discrete coherence.

12.2 Discrete coherence networks

Discrete coherence is a simple but crude type of coherence. An advantage of studying discrete coherence, is that already exhibits a number of essential features of coherence.

Definition 12.1 (Coherence) i. In the discrete setting, there are three different *activation values* for propositions, namely, -1, 0, and 1. If the activation value of a proposition is 1, we say that it is believed, or *accepted*; if it is -1, we say that it is disbelieved, or *rejected*. The value 0 expresses indifference.

ii. Propositions may be connected by links. Possible link values are -1, 0, and 1.

iii. The *coherence* between two propositions is the product of their activation and the weight of the link that connects them.

- if the coherence between two propositions P and Q is 1, we say that they *cohere* and write $P \sim Q$.
- if the coherence between two propositions P and Q is -1, we say that they *incohere* and write $P \nsim Q$

Note that "$\sim$" is symmetric, and that it is possible that two propositions neither cohere nor incohere.

iv. The (global) *coherence* of a network is the sum of the local coherence values. Global coherence is also named *harmony*, or *goodness-of-fit*.

v. An *optimal solution* is an assignment of activation values that maximizes global coherence.

vi. A *perfect solution* is an assignment of activation values that fulfills every (in)coherence relation.

Some observations:

a. Coherence can be a local matter, or it can refer to the entire constellation of propositions. Items (i)-(iii) are on local coherence, while items (iv)-(vi) are on global coherence.

b. The notion "incoherence" is intended to mean more than just that two propositions do not cohere: to incohere is to *resist* holding together.

c. The global coherency of a network is a *non-standardized measure of coherence*: larger networks usually possess a higher coherency than smaller ones, simply because they have more links.

d. A standardized measure of coherence could be

$$\frac{\text{global coherence}}{\text{coherence of an optimal solution}}$$

Thus, an optimal solution would always have measure one. A problem, however, is that this quotient is difficult to obtain, since optimal solutions are generally not known beforehand.

e. Another standardized measure of coherence could be

$$\frac{\text{global coherence}}{\text{coherence of a perfect solution}}$$

This one is easy to compute, because the coherence of a perfect solution always equals the number of nonzero links in the corresponding graph. The ratio does not necessarily indicate the closeness to the optimal solution as the previous measure would, but it does have the property that the higher the ratio, the closer the solution is to optimal. Thus it gives a size-independent measure of coherence. In addition, if a perfect solution exists both measures coincide.

f. The above definition does not say how to compute the coherence of individual propositions, nor does it say how to compute, or define, the coherence of a subset of propositions in a network. There are two reasons for doing so. Firstly, such a definition is not needed and, secondly, there are different ways in which the coherency of subsets may be defined, but none of them is satisfactory.

Our task is to divide the propositions into ones that are accepted and ones that are rejected, such that global coherency is maximized. How to do that is explained later, in Section 12.5 on page 241. First, we generalize discrete coherence into real-valued coherence networks.

PROBLEMS (Sec. 12.2)

1. Argue that a perfect solution is optimal, but that converse is not necessarily true. I.e., argue that there exist optimal solutions that are not perfect.

2. Let $\mathcal{D}$ be the discrete coherence network with nodes "lion," "tiger," "fish," and "pig", with the following values:

link values:	lion	tiger	fish	pig
lion		1	−1	0
tiger			−1	0
fish				−1

node values:	lion	tiger	fish	pig
	1	1	−1	0

   (a) Does the attribution of link values make sense to you? Why? Or why not? (Solution.)

   (b) Does the attribution of node values make sense to you? Why? Or why not? (Solution.)

   (c) Compute the local coherence between "lion" and "fish". (Solution.)

   (d) Compute the global coherence of the network. (Solution.)

   (e) Change the network so that the global coherency decreases. What have you done? Why? (Solution.)

   (f) What is an optimal solution for this network? Is it unique? (Solution.)

   (g) Does the network possess a perfect solution? (Solution.)

3. Sometimes, perfect coherence cannot be achieved. I.e., it is not always possible to satisfy all (in)coherency relations among a set of propositions. Construct a network in which this is the case, i.e., in which a perfect solution does not exist. (Solution.)

4. Compute, for the general case, the coherence between all different combinations of pairs propositions, weights and activation values. (Due to symmetry only a selection of the $3^3 = 27$ combinations have to be considered.) (Solution.)

12.3 Real-valued coherence networks

Translating logical formulas into a coherence network of propositions requires a notion of coherence that is more refined than what can be offered in a discrete setting. For example, if different logical expressions suggest coherence between one and the same pair of propositions, this should cause more coherence than a situation in which only *one* logical expression suggests coherence. Therefore, a plausible generalization of discrete coherence networks is the following.

Definition 12.2 (Coherence) Coherence is a symmetric, real-valued relation between two propositions. Coherence ranges from -1 (absolute incoherence) to 1 (absolute coherence).

- If P and Q cohere with degree 0.57 we write $P \sim_{0.57} Q$.
- If P and Q incohere with degree 0.23 (or cohere with degree -0.23, which is the same) we write $P \sim_{-0.23} Q$.

This generalization concerns the links between propositions. A second possible generalization is if we make the activation values *themselves* real-valued. In this way we obtain four types of networks:

1. Discrete networks
2. Networks with real-valued links and discrete nodes
3. Networks with discrete links and real-valued nodes
4. Real-valued networks

It might be argued that (1) and (3) are less plausible alternatives because they treat coherency as yes-or-no matter. Type (2) and (4), then, are are more plausible.

Note that coherence networks do not negate propositions, but attach a negative degree belief to propositions. Thus, in coherence networks, propositions are disbelieved rather than negated. More specifically, in Chapters 8 and 10, for example, the DOB ranges from 0 to 1, while in coherence networks, the DOB ranges from -1 to 1. In coherence networks, a strong disbelief in, say, x, would be expressed as $\text{DOB}(x) = -0.9$, while the approach followed in Chapters 8 and 10 would express this as $\text{DOB}(\neg x) = 0.9$. An advantage of the latter approach is that it is more expressive, since we can represent situations in which x is firmly believed as well as firmly disbelieved. I.e., we can represent situations in which $\text{DOB}(x) = 0.9$ and $\text{DOB}(\neg x) = 0.9$. Whether such situations are desirable is another (perhaps philosophical) matter.

12.4 Deriving principles of coherence

One way to define coherence, is to derive it from the rules and propositions given. The idea of derived coherence is of Paul Thagard *et al.* [112, 113, 115, 114, 120].

Five principles of coherence

There are five coherence principles, three of which deal with coherence proper, and two with *in*coherence.

1. *Implication.* Each rule, reason, or implication $P_1, \ldots, P_m \rightarrow Q$ strengthens the coherence between
   - P_i and Q, for each i with $1 \leq i \leq m$.
   - P_i and P_j, for each i and j with $1 \leq i < j \leq m$.

PROCEDURE create network

1. Create nodes for all propositions, plus a node for the proposition `true`.
2. Increase the degree of coherence between all propositions that are coherent according to the implication and analogy principles with a standard amount, say 0.04.[a] (Take into account that the length of the antecedent may diminish a contribution. Further, weights are additive, so that if more than one principle applies, the weights add up.)
3. Decrease the degree of coherence between all propositions that are incoherent according to the contradiction and competition principles with a standard amount, say 0.06.
4. Set the degree of coherence between `true` and data propositions to a small positive value, say 0.05.

The degree of coherence between all other pairs of propositions remains undefined, until some adjustment must be made. In such a case the degree of coherence is set to 0, after which the adjustment is made.

[a] The numbers are more or less arbitrary and are determined from experience.

Table 12.2: Creating a coherence network.

In both cases, the additional strength in coherence is inversely proportional to the length of the antecedent. (Later, we will become more concrete, and provide the details. For now, let us confine ourselves by the principles.)

2. *Analogy.* Each analogy $\langle P_1 \to Q_1, P_2 \to Q_2 \rangle$ strengthens the coherence between
   1. P_1 and P_2
   2. Q_1 and Q_2

   An *analogy* is formed by two implications $P_1 \to Q_1$, $P_2 \to Q_2$, together with an explicit statement that P_1 is analogous to P_2, and Q_1 is analogous to Q_2.

3. *Contradiction.* Each contradiction between two propositions diminishes the coherence between them.

   A *contradiction* is formed by two opposite propositions, e.g., P and $\neg P$.

4. *Competition.* Each form of competition between two propositions diminishes the coherence between them. Two propositions *compete* if they occur in the antecedents of two different rules with identical consequents.

5. *Plausibility of data.* Propositions that are the result of observation, cohere with the special proposition `true`.

These five principles can be used to create a coherence network of propositions (Table 12.2).

At this point, all coherence relations are determined and remain fixed. Our task now is to divide the network into a set of accepted propositions (high activation value) and a set of rejected propositions (low activation value) in such a way that most constraints are satisfied. This is the topic of the next section.

PROCEDURE initialize network

1. Set the activation of `true` to 1, and of all other propositions to a small positive value, say 0.01.

   The value 0.01 can be considered as a seed-value that gives all non-observed propositions initially some benefit of the doubt. The rest of their activation, then, must be obtained from other propositions.

Table 12.3: Initializing a coherence network

Principles of acceptability

Often, acceptance is not a yes-or-no matter. Sometimes, we firmly believe a proposition while at other times we are only prepared to go as far as to give a proposition the benefit of the doubt. A plausible generalization of discrete activation is the following.

Acceptability Acceptability is a quantitative valuation of individual propositions, and ranges from -1 (completely rejected) to 1 (completely accepted).

Truth The special proposition `true` is completely accepted.

Data priority Propositions that describe the results of observation, initially have a degree of acceptability on their own.

Note that the principle says *initially*. Once the network is evaluated, it may happen that data elements turn out not to cohere with, say, firm logical principles, so that they become de-activated rather quickly.

Coherence The acceptability of a proposition in a system S depends on its coherence with other propositions that are accepted in S.

In Table 12.3 is described how to initialize a coherence network.

After all initial activation values have been set, a network typically contains lots of pairs of equally activated but incoherent propositions. This makes most networks globally incoherent. Our job is to make such networks more coherent, that is, our job is to lessen the "logical tension" that exists among different propositions. The whole situation might be seen as a three-dimensional graph, where links between nodes are spiral springs between wooden balls. Some springs are shorter than others. A short spring between two nodes means that the two nodes are coherent (while a long spring means that two nodes are incoherent). Pulling two coherent nodes apart costs energy, and putting two incoherent nodes together costs energy as well. Nevertheless, it may sometimes happen that two incoherent nodes are brought together by other nodes in the network, because the two incoherent nodes both belong to the same coherent clique. Conversely, it may happen that two coherent nodes are pulled apart because they belong to two different groups that are incoherent. Certain 3D-configurations of the nodes cause more tension in the network than other 3D-configurations. The least strenuous configuration is simply obtained by releasing the network, i.e., by letting it loose, so that all nodes assume a position that optimally contribute to the highest possible decrease of tension in the network.

Before it described how to make networks more coherent-, we will first have to generalize the notion of global coherence to real-valued networks, so that we know what we mean when we speak about global coherence.

Global coherence

Like discrete networks, the global coherence of a real-valued network is defined as the sum of the local coherence values:

Example 12.3 If $\mathcal{C} = \{P, Q, R\}$ is a coherence network with links

$$P \sim_{0.98} Q \sim_{0.54} R \sim_{-0.97} P$$

and p, q, and r are the activation values of P, Q, and R, then

$$\mathit{global\_coherence}(\mathcal{C}) = 0.98pq + 0.54qr - 0.97rp \tag{12.1}$$

Here are some examples for different values of p, q, and r:

p	0.00	1.00	-1.00	1.00	1.00	0.98	1.00	1.00
q	0.00	1.00	-1.00	1.00	1.00	1.00	0.98	1.00
r	0.00	1.00	-1.00	0.00	-1.00	-1.00	-1.00	-0.98
global coherence	0.00	0.55	0.55	0.98	1.41	1.37	1.40	1.40

For example, the combination $(p, q, r) = (1.00, 1.00, -0.98)$ yields a relatively high global coherence of 1.40.

Establishing optimal (global) coherence, is called a *coherence problem*.

PROBLEMS (Sec. 12.4)

1. Consider the following information

   - C is a reason for D - A is a reason for B - A is the result of an observation
   - B contradicts D - A, B is a reason for C - $A \rightarrow B$ is analogous to $C \rightarrow D$

   (a) Draw four points A, B, C, D in square form. Compute the coherence of each pair of propositions, based on the implication principles. Draw valued links that correspond to the results.

   (b) Compute the coherence of each pair of propositions, based on the principles of contradiction and modify the values of the links in the graph constructed at (1a) according to the results.

   (c) Same as (1b), but now with competition.

   (d) Same as (1b), but now with analogy.

   (e) Including a special node named "true". (Cf. Table 12.2 on page 238.) and draw the appropriate valued links. I.e., same as (1b), but now with plausibility of data.

   (f) Which of the following three valuations *Val1*, *Val2*, *Val3* yields the highest (global) coherence of the network?

	Val1	*Val2*	*Val3*
A	0.98	−0.68	−0.23
B	0.18	−0.33	−0.46
C	−0.21	0.81	0.98
D	0.95	−0.12	−0.87

2. (a) Create a format in which it is possible to express rules, observations, contradictions and analogies in ASCII. Represent the data from (1) in this new format. (Solution.)

(b) Write a subroutine that is able to read the ASCII-format, and forms the corresponding data-structures.

(c) Write a another subroutine that is able to create a coherence network on the basis of the rules, observations, contradictions and analogies that are parsed and internally represented by the first routine. (Hint: competition and analogy are the hardest principles.)

12.5 Computing acceptance

In principle, the way in which a coherence problem is solved is not important, as long as it is reliable and computationally efficient. Unfortunately, a coherence problem in its general form is NP-complete, which means that is extremely likely, and in fact almost certain, that no algorithm exists that solves every coherence problem in a reliable and computationally efficient way. Fortunately, there exist algorithms that are either reliable (at the price of computationally efficiency), or computationally efficient (at the price of reliability).

In this section we describe five algorithms for maximizing coherence:

1. An *exhaustive* search algorithm that considers all possible solutions;
2. An *incremental* algorithm that considers elements in arbitrary order;
3. A *greedy* algorithm that uses locally optimal choices to approximate a globally optimal solution;
4. A *semidefinite programming* (SDP) algorithm that is guaranteed to satisfy a high proportion of the maximum satisfiable constraints;
5. A *connectionist* algorithm that tries to achieve global coherence by improving local coherence repeatedly.

The first two algorithms are of limited use, but the others provide effective means of computing coherence.

Algorithm 1: Exhaustive

The obvious way to maximize coherence is to consider all the different ways of accepting and rejecting elements:

1. Generate all possible ways of dividing elements into accepted and rejected.
2. Evaluate each of these for the extent to which it achieves coherence.
3. Pick the one with highest global coherence.

There are several problems with this approach. The first problem is that it only works for discrete networks. Even for one node in a continuous network it is impossible to try out every activation value between -1 and 1. The second problem is that the number of possible acceptance sets in discrete networks, though finite, grows exponentially. A small coherence problem involving only 100 propositions would require considering $2^{100} = 1,267,650,600,228,229,401,496,703,205,376$ different solutions. No computer, and presumably no mind, can be expected to compute coherence in this way except for trivially small cases.

In computer science, a problem is said to be intractable if there is no polynomial-time solution to it, i.e. if the amount of time required to solve it increases at a faster-than-polynomial rate as the problem grows in size. For intractable problems, the amount of time and memory space required to solve the problem increases rapidly as the problem size grows. Consider, for example, the problem of using a truth table to check whether a compound proposition is consistent. A proposition with n connectives requires a truth table with $2n$ rows. If n is small, there is no

difficulty, but an exponentially increasing number of rows is required as n gets larger. Problems in the class NP include ones that can be solved in polynomial time by a *nondeterministic* algorithm that allows guessing.

Members of an important class of problems called NP-complete are equivalent to each other in the sense that if one of them has a polynomial time solution, then so do all the others. A new problem can be shown to be NP-complete by showing (a) that it can be solved in polynomial time by a nondeterministic algorithm, and (b) that a problem already known to be NP-complete can be transformed to it, so that a polynomial-time solution to the new problem would serve to generate a polynomial-time solution to all the other problems. If only (b) is satisfied, then the problem is said to be NP-hard, i.e. at least as hard as the NP-complete problems. In the past two decades, many problems have been shown to be NP-complete, and polynomial-time solutions have been found for none of them, so it is widely believed that the NP-complete problems are inherently intractable. An excellent introduction to the theory of intractability is [30].

Millgram [68] noticed that the problem of computing coherence appears similar to other problems known to be intractable and conjectured that the coherence problem is also intractable. He was right. In 1998, Verbeurgt *et al.* [120] showed that MAXCUT, a problem in graph theory known to be NP-complete, can be transformed to a discrete coherence problem. If there were a polynomial-time solution to coherence maximization, there would also be a polynomial-time solution to MAXCUT and all the other NP-complete problems. So, on the widely held assumption that P $\neq$ NP (i.e. that the class of problems solvable in polynomial time is not equal to NP), we can conclude that the general problem of computing coherence is computationally intractable. As the number of elements increases, a general solution to the problem of maximizing coherence will presumably require an exponentially increasing amount of time.

For epistemic coherence and any other kind that involves large numbers of elements, this result is potentially disturbing. Each person has thousands or millions of beliefs. Epistemic coherentism requires that justified beliefs must be shown to be coherent with other beliefs. But the transformation of MAXCUT to the coherence problem shows, assuming that P $\neq$ NP, that computing coherence will be an exponentially-increasing function of the number of beliefs.

Algorithm 2: Incremental

Here is a simple, efficient serial algorithm for computing coherence:

1. Take an arbitrary ordering of the propositions $P_1, \ldots P_n$
2. Let A and R, the accepted and rejected elements, be empty.
3. For each proposition P_i,

   > Add P_i to either A or R, depending on the contribution to the global coherence this would have.

This approach also involves several problems. The first problem is that the algorithm only works for discrete networks. The second (and major) problem with this algorithm, however, is that seriously depends on the ordering of the elements. Suppose we have four elements, such that there is a negative constraint between P_1 and P_2, and positive constraints between P_1 and P_3, P_1 and P_4, and P_2 and P_4 (Figure 12.2 on the facing page). In terms of explanatory coherence, P_1 and P_2 could be thought of as competing hypotheses, with P_1 explaining more than P_2. The three other algorithms for computing coherence discussed in this section accept P_1, P_3, and P_4, while rejecting P_2. But the serial algorithm will accept P_2 if it happens to come first in the ordering. In general, the serial algorithm does not do as well as the other algorithms at satisfying constraints and accepting the appropriate elements.

Although the serial algorithm is not attractive as an account of how coherence should be computed, it may well describe to some extent people's limited rationality. Ideally, a coherence inference should be nonmonotonic in that maximizing coherence can lead to rejecting elements

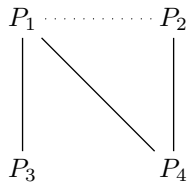


Figure 12.2: Coherence network. (Dotted lines represent negative constraints.)

that were previously accepted. In practice, however, limitations of attention and memory may lead people to adopt local, suboptimal methods for calculating coherence. (Cf. [44].) Psychological experiments are needed to determine the extent to which people do coherence calculations suboptimally.

Algorithm 3: Greedy

In computer science, a *greedy* algorithm is one that solves an optimization problem by making a locally optimal choice intended to lead to a globally optimal solution. Selman, Levesque, and Mitchell (1992) presented a greedy algorithm for solving satisfiability problems, and a similar technique produces the following coherence algorithm. The algorithm starts with a randomly generated solution and then improves it by repeatedly flipping elements from the accepted set to the rejected set or vice versa.

1. Randomly assign the propositions into A or R.
2. For each proposition, calculate the gain (or loss) in the weight of satisfied constraints that would result from flipping it, i.e., moving it from A to R if it is in A, or moving it from R to A otherwise.
3. Produce a new solution by flipping the element that most increases coherence, i.e. move it from A to R or from R to A. In case of ties, choose randomly.
4. Repeat 2 and 3 until there is no flip that increases coherence. or until a maximum number of tries have taken place.

On the examples on which Verbeurgt *et al.* [120] tested this algorithm, it produced the same result as the connectionist algorithm, except that the greedy algorithm breaks ties randomly. With its use of random solutions and a great many coherence calculations, this algorithm seems less psychologically plausible than the connectionist algorithm.

Algorithm 4: Semidefinite programming

The proof that the graph theory problem MAXCUT can be reduced to the coherence problem shows a close relation between them. MAXCUT is a difficult problem in graph theory that until recently had no good approximation: for twenty years the only available approximation technique known was one similar to the incremental algorithm for coherence we described above. This technique only guarantees an expected value of 0.5 times the optimal value. Recently, however, Goemans and Williamson [34] discovered an approximation algorithm for MAXCUT that delivers an expected value of at least 0.87856 times the optimal value. Their algorithm depends on rounding a solution to a relaxation of a nonlinear optimization problem, which can be formulated as a semidefinite programming (SDP) problem, a generalization of linear programming to semidefinite matrices. Mathematical details are given in [34, 120].

What is important from the perspective of coherence are two results, one theoretical and the

other experimental. Theoretically, Verbeurgt *et al.* prove that the semidefinite programming technique that was applied to MAXCUT can also be used for the coherence problem, with the same 0.878 performance guarantee: using this technique guarantees that the weight of the constraints satisfied by a partition into accepted and rejected will be at least 0.878 of the optimal weight. But where does this leave the connectionist algorithm which has no similar performance guarantee? Verbeurgt *et al.* have run computational experiments to compare the results of the SDP algorithm to those produced by the connectionist algorithms used in existing programs for explanatory and deliberative coherence. Like the greedy algorithm, the semidefinite programming solution handles ties between equally coherent partitions differently from the connectionist algorithm, but otherwise it yields equivalent results.

Algorithm 5: Connectionist

The connectionist way to maximize global coherence, is to improve on local coherence at every node repeatedly, thus speculating on the effect that a large number of such local improvements contribute to global coherency.

To illustrate this approach, let us consider the network $\mathcal{C}$ from Ex. 12.3 on page 240 again. The global coherence of that network depends on the activation values p, q and r, as follows:

$$\mathit{global\_coherence}(\mathcal{C}) = 0.98pq + 0.54qr - 0.97rp$$

(Recall that the numbers p, q and r stand for the activation values of the nodes P, Q and R.) To achieve global coherence, the idea is to start out with small activation values. Then, the network is run in cycles where each cycle changes the activation of every node a tiny bit, with the objective to improve global coherence.

To determine how activation values must change, we use differential calculus and formulate the derivative of global coherence, relative to the activation values. For p, for instance, the derivative is

$$\frac{\partial \mathit{global\_coherence}(\mathcal{C})}{\partial p} = \frac{0.98pq + 0.54qr - 0.97rp}{\partial p} = 0.98q - 0.97r$$

Let us abbreviate this by n_p. Since n_p is also the through coherency weighted sum of the activation values of the neighbors of P, it is often called the *nett input* to P. (Hence, the "n".) Similarly, $n_q = 0.98p + 0.54r$ and $n_r = -0.97p + 0.54q$.

For the moment, let us stick with node P. It follows from differential calculus principles that, to improve P's contribution to global coherence, we will have to increase p with it's nett input, n_p, or at least a fraction of n_p. So $p + n_p$ is a good first candidate. The problem with $p + n_p$, however, is that if P has many neighbors, its activation will be dominated by the activation of its neighbors. To remedy this, we use the *average nett input*, defined as

$$\bar{n}_p =_{Def} \frac{n_p}{\text{number of links emanating from } p} \tag{12.2}$$

In this way $-1 \leq \bar{n}_p \leq 1$, so that $\bar{n}_p$ is of the same order of magnitude as p's activation.

There is another problem. If we increase p with $\bar{n}_p$, it may happen that $p + \bar{n}_p$ may lie outside the interval $[-1, 1]$. This is a problem, since activation values are supposed to stay between -1 and 1. Therefore, we reside to the so-called *hyperbolic sum*, defined by $x \oplus y =_{Def} (x + y)/(1 + xy)$. This sum behaves like ordinary addition (it is commutative, for example) except that the hyperbolic sum ensures that the result of an addition stays within $[-1, 1]$. Thus, if $-1 \leq x, y \leq 1$, then $-1 \leq x \oplus y \leq 1$. Thus, $p \oplus \bar{n}_p$ would be another good candidate to update P's activation.

Still, this new update formula is not yet correct. Neighbor-based updating is a circular process that never stabilizes if all nodes just do a mutual update at each cycle. In that case, they keep "sending" their values back and forth with the nett effect of merely swapping their activation values. To gradually dampen the update process, let us carry through a final change in the

PROCEDURE harmonize network

REPEAT

Update the activation values of all nodes synchronously, as described by Eq. 12.3.

UNTIL the network has stabilized, or some (predefined) number of cycles has passed.

Table 12.4: Harmonizing, or *relaxing*, a coherence network

definition of P's update. (A change for the better, of course.) The change amounts to the fact that we stipulate that the influence of P's activation value p should diminish over time. This is the whole idea behind harmonizing a network: start out with initial activation values, and then turn them loose gradually for the sake of reaching global coherence. To introduce this effect, we use a small number, d, called the *decay factor*. The decay factor ensures that the influence of p in the first update is $p(1-d)$, in the second update $p(1-d)^2$, in the third update $p(1-d)^3$, and so forth. The final definition of p's update thus becomes

$$p_{new} =_{Def} (1-d)p \oplus \bar{n}_p. \tag{12.3}$$

and similarly for q's update and r's update (Table 12.4)[2] One way to consider a network of activation values as stabilized, is when the maximum change of activation values drops below a (predefined) threshold value. There are other ways, such as taking the *average* change of activation values. An advantage of the latter metrics is that the algorithm stops, even in the presence of a small subset of propositions that fails to converge to fixed values (a so-called a chaotic clique).

PROBLEMS (Sec. 12.5)

1. Let $x \oplus y$ be the hyperbolic sum defined by $(x+y)/(1+xy)$. For which values of $-1 \leq x, y \leq 1$ is $x \oplus y$ undefined? For which values is the hyperbolic sum equal to 1? Show that the hyperbolic sum is commutative, i.e. $x \oplus y = y \oplus x$. Show that the hyperbolic sum is associative, i.e. $x \oplus (y \oplus z) = (x \oplus y) \oplus z$. Write out $a \oplus b \oplus c \oplus d$. (Solution.)

2. (a) Write a subroutine with formal parameters a, b, and c that creates a random network with at least a nodes and at most b nodes, such that every node has at least one and at most c neighbours, and such that coherence links have random values between -1 and 1.

   (b) Write a subroutine that computes the global coherence of the type network created at (2a).

   (c) Write an update routine that updates each node according to Eq. 12.3.

   (d) Write a routine that cycles through the update routine until the difference in global coherence between two rounds becomes less than, say, 0.001.

   (e) i. Experiment with decay factors $\neq 0.05$.

   ii. See what happens if a node P with activation value p is updated with the nett input n_p instead of the average nett input $\bar{n}_p$.

[2] Thagard [113] uses an update function different from (12.3). This update function is more complicated, since it requires a case distinction between $p \geq 0$ and $p < 0$. Moreover, experiments have shown that (12.3) causes networks to stabilize faster.

iii. See what happens if a node P with activation value p is updated with normal addition $+$ rather than with the hyperbolic sum $\oplus$.

iv. Experiment with alternative update routines.

Chapter 13

The big picture

The aim of this chapter is to "put it all together". One thing we might do is to take a closer look at the differences between the various algorithms. With ATP, for example, we might compare algorithms with respect to effectiveness, efficiency (with respect to time and memory), conceptual transparency, and robustness.

For non-deductive automated reasoning
the analysis becomes even more interesting, because different non-deductive reasoners may, in principle, return different answers. Thus, non-deductive algorithms might additionally be compared with respect to the answers they give.
This is an important difference with deductive reasoning.

Another thing we might do is to take a step
step backwards to put matters in perspective. In this way, we might see how different algorithms could be integrated into something more powerful. We do this in Section 13.1.

The chapter ends
with an overview of the prospects of automated reasoning, considered from a technical point of view as well as from a user-oriented point of view. This is done in Section 13.3

Figure 13.1: John Fox.

13.1 Analysis

In 1986, the English psychologist, logician and Artificial Intelligence researcher John Fox (Figure 13.1)[1] wrote an article in the proceedings of the First Annual Conference on Uncertainty in Artificial Intelligence, at that time a ground-breaking initiative to collect new views of researchers who pioneered the new grounds of probability and AI. The article is entitled "Three Arguments for Extending the Framework of Probability," and is more a position paper than a meticulously exposed scientific theory [28]. In this article Fox makes a strong case out of the fact that different problems require different uncertainty calculi. Particularly strong is the following analogy:

> We can illustrate what is meant by a logical distinction [between different uncertainty calculi] by considering the less contentious subjects of place and time. Place can be

[1] I am aware of the fact that supplementing the picture of a contemporary scientist to a gallery of historic icons, might suggest a bad sense for historic proportions. This is by no means my intention. By including a contemporary researcher in my gallery I would like to show that science is the work of humans who are essentially all alike. There are no historic heroes.

> represented by means of $(x, y, z,)$-coordinates in Euclidian space but, for much problem solving, qualitative terms can be more useful. It can be convenient to distinguish important places like *here* and *there* and more complex ones like *somewhere*, *elsewhere*, and *everywhere*. Time is normally represented by reference to a continuous state as well, but we have a complementary qualitative vocabulary which distinguishes concepts like *now*, *sometime*, *never* and relations like *before*, *after*, *during*, and so forth. The formal nature of these languages of place and time is unclear. "Here" is a subjective concept while "everywhere" is not; "now" is an instant, while "never" is an unbounded interval. However, the terms reflect real distinctions which are complementary to, not rivals of, quantitative representations. (Cited from [28, p. 449].)

Here is another analogy: for a long time, widget programming in graphical user interfaces (GUI's) was a drag. You had control over every little detail, but then again you had to control every little detail: first you had to determine the dimensions of the largest menu item, then you had to take account over the pixels above and below the menu item, then you had to estimate the width of the menu window based on the current font, the absolute coordinates (in pixels) of the left-upper corner of the window, the size (in pixels) of the window—all in absolute (x, y)-coordinates of course. (Note the analogy with probabilistic reasoning!) Later, so-called "packers" or "sizers" became the method of choice in GUI programming. This was much easier since you had only to specify the order of the widgets. After that, the window manager did the rest for you and took care of the exact layout. Thus, with GUI programming, do not always need access to the lowest level (read: every pixel) in order to build visually appealing user interfaces.

Also in the domain of uncertainty, different problem descriptions require different sizes of granularity, depending on the type, quality and precision of the information that is at hand. Sometimes we need exact numbers, at other times we can do (or have to contend) with nominal values (Figure 13.2). Furthermore, the choice of the formalism depends on how much information on the problem is available. Consequently, there is no "best calculus".

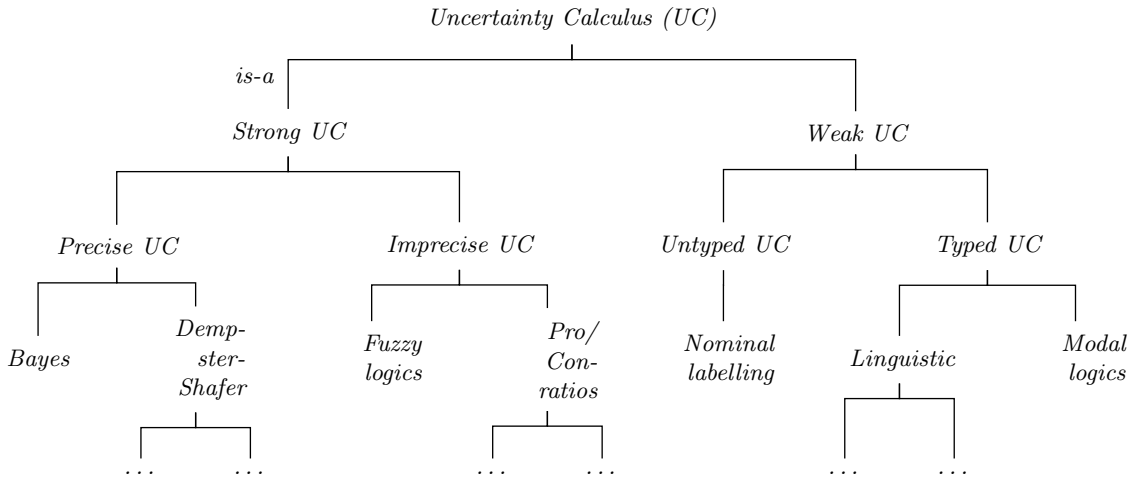


Figure 13.2: A space of methods for managing uncertainty [28, p. 457].

See also the different historical motivations and justifications behind probability that are outlined in Hacking's *"The Emergence of Probability"* [40]. Although Hacking's work does not deal with automated reasoning and does not allude to it, it is interesting to note that Hacking ascribes a major role to G.W. Leibniz in the development of probability: "Leibniz has been our constant witness to events in probability from 1665 until 1713. He was the first philosopher of probability and anticipated, often in great detail, many of our probabilistic conceptions" [40,

p. 185]. Together with Leibniz' ideas on automated reasoning (as briefly described on p. 7), this indicates to me that automated reasoning and uncertainty are tightly interwoven and that probability is just one out of many calculi that can be used to deal with uncertainty.

Different answers

Another point of view might be that different non-deductive automated reasoners may produce different answers (at otherwise identical cases). On the one hand, this can been seen as a disadvantage, in the sense that "only one answer can be right, so that all other answers must be wrong". On the other hand, the fact that different non-deductive automated reasoners may produce different answers can also be put to use in situations where careful decision-making is of the essence.

Example 13.1 (A panel of automated reasoners) Suppose we have ten automated reasoners: three argument-based reasoners, four probabilistic reasoners and five reasoners based an coherence—a bit like Tricolore's pizza menu card:

i. Bayesian reasoner (sum normal)
ii. Bayesian reasoner (sum fast retract)
iii. Bayesian reasoner (max normal)
iv. Probabilistic reasoner (Dempster-Shafer)
v. Probabilistic reasoner (certainty-factors)
vi. Probabilistic reasoner (subjective probabilities)
vii. Argument-based reasoner (local propagation of conclusive force)
viii. Argument-based reasoner (with defeat among global arguments)
ix. Argument-based reasoner (with defeat among global arguments)
x. Coherentist reasoner (á la Thagard)
xi. Coherentist reasoner (direct coherence)

Suppose further that the problem issued to these reasoners is: "shall I accept the job offer or not?" Then we may imagine a situation where seven out of eleven modules say "yes," two say "don't know, "while the remaining two say "don't do it". In this respect an interesting graduate project would be to write tools that convert input files between the formats of different uncertainty reasoner so that one problem can be submitted to different reasoners and the different answers can be compared.

13.2 Synthesis

A tool or technique can be interesting in its own right. However, the reason why the tool or technique came into existence, was probably because someone encountered a difficulty in the process of solving a more practical problem. (A good example are propositional formulas that arise out of railway interlocking safety conditions.) When the sub-problem has been solved in a way that suffices for practical purposes, we can take "a step back" and see whether the newly developed technique can be combined with other automated reasoning tools and/or techniques.

The following example is meant to demonstrate how material from the preceding chapters may be combined into "something more powerful".

Example 13.2 (From raw data to "intelligent" discussions) Anyone who has worked in the knowledge-engineering industry, or anyone who has followed a class in knowledge-acquisition, knows that knowledge-acquisition is a difficult, non-trivial, and often strenuous process. One possible reason why knowledge-acquisition is hard, is that causal patterns, social laws, or legal

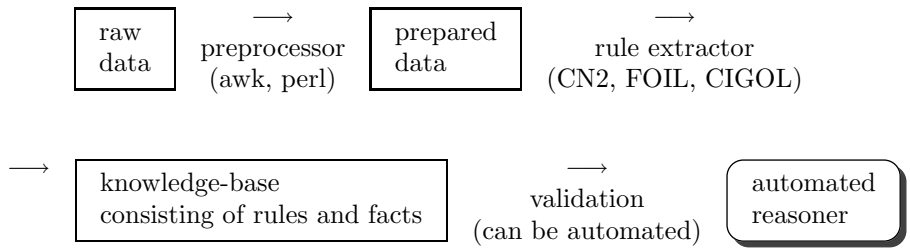


Figure 13.3: Piping raw data trough various preprocessors

procedure are subjective interpretations of the expert and must be interpreted in co-operation with the knowledge-engineer. The knowledge-engineer, then, has the formidable task to put vague knowledge into the straight-jacket of a particular knowledge representation language. Thus, eliciting rules is difficult, and requires a skill to discover regularities in the data that is available. The reason why knowledge-acquisition nevertheless pays off is that knowledge becomes less ambiguous, less vague, less private, more consistent, more accessible and more stable. Once it is put out in the open it can, in principle, be understood and used by anyone who has access to the computer.

A precondition of intelligent reasoning is the existence of rules (or reasons) to form arguments. Sometimes, such rules of inference do exist, and can be used. At other times, rules do not exist, and must be produced. To produce rules of inference, we need cases, or instances, from which rules can be learnt (Chapter 11). If we are lucky, decisions about previous cases are stored (in files, or databases, for example) so that we may use them. At other times, cases do not exist, and must be produced, if possible. One way to produce cases is by processing raw data into something that can be handled by algorithms that learn rules.

If we succeed in doing this, we may use the computer to help us to bring order in large and/or unstructured data-sets, by letting it produce arguments for and against particular decisions, or by letting it produce counterarguments, explanations or other rational reconstructions of claims. (End of the example.)

13.3 Technical prospects

1. Selman's challenge's.
2. Better understanding of relation between, e.g., Bayesian belief networks and argumentation.
3. Propositional coherency.

13.4 User-oriented prospects

The purpose of this section is to defend the logical-symbolic approach in the context of alternative approaches such as connectionism and other biologically inspired alternative computational paradigms such as embodied intelligence and artificial life. To this end, we enumerate some advantages that logical-symbolic approaches have over connectionistic approaches.

One important advantage of the logical-symbolic approach is that all reasoning is transparent. For example, proofs in OTTER can always be verified by hand. Transparency is beneficial, because we human beings are such that we like to be kept informed about what is going on

Personal assistant:	I just came to the conclusion that it is better for you to have brain surgery.
You:	???
Personal assistant:	Trust me, I'm a cyborg Bioplant type `cb-675-CYB-bio-12a`. Bioplant's type `cb-675-CYB-bio-12a` are especially good at medical diagnosis. I can explain that, if you want.
You:	No thanks, I know—and I do trust you on this issue. Still, brain surgery, I don't know ... it sounds rather radical to me. Please tell me a bit more about it. I mean, why do you think I should have brain surgery?
Personal assistant:	I wish I was able to tell you, but I can't. My neural net simply says it is better for you to have it done. Let's call it ... "intuition".
You:	...
Personal assistant:	Shall we?
You:	...

Figure 13.4: Dialogue between human being and personal assistant.

inside the machine. We would like to understand what is happening, and we would be able to reconstruct and understand decisions ourselves. We want to keep in control.

As opposed to the transparency of logical calculi, connectionistic approaches are famous for their lack of transparency. Conclusions are often produced without explanation, and in many cases the "reasoning" process was non-deterministic so that conclusions are not even reproducible. (Cf. the work on GSAT on page 127.) It is not hard to imagine that the absence of justifications for serious or life-critical decisions may become problematic, such as for instance in the dialogue displayed in Figure 13.4.

With this dialogue, I'd like to conclude the reader. Thanks for bearing with me.

Appendix A

WinKE

Installation

Download `http://www.cs.uu.nl/docs/vakken/ar/software/wke599.exe` and follow the instructions in the README.

Note: system administration has already installed the WinKE-font on the student computers.

README

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                           *
*   WinKE -- a KE based pedagogic tool for teaching logic and reasoning     *
*                                                                           *
*                          beta release: ver 05/99                          *
*                                                                           *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


1) Installation
===============

The file wke599.exe is a self-extracting archive. You should extract
it directly to your C: drive. This will create a folder "WinKE" with
the program files in it.

After that you have to install the WinKE-font on your system. Under
Windows 9x this is usually done by copying WinKE.TTF to the directory
C:\Windows\Fonts. For Windows NT this directory is called C:\Winnt\Fonts.

Use the shortcut WinKE to start the program.

(The command line should be: C:\WinKE\WinKE.EXE /V1 /T512 /H512.
The value after the command line switch /H denotes the heap space
available to WinKE. You might want to try to increase this value
when working on rather large problems.)


2) World Wide Web
```

```
=================

Several papers on WinKE and the KE calculus can be found
on our web site:

http://www.dcs.kcl.ac.uk/~endriss/WinKE/


3) Textbook
===========

For further information on the KE calculus you may want to consult
the following textbook:

Marco Mondadori and Marcello D'Agostino. Logica.
Edizioni Scolastiche Bruno Mondadori, Milan 1997.


4) Contact
==========

If you have any problems using the WinKE software, if you have
questions, or if you would like to make comments, please email
Ulrich Endriss on endriss@dcs.kcl.ac.uk.
```

Appendix B

Otter

Otter is a first-order-logic-with-equality theorem prover. Its inference rules are based on resolution and paramodulation, and it includes facilities for term rewriting, term orderings, Knuth-Bendix completion, weighting, and strategies for directing and restricting searches for proofs.

Installation

Source, [Win32,Mac,...]-binaries and documentation are available at `http://www.mcs.anl.gov/AR/otter`.

Unlike WinKE, Otter is a command-line application that communicates via standard input and standard output (and standard error). Therefore, a typical invocation of Otter looks like

```
otter < input_file > output_file
```

The program does not have command-line switches, all options must be set in the input file.

- There are HTML-ed versions of Otter's Reference Manual out there on the web. Get one if if you plan to work with Otter. While working with Otter it is extremely convenient to have (a searchable version of) the ref open in another window.
- An excellent companion to Otter is Kalman's "Automated Reasoning with Otter" [48].

Solutions to selected exercises

1, p. 26: Here is a proof of $p \supset p$ in Hilbert's system:

1. $(p \supset ((p \supset p) \supset p)) \supset ((p \supset (p \supset p)) \supset (p \supset p))$	(Instance of Axiom Schema 2)
2. $p \supset ((p \supset p) \supset p)$	(Instance of Axiom Schema 1)
3. $(p \supset (p \supset p)) \supset (p \supset p)$	(From 1, 2 by MP)
4. $p \supset (p \supset p)$	(Instance of Axiom Schema 1)
5. $p \supset p$	(From 3, 4 by MP)

2a, p. 26: Because Hilbert's systems consist of three axiom schema's (rather than three axioms). These three schema's would make the set of axioms A look like

$$
\begin{aligned}
A = \{\, & p \supset (p \supset p),\\
& q \supset (q \supset q),\\
& p \supset (q \supset p),\\
& q \supset (p \supset q),\\
& p \supset (r \supset p),\ \ldots\\
& (p \supset (p \supset p)) \supset ((p \supset p) \supset (p \supset p)),\\
& (p \supset (q \supset p)) \supset ((p \supset q) \supset (q \supset p)),\ \ldots\\
& (\neg p \supset \neg p) \supset (p \supset p),\\
& (\neg q \supset \neg q) \supset (q \supset q),\\
& (\neg p \supset \neg q) \supset (p \supset q),\\
& (\neg p \supset \neg r) \supset (p \supset r),\ \ldots\}
\end{aligned}
$$

i.e, A would be infinite.

2b, p. 26:

set of conclusions	*applied rule of inference*	*new set of conclusions*
$\{\neg c, e\}$	$e, \neg c \to a$	$\{a, \neg c, e\}$
	$a, e \to d$	$\{a, \neg c, d, e\}$
	$d, \neg c, e \to \neg f$	$\{a, \neg c, d, e, \neg f\}$
	$\neg f \to b$	$\{a, b, \neg c, d, e, \neg f\}$
	$a, b \to t$	$\{a, b, \neg c, d, e, \neg f, t\}$

2d, p. 27: The algorithm will consist of two procedures: a procedure `axiom` that potentially enumerates all possible axioms. For example, calling `axiom` for the first time gives the first possible axiom $p \supset (p \supset p)$. Calling it for the second time would give $q \supset (q \supset q)$; a third call would produce $p \supset (q \supset p)$, and so forth. (Writing such a procedure is far from easy, since it has to remember which formulas already were generated and which not. Programming languages with lazy evaluation, such as Haskell, are good at this.)

The second (much less complicated) procedure, then, would consider the newly generated axiom ϕ an scan T for formulas of the form $\phi \supset \psi$. Additionally, the it would scan T for formulas χ, if ϕ is of the form $\chi \supset \psi$. In both cases, MP can be applied, so that a new theorem can be added to T.

3a, p. 27: Backward-chaining with Modus Ponens amounts to moving from ψ to ϕ and $\phi \supset \psi$. The problem is that ϕ may be *any* formula.

3b, p. 27: Backward-chaining does work with Prolog because the inference system is not as strong as the inference system of propositional logic. More specifically, Prolog is incomplete with respect to the semantics of propositional logic.

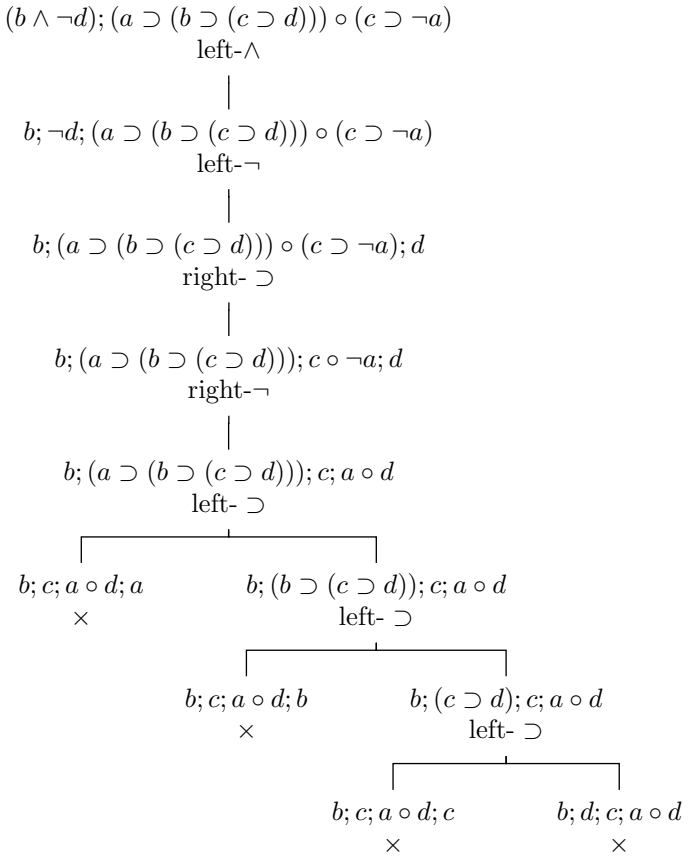
1, p. 28: If a branch is open and completed, we have not been able to close that branch and we never will, because no further reductions are possible. This branch contains a counterexample.

If a branch is open and incomplete, we have not been able to close that branch yet, but may possibly do so further down the branch, because no further reductions are possible.

If a branch is closed, we have shown that this particular branch cannot be analyzed into a countermodel. (Completion does not matter here, since closure alone is enough information to stop the analysis.)

2, p. 29: When building a refutation tree, four different types of formulas may be encountered: conjunctions, disjunctions, implications, and negations. Moreover, each of these formulas may occur at the LHS or RHS of the sequent. Thus, there are eight different cases to consider.

3, p. 29: (a) Invalid sequent. Countermodel: w, with $w(p) = w(q) = w(r) = 0$. (b) Invalid sequent. Countermodel: w, with $w(t) = w(k) = w(o) = 0$. (c) Valid sequent. Derivation:



Answers of (d-e) are not given.

4, p. 29: The statement refers to *the* refutation tree, as if refutation trees would be unique. But refutation trees need not be unique. Indeed, the expression $p \wedge q \wedge r \circ p$ may be reduced in two different ways: we may choose to remove the first conjunction first, and then remove the second conjunction, or we may do this reduction the other way around. Better: "every refutation tree of $p \wedge q \wedge r \vdash p$ closes".

5, p. 29: The method requires that all branches are completed, which often involves (much) more work compared to a refutation.

6, p. 29: If ϕ is a formula, let $l(''\phi'')$ be the length of ϕ (i.e., the number of characters in ϕ), and let $s(''\phi'')$ be the number of subformulas of ϕ. To show that the number of sub-formulas of a formula linearly depends on its length, we first prove the result for formulas that are stuffed up to the maximum with parentheses. E.g., if $\phi = \neg r \wedge s \wedge p$ we write $\phi = (((\neg(r)) \wedge (s)) \wedge (p))$ and prove the result for this type of formulas.

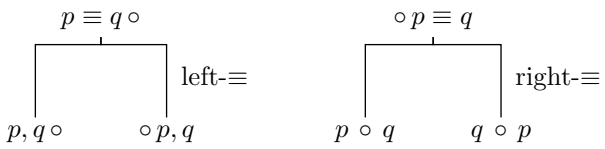
For such formulas we can now prove that the number of sub-formulas of ϕ is $l(''\phi'') = 3s(''\phi'')$. For example, if $\phi = \phi_1 \wedge \phi_2$,

$$\begin{aligned} l(''\phi'') &= l(''(\phi_1 \wedge \phi_2)'') \\ &= 1 + l(''\phi_1'') + 1 + l(''\phi_2'') + 1 \\ &= 1 + 3s(''\phi_1'') + 1 + 3s(''\phi_2'') + 1 \\ &= 3(s(''\phi_1'') + s(''\phi_2'') + 1) \\ &= 3(s(''\phi'')) \end{aligned}$$

The last step is justified by the fact that the number of subformulas of $\phi_1 \wedge \phi_2$ is equal to the number of subformulas of ϕ_1 and the number of subformulas of ϕ_2 and the formula ϕ $(= \phi_1 \wedge \phi_2)$ itself.

Now that we have the result for formulas with parentheses, we observe that the maximum number of parentheses that may be inserted (which directly corresponds to the number of sub-formulas in ϕ) *also* linearly depends on the length of the formula, hence the conclusion.

7a, p. 29: If $p \equiv q$ is true, then both propositions are true, or both propositions are false. If $p \equiv q$ is false, then exactly one of them is false while the other is true:



1, p. 31: Hint: if the tree of a failing refutation is put upside-down (leaves up), we almost have a proof in the Gentzen cut-free sequent calculus.

2, p. 32: If $m = 0$, then the LHS of the sequent is empty. Such a sequent is valid if and only if at least one formula on the RHS is a tautology (i.e., true in all valuations). If $n = 0$, then the RHS of the sequent is empty. Such a sequent is always invalid. In particular the sequent with LHS and RHS both empty is always invalid.

1, p. 35: Formula (2.3) on page 35 represents an exhaustive enumeration of complete scenario descriptions ("either both of 'em spoke the truth, or Pete lied, or Quincy lied, or they both lied"). Since in two-valued logic always one of these scenario's must hold, (2.3) on page 35 is valid.

2, p. 36: See Fig. B.1 on the next page.

2a, p. 36: There are nine leaves, hence nine branches. All branches end in a contradiction, so the formula is valid.

3a, p. 36: Thus, f_n is a disjunction of 2^n conjunctions, where each term (disjunct) is a conjunction of n literals. Hence, the length of f_n is $cn2^n$, where c is a fixed constant that depends how we write f_n (e.g. how many parenthesis we normally use).

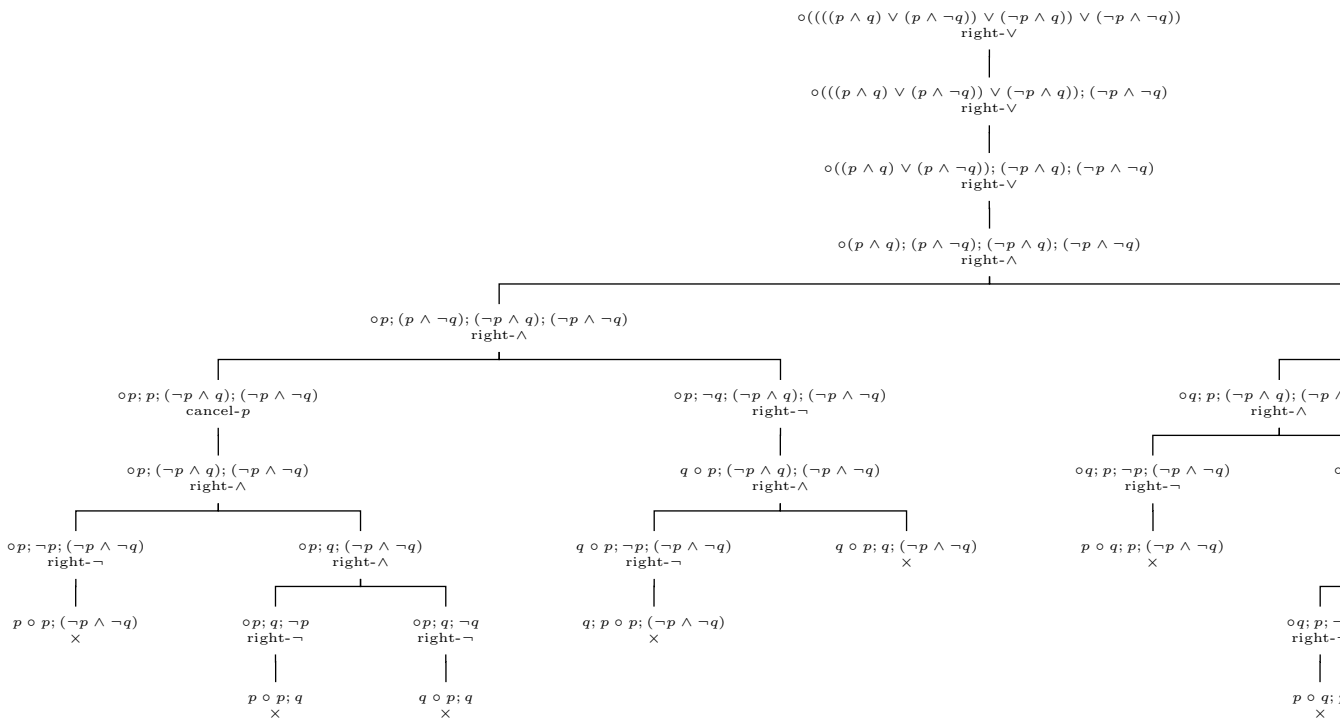


Figure B.1: Solution to Exercise 2 on page 36.

3b, p. 36: For every $n \geq 1$, the formula f_n is valid.

3c, p. 36: The formula f_n does have 2^n disjuncts. The truth-table of f_n has the same amount of rows, viz. 2^n.

3d, p. 36: The length of f_n is $(n + c)2^n$, because every disjunct has n proposition letters. The c is a linear estimate of the remaining connectives.

$$\frac{\text{length of } f_n}{\text{the number of rows in a truth-table of } f_n} = \frac{(n+c)2^n}{2^n} = n + c.$$

So the truth-table method is linear compared to the length of f_n.

3e, p. 36: For f_2 nine (9) branches; for f_3 three-hundred and forty-one (341) branches.

3f, p. 36: To describe what happens, we need some notation. Let

$b(\phi) =_{Def}$ the number of branches of a minimal refutation tree of ϕ"

It will be clear that, with this notation, $B(n) = b(f_n)$. Further, $B(1) = b(f_1) = 1$.

Let C_k represent a (any) conjunction consisting of k literals. From the definition of f_n it follows that f_n consists of 2^n formulas of type C_n. We may write this as

$$f_n = \underbrace{C_n \vee \ldots \vee C_n}_{2^n}$$

or, shorter,

$$f_n = 2^n C_n.$$

Hence,

$$\begin{aligned}
b(f_n) &= b(2^n C_n) && \text{[defintion of } f_n\text{]}\\
&= b(C_n \vee (2^n - 1)C_n) && \text{[spawning } C_n \text{ from } 2^n C_n\text{]}\\
&= 1 \,+\, n \cdot b(\pm p_i \vee (2^n - 1)C_n) && \text{[root, and material in } n \text{ branches]}\\
&\geq 1 \,+\, n \cdot b((2^n - 1)C_n) && \text{[}\pm p_i \text{ at most one logical connective]}\\
&\geq 1 \,+\, n \cdot b(2^{n-1} C_n) && \text{[}2^n - 1 \geq 2^{n-1}\text{, for } n \geq 1\text{]}\\
&\geq 1 \,+\, n \cdot b(2^{n-1} C_{n-1}) && \text{[}C_{n-1} \text{ less nodes than } C_n\text{]}\\
&= 1 \,+\, n \cdot b(f_{n-1}) && \text{[definition of } f_n\text{]}
\end{aligned}$$

where $\pm p_i$ is either p_i or $\neg p_i$. Hence, $B(n) \geq nB(n-1) + 1$. Since $B(1) = 1$, it follows with a simple induction argument that

$$\begin{aligned}
B(n) &\geq n + n(n-1) + n(n-1)(n-2) + n(n-1)(n-2)\cdots 1\\
&= n!\left(1 + \frac{1}{1!} + \frac{1}{2!} + \cdots + \frac{1}{n!}\right).
\end{aligned}$$

Hence, $B(n) > n!$

3g, p. 36: Since $B(n) > n!$, $B(n)$ grows faster than any polynomial function of 2^n, and since there are 2^n rows in the truth-table of f_n, even the most compact analytic refutation still performs much worse on formulas of the type f_n than ordinary truth-tables do:

$$\frac{\text{length of } f_n}{\text{number of branches in a minimal analytic refutation tree of } f_n} = \frac{cn2^n}{B(n)} < \frac{cn2^n}{n!}.$$

The last term goes to zero. Thus, as far as computational complexity is concerned, analytic refutation does not provide a uniform improvement on the truth-table method .

1, p. 39: We will prove something stronger, namely, that there simply does not *exist* such a problem. (Where a "problem" is a formula for which a theorem prover has to decide whether it is a tautology or not.) For let ϕ_0 such a problem. Here is a how to write a theorem prover that decides, within linear time, whether ϕ_0 is a tautology or not.

1. Calculate off-line, by hand or otherwise, whether ϕ_0 is a tautology or not.
2. Write a program that accepts propositional formulas and checks whether its input is syntactically equal to ϕ_0. If the input equals ϕ_0, the program returns the answer that was calculated beforehand. Else, it returns an arbitrary answer.

This "theorem proving procedure" is obviously correct for ϕ_0. Further, it is of linear time, because the syntactic check amounts to a simple pattern match, and simple pattern matching can be executed in linear time. The program is obviously of no use to other formulas.

2a, p. 39: *Linear KE is not sound.* We will have to prove that there exists an invalid sequent for which the corresponding tableau closes. [A tableau closes if every possible branch closes. A branch closes if we have applied every rule but fail to produce a counterexample nevertheless.]

Consider the sequent $\vdash p \wedge q$. The corresponding linear KE tableau starts with $\circ\, p \wedge q$. The formula on the right-hand side cannot be further reduced with rules from linear KE. At the same time, the node itself is not a counterexample. (It is generally impossible to see from a compound formula if it eventually leads to a counterexample. It might or it might not.) Hence, our attempt to find a counterexample in this branch fails, and the tableau closes.

2b, p. 39: *Linear KE is complete.* We will have to show that every open branch in a linear KE tableau corresponds to a counter-model to the original sequent. First, observe that linear KE is analytical: connectives can only be eliminated and cannot be introduced. The fact that linear KE is analytical implies that each branch is linear and finite. Let us call this Observation (i). Secondly, observe that linear KE rules respects satisfiability in both directions: if model m satisfies node N and node N' is derived from node N within linear KE, then there is a model m' that satisfies node N'. Conversely, if N' is derived from N within linear KE, and m' satisfies node N', then there is a model m that satisfies N. Let us call this Observation (ii).

Now suppose we have an open branch B in linear KE. From (i) we know that B is finite. Since B is open, B's end-point (the "leaf") must then consist of atoms only. These atoms spell out a counter-model for the end-node that thanks to (ii) propagates back to the root of B. Hence, the original sequent is falsified by the same counter-model.

Resumé

- A branch is *closed* as soon as it is clear that it cannot be further used to construct a counter-model. A branch closes in the following circumstances.

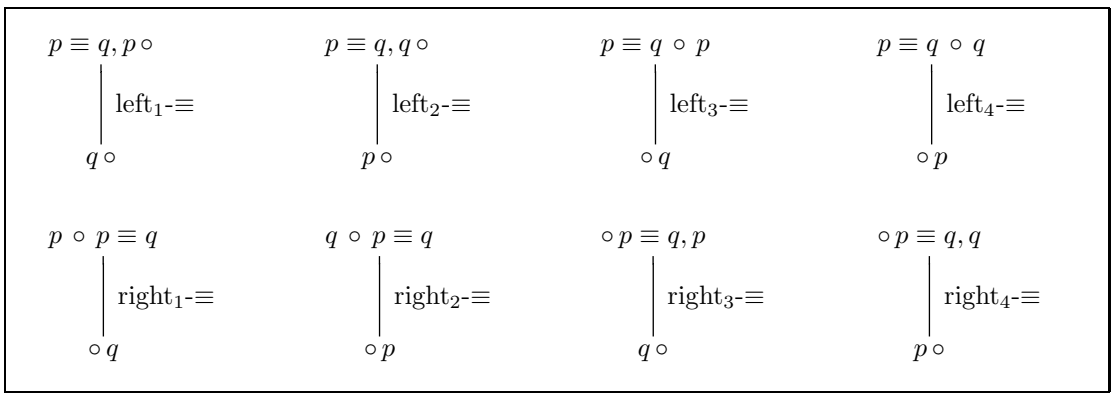
  1. A reduction yields two complementary formulas.
  2. No reduction is possible while there are still compound formulas.

  With (1) we are asked to produce a counterexample that valuates one and the same formula both true and false, which is impossible. With (2) we cannot tell whether the branch in question leads to a counterexample for it is generally impossible to decide on the satisfiability of compound formulas.

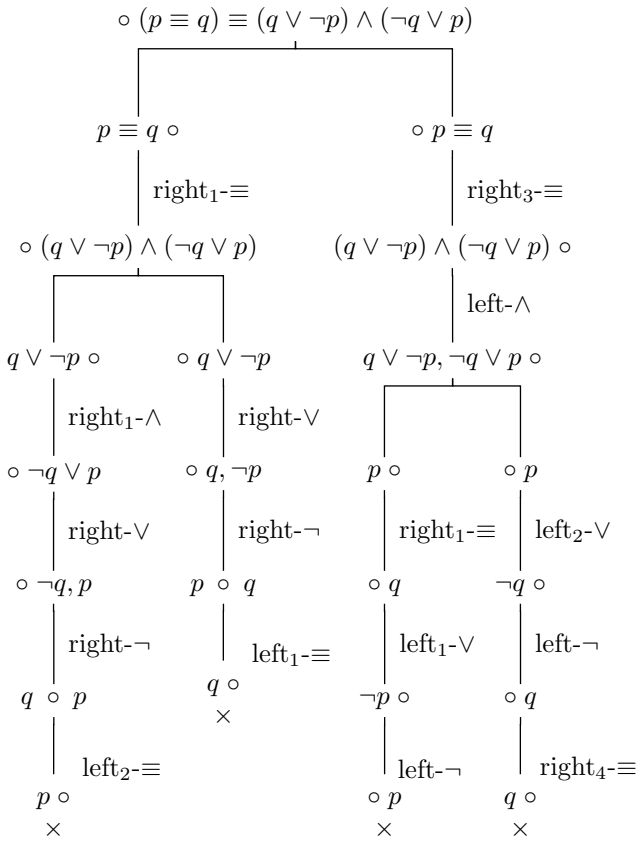
- A branch is *open* otherwise.

  An open branch on which no reductions are possible represents a counterexample. This is easy to prove for standard propositional analytic tableaus and linear KE. A completeness proof for KE and for first-order logics is more difficult, because these proofs systems allow for open branches that are infinite. With infinite open branches the counter-model is not represented by a leaf node, but is represented by the branch itself.

3a, p. 39: SAKE is a specialization of KE. Table 2.4 on page 38 contains rules in KE for negation, conjunction, disjunction and implication. Similarly, the following rules for equivalence can be formulated:



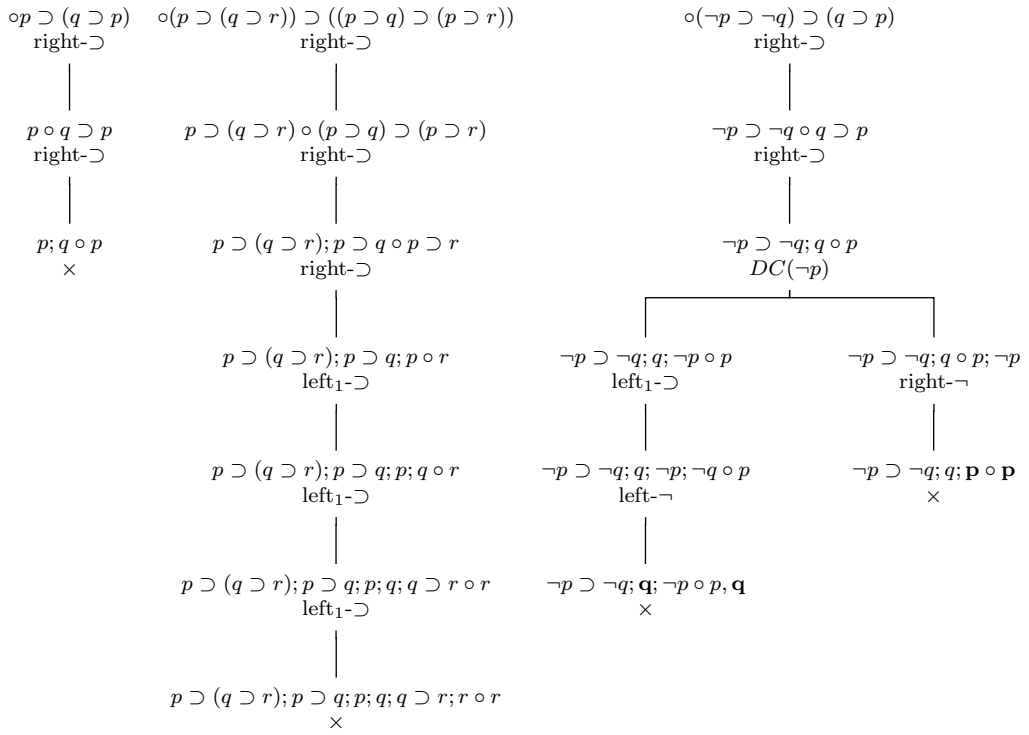
3b, p. 40: With the use of equivalence rules we can try to refute the equivalence in question:



Splits are always applications of DC. The first split, for instance, represents DC-$(p \equiv q)$. A counterexample does not exists since all branches close. Hence, the equivalence is logically valid.

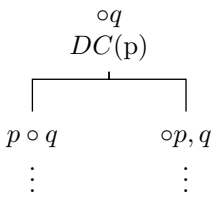
3c, p. 40: There is a refutation tree in which DC is applied maximally twice in each branch. An upper bound is therefore two. If you find a tree in which DC is applied maximally three times in a branch, then your upper bound is three.

4a, p. 40: Note that $(\neg p \supset \neg q) \supset (q \supset p)$ is proven by distinguishing cases with $\neg p$. (This, the left branch explores the case in which $\neg p$, and the right branch explores the case in which $\neg(\neg p)$:



4b, p. 40: We while have to show that if $\vdash_{KE} p$ and $\vdash_{KE} p \supset q$, then $\vdash_{KE} q$.

Suppose that $\vdash_{KE} p$ and $\vdash_{KE} p \supset q$. We will try to construct a KE-tree for q, and see where we get stuck and what we need at that point. Since p should somehow play a role in the refutation, we distinguish cases with p:

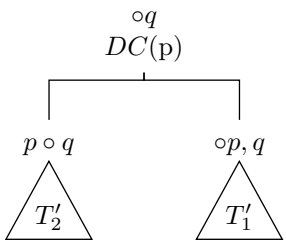


We now would like to have closed trees for $p \circ q$ and $\circ p, q$.

At this point, we use our knowledge that there are closed KE-refutation trees starting with $\circ p$ and $\circ p \supset q$. Denote these trees by T_1 and T_2, respectively.

- T_1 starts with $\circ p$. It doesn't "hurt" if we put q on the RHS of $\circ p$ and take it all along down the tree. In that case, we obtain a tree T_1' starting with $\circ p, q$.
- T_2 starts with the reduction $\circ p \supset q \xrightarrow{\text{right-}\supset} p \circ q$. Thus, there exists a closed KE-tree T_2' for $p \circ q$.

We now have a closed KE-tree for $\circ q$:

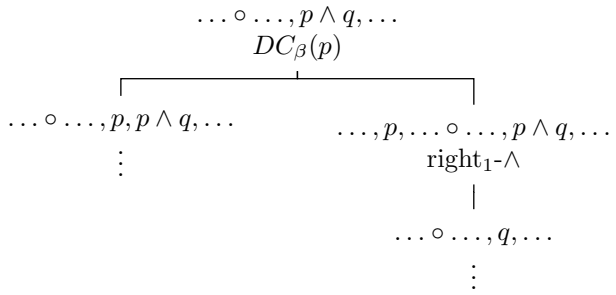


4c, p. 40: KE can simulate Hilbert's system. Hilbert's system is complete. Hence, KE is complete.

5, p. 41: Sketch: first apply DC to all proposition letters. This yields a refutation tree with 2^k branches of size $2^{k+1} - 1$ (draw a picture). Now each branch can be closed with a linear KE-tree. Let c be the length of the longest such linear KE-tree. According to Exercise 6 on page 29, c linearly depends on the maximum length of all formulas that occur in the sequent. Hence, the total amount of nodes that is needed for linear KE, is less than, or equal to, $c2^k$. Thus, the size of the total tree is $\leq (2^{k+1} - 1) + c2^k < (c + 2)2^k$. The last number is polynomially related the size of a truth-table for this tautology, which is of the order of magnitude of 2^k as well. Thus, KE-refutation can polynomially simulate truth-tables.

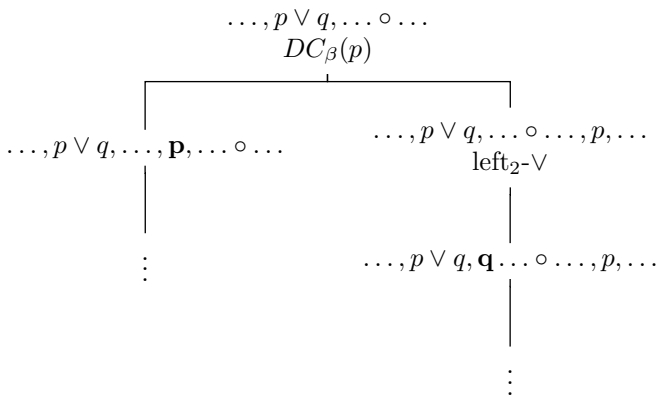
1, p. 42: If a β-rule is applied, the two resulting branches are linearly saturated, which means that either one of them will contain the desired sequent.

For example, if linear rules do not apply on $\ldots \circ \ldots, p \wedge q, \ldots$, we are permitted to use $DC_\beta(p)$ to distinguish cases. The branch is split and, in this case, q occurs on the RHS of a sequent on one of the two branches.



2, p. 43: We prove the claim for sequents with a disjunction on the LHS. Suppose the sequent $\ldots, p \vee q, \ldots \circ \ldots$ occurs on a saturated branch. We must show that either $\ldots, p \ldots \circ \ldots$ or $\ldots, q \ldots \circ \ldots$ occurs on that branch as well. As follows:

1. If p occurs on the RHS of $\ldots, p \vee q, \ldots \circ \ldots$, we have $\ldots, p \vee q, \ldots \circ \ldots, p, \ldots$ so that by $\text{left}_1\text{-}\vee$ we may produce $\ldots, p \vee q, q \ldots \circ \ldots, p, \ldots$.
2. If q occurs on the RHS of $\ldots, p \vee q, \ldots \circ \ldots$, we may apply $\text{left}_2\text{-}\vee$ to produce $\ldots, p \vee q, p \ldots \circ \ldots, p, \ldots$.
3. If neither p nor q occur on either side of the sequent, we may distinguish cases with p



In all cases, all branches have a sequent with either p or q on the LHS. In particular, the saturated branch contains such a sequent, which was to be proven.

3, p. 43: We must prove that every Hintikka set S of falsifiable sequents is falsified by a single countermodel. This can easily be seen by realizing that a countermodel m is "spelled out" by atomic sequents, i.e., sequents that have atoms on both sides of the "'$\circ$" and are no further reducible:

$$m(p) = \begin{cases} 1 & \text{if } p \text{ occurs on the LHS of a sequent in } S \\ 0 & \text{if } p \text{ occurs on the RHS of a sequent in } S \\ 1 & \text{otherwise} \end{cases}$$

Because no Hintikka-set contains a pair of sequences such that $\ldots, p, \ldots \circ \ldots$ and $\ldots \circ \ldots, p, \ldots$ for some p, m is well-defined. By definition, all atomic sequences in S are falsified by m. With induction on the number of connectives in a sequent and using the fact that S is downwards saturated, one can easily prove that all other elements of S are falsified by m as well.

4, p. 43: By (2), a saturated branch is a Hintikka set. By (3), all sequents in a Hintikka set are falsifiable by one and the same countermodel. The root is one particular element of this Hintikka set, and therefore also falsifiable.

1, p. 50: The claim (that a theorem prover solves *all* seventy-five problems of the Pelletier problem list within 1 milliseconds) of itself does not say very much, because the theorem prover can be tailored specifically to the Pelletier problem list. This can be done, for instance, by making a list of all problems on the Pelletier list first, supplemented with proofs. Before the input is handed over to the "real" theorem prover, it is matched against all seventy-give members of the list. If a pattern-match succeeds, the input apparently corresponded to one of the Pelletier problems, and the theorem prover reports that it has "proven" the theorem and returns the proof that it has stored in memory.

2b, p. 50:
```
(s0_0 <-> (p0 & q0)) & ~c0_0 & (s1_0 <-> (p1 & q0)) & ~c1_0 & ~c1_0
& (s1_1 <-> (c0_0 <-> (s1_0 <-> ((p0 & q1))))) & (c1_1 <-> ((c0_0 &
s1_0) | (c0_0 & p0 & q1) | (s1_0 & p0 & q1))) & ~(p1 & q1) & ~c1_1 &
~s0_0 & s1_1 & ~q1 & ~p1.
```

1, p. 54: Formula (3.1) on page 54 says that if a relation x is anti-symmetric, then x is also irreflexive. Take $x =$ "$<$", for instance. Then (3.1) says that, if $y < z$ implies $z \not< y$, then $y \not< y$, for all y. It is hard to fabricate an interpretation in which (3.1) is false. We can't be sure, however, because we do not have exact knowledge of the semantics of second-order languages.

2, p. 54: Let's start with the definition of a WFF. A *well-formed formul* is a logical expression of which the syntax is defined inductively, along the lines of: "If ϕ_1 and ϕ_2 are WFFs, then $\phi_1 \wedge \phi_2$ is a WFF,"' and "if ϕ is a WFF and $i \geq 1$ then $(\forall x_i)(\phi)$ is a WFF. With WFFs you typically can't compromise on the use of parentheses and other auxiliary symbols (such as periods and commas).

A *formula*, then, is a logical expression that can be parsed unambiguously by you and me and by an ATP as a well-formed formula. Typically, a formula is a WFF with some parentheses left out.

Finally, a *sentence* is a formula in which every variable is bound by a quantifier.

3, p. 54: (a) Formula, but not a sentence, because not every variable is bound by a quantifier; particularly y is not bound by a quantifier; (b) Formula, and a sentence; (c) Formula, although the notation is rejected by, for instance, parsers that expect nested quantifiers; the formula is a sentence; (d-e) same as (c); (f) Formula, but not a sentence; it is not obligatory that $(\forall z)(Pf(x), a)$ is written as, for instance $(\forall z)(P(f(x), a))$. On the other hand, the expression $(\forall z)(Pfx, a)$ (note the absence of parentheses around the function arguments) could not be exchanged as a formula, because this would give problems with parsing large terms.

4, p. 55: (a) Px, y free: $\{x, y\}$; bound: $\varnothing$; (b) $(\forall x)(Px, y)$ free: $\{y\}$; bound: $\{x\}$; (c) $(\forall y)(Px, y)$ free: $\{x\}$; bound: $\{y\}$; (d) $(\forall x)(Py) \wedge (\forall y)(Px, y)$ free: $\{x, y\}$; bound: $\{y\}$. Note that y occurs both as a free variable and as a bound variable. In general, variables can be free and bound in the same formula (but obviously in different places of that formula).

5, p. 55: (a) $(\forall x)[Px, y \supset (\forall w)(Pw, x \vee (\exists y)(Py, z))]$
(b) $(\forall x)[Px, \mathbf{f}(\mathbf{b}, \mathbf{g}(\mathbf{a}, \mathbf{w})) \supset (\forall w)(Pw, x \vee (\exists y)(Py, z))]$
(c) $(\forall x)[Px, y \supset (\forall w)(Pw, x \vee (\exists y)(Py, \mathbf{f}(\mathbf{b}, \mathbf{g}(\mathbf{a}, \mathbf{w}))))]$
(d) $(\forall x)[Px, y \supset (\forall w)(Pw, x \vee (\exists y)(Py, z))]$

6, p. 55: (a) Every term is free for x in ϕ, because x does not occur free in ϕ. Accordingly, $\phi[t_1/x]$ would be a fair substitution since no free occurrence of x is replaced by t_1. (b) Take $t_2 = x$. Then $(\exists x)(Rx, y)[t_2/y] = (\exists x)(Rx, y)[x/y] = (\exists x)(Rx, x)$. (c) Take $R =<$ in the domain of the integers. Then $(\exists x)(Rx, y)$ can be made true while $(\exists x)(Rx, x)$ is always false.

7, p. 55: The formula

$$\phi = (\forall x)(Px, y \supset (\forall w)(Pw, x \vee (\exists y)(Py, z))) \tag{B.1}$$

quantifies over x, w, and y. (a) to verify whether $f(b, g(a, w))$ is free for x in ϕ, we first track which occurrences of x are free in ϕ. Since the formula starts with an x-quantifier, there are no free xs, so that the requirement that all free occurrences of x in ϕ are not within the scope of a variable that occurs in $f(b, g(a, w))$, is vacuously fulfilled. Thus, $f(b, g(a, w))$ is free for x in ϕ.
(b) To verify whether $f(b, g(a, w))$ is free for y in ϕ, we observe that the only free occurrence of y in ϕ is the first one. This occurrence does not occur within the scope w, however, so that the answer is affirmative.
(c) The variable z occurs free in ϕ, but is bound by a universal w-quantifier. Thus, the answer is negative.
(d) All conditions are vacuously fulfilled, so the answer is affirmative.

8a, p. 55: Let M be the model with a domain D that is equal to the set of integers, and interpret the predicate P as "less than". Every variable assignment $\nu :$ VARS $\rightarrow D$ will satisfy the three formulas given, so that, for every ν, the pair (M, ν) is a solution.

8b, p. 55: Let D be the set of fractions, and interpret the predicate P as "less than". A property of fractions is that there always lies fraction between two other fractions. Every variable assignment will satisfy the four formulas given.

8c, p. 56: Let D be the set of negative integers, and interpret the predicate P as "less than". Any variable assignment $\nu :$ VARS $\rightarrow D$ such that $\nu(y) = -1$ will satisfy the four formulas given.

8d, p. 56: Let D be the set of integers, interpret the predicate E as "equal to," interpret the function f as "times," and interpret the constant a as 1. Any variable assignment $\nu :$ VARS $\rightarrow D$ such that $\nu(z) = 0$ will satisfy the four formulas given.

9, p. 56: We prove $s(t'[t/x]) = s[t_s/x](t')$ with induction on the composition of t'. If t' is a constant a, say, then there is nothing to substitute and both sides of the equality reduce to a_M. So in this case, the equality is settled. If $t' = x$, then

$$s(t'[t/x]) = s(x[t/x]) = s(t).$$

Similarly,

$$s[t_s/x](t') = s[t_s/x](x) = t_s = s(t).$$

If t' is a variable $y \neq x$, then

$$s(t'[t/x]) = s(y[t/x]) = s(y).$$

Similarly,

$$s[t_s/x](t') = s[t_s/x](y) = s(y),$$

since $s[t_s/x]|_{\mathrm{VARS}\setminus\{x\}} \equiv s|_{\mathrm{VARS}\setminus\{x\}}$. Finally, if $t' = f(t_1, \ldots, t_n)$, then

$$\begin{aligned}
s(t'[t/x]) &= s(f(t_1, \ldots, t_n)[t/x]) && \text{[definition of } t'\text{]} \\
&= f_M(s(t_1[t/x]), \ldots, s(t_n[t/x])) && \text{[inductive definition of } s\text{]} \\
&= f_M(s[t_s/x](t_1), \ldots, s[t_s/x](t_n)) && \text{[induction hypothesis]} \\
&= s[t_s/x](f(t_1, \ldots, t_n)) && \text{[}s\text{, opposite direction]} \\
&= s[t_s/x](t') && \text{[definition of } t'\text{]}
\end{aligned}$$

10a, p. 56: A simple counterexample is the formula $(\forall y)(R(x, y))$. With this formula, x lies in the scope of y. If x is substituted by a term t containing a y, then the occurrence of y in t will be bound by the quantifier $(\forall y)$. For example, if $t = y$, then $(\forall y)(R(x, y))[t/x]$ becomes $(\forall y)(R(f(y), y))$. It is clear that these two formulas are not satisfied by the same models.

10b, p. 56: The objective is to show that $(M, s) \models \phi[t/x]$ iff $(M, s[t_s/x]) \models \phi$. As as usual, we prove this with induction on the complexity of the formula in question. We prove three typical cases: ϕ is an atomic predicate, ϕ is a conjunction, ϕ is a quantified over x, and ϕ is a quantified over a variable different from x.

$$\begin{aligned}
(M, s) \models \phi[t/x] \text{ iff } & (M, s) \models P(t_1, \ldots, t_n)[t/x] && \text{[definition of } \phi\text{]} \\
\text{iff } & (M, s) \models P(t_1[t/x], \ldots, t_n[t/x]) && \text{[by definition of substitution]} \\
\text{iff } & ((t_1[t/x])_s, \ldots, (t_n[t/x])_s) \in P_M && \text{[definition of satisfiability]} \\
\text{iff } & ((t_1)_{s[t_s/x]}, \ldots, (t_n)_{s[t_s/x]}) \in P_M && \text{[substitution lemma for terms]} \\
\text{iff } & (M, s[t_s/x]) \models P(t_1, \ldots, t_n) && \text{[definition of satisfiability]} \\
\text{iff } & (M, s[t_s/x]) \models \phi && \text{[definition of } \phi\text{]}
\end{aligned}$$

If ϕ is a conjunction, we have

$$\begin{aligned}
(M, s) \models \phi[t/x] \text{ iff } & (M, s) \models (\phi_1 \wedge \phi_2)[t/x] && \text{[definition of } \phi\text{]} \\
\text{iff } & (M, s) \models \phi_1[t/x] \wedge \phi_2[t/x] && \text{[by definition of substitution]} \\
\text{iff } & (M, s) \models \phi_1[t/x] \text{ and } (M, s) \models \phi_2[t/x] && \text{[def. of sat.]} \\
\text{iff } & (M, s[t_s/x]) \models \phi_1 \text{ and } (M, [t_s/x]) \models \phi_2 && \text{[induction hypothesis]} \\
\text{iff } & (M, s[t_s/x]) \models \phi_1 \wedge \phi_2 \\
\text{iff } & (M, s[t_s/x]) \models \phi.
\end{aligned}$$

If ϕ is a quantified over x, then

$$\begin{aligned}
(M, s) \models \phi[t/x] \text{ iff } & (M, s) \models (\forall x)\phi_1[t/x] \\
\text{iff } & (M, s) \models (\forall x)\phi_1 && \text{[}x\text{ not free in } (\forall x)\phi_1\text{]} \\
\text{iff for all } d \in D\text{: } & (M, s[d/x]) \models \phi_1 \\
\text{iff for all } d \in D\text{: } & (M, s[t_s/x][d/x]) \models \phi_1 && \text{[no problem, since } t_s \\
\text{iff } & (M, s[t_s/x]) \models (\forall x)\phi_1 && \text{ is overwritten by } d\text{]}
\end{aligned}$$

Finally, if ϕ is a quantified over a variable different from x, we have

$$\begin{aligned}
(M,s) \models \phi[t/x] \text{ iff } & (M,s) \models ((\forall y)\phi_1)[t/x] \\
\text{iff } & (M,s) \models (\forall y)(\phi_1[t/x]) && \text{[by definition of substitution]} \\
\text{iff for all } d \in D: \ & (M, s[d/y]) \models \phi_1[t/x] && \text{[definition of } \forall\text{-quantifier]} \\
\text{iff for all } d \in D: \ & (M, s[d/y][t_{s[d/y]}/x]) \models \phi_1 && \text{[induction step]} \\
\text{iff for all } d \in D: \ & (M, s[d/y][t_s/x]) \models \phi_1 && [t \text{ is free for } x \text{ in } (\forall y)\phi_1] \\
\text{iff for all } d \in D: \ & (M, s[t_s/x][d/y]) \models \phi_1 && \text{[different variables,} \\
\text{iff } & (M,s) \models (\forall y)\phi_1[t_s/x] && \text{so we may swap]}
\end{aligned}$$

1a, p. 58: $D(Px, a) = \{a\}$

1b, p. 58: $D((\forall x)Px \supset Qf(x), a) = \{f^n(a) | n \geq 0\} = \{a, f(a), f(f(a)), \ldots\}$

1c, p. 58: $D(Rg(f(x), y), z)$ is the set H such that $c_0 \in H$, $f(t) \in H$ if $t \in H$, and $g(t_1, t_2) \in H$ if $\{t_1, t_2\} \subseteq H$. I.e., $H = \{c_0,\ f(c_0),\ g(c_0, c_0),\ f(c_0),\ f^2(c_0),\ f(g(c_0, c_0)),\ \ldots\}$.

1, p. 62: If the rules are not changed, it is impossible to escape from the endless loop, because the fresh constants produced by "left-$\exists$" and "right-$\forall$" yields new material for "left$^+$-$\forall$" and "right$^+$-$\exists$", no matter in which order the rules are applied.

2, p. 62: It is possible to recognizes cases such as displayed in Example 3.8 on page 62 by dropping the constraint that every application of *left*-$\exists$ and *right*-$\forall$ should use fresh constants. Thus, we allow reuse of formerly introduced constants. This is no problem if we find a countermodel: in such cases we apparently have found a countermodel in which some constants play a double rôle—no problem. We *do* have a problem, however, if the use of a tainted variable does *not* yield a counterexample, for then we might have inadvertently introduced a dependency between two otherwise unrelated predicates. Thus, if the branch closes we will have to backtrack to the point at which we applied *left*-$\exists$ and *right*-$\forall$ to a constant that was tainted and try another one. If the new constant is tainted again, we again run the risk of being forced to backtrack in case we fail to find a counterexample.

Let us see how this works out for $(\forall x \exists y)(p\,x, y) \circ (\exists y \forall x)(p\,x, y)$:

$$\begin{array}{c}
(\forall x \exists y)(p\,x, y) \circ (\exists y \forall x)(p\,x, y) \\
\text{left}^+\text{-}\forall \text{ with } c_0 \\
\Big| \\
(\exists y)(p\,c_0, y) \circ \\
\text{right}^+\text{-}\exists \text{ with } c_0 \\
\Big| \\
\circ(\forall x)(p\,x, c_0) \\
\text{left-}\exists, \textbf{ reusing } \mathbf{c_0} \textbf{ [backtrack point A]} \\
\Big| \\
p\,c_0, c_0 \circ \\
\text{right-}\forall, \textbf{ reusing } \mathbf{c_0} \textbf{ [backtrack point B]} \\
\Big| \\
\circ p\,c_0, c_0 \\
\times
\end{array}$$

Thus, reusing c_0 at point A and B does not result in a counterexample. We backtrack to B and

try c_{i+1}. This is c_1, which happens to be fresh:

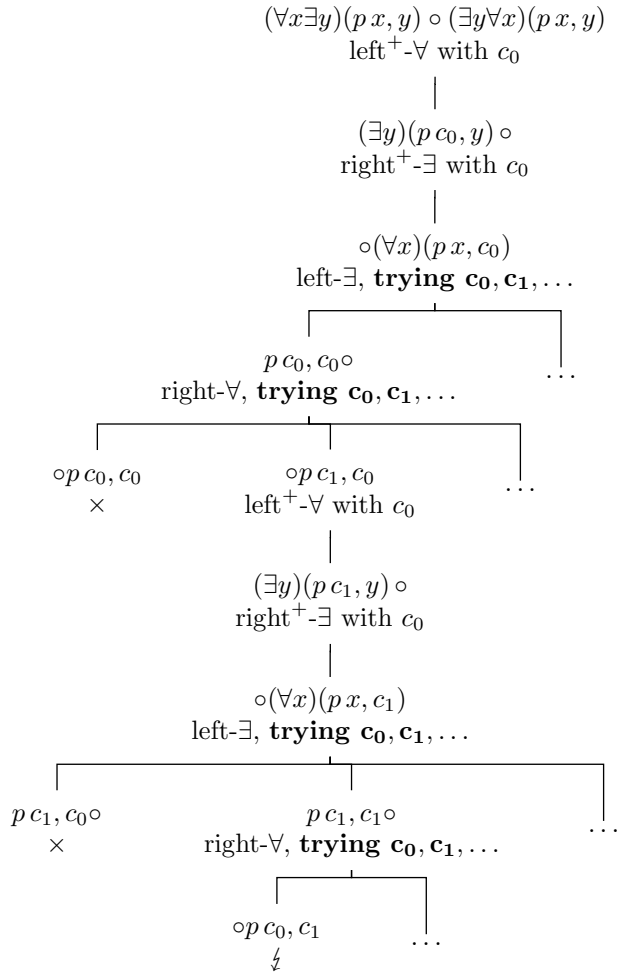
$$
\begin{array}{c}
\vdots \\
| \\
p\,c_0, c_0 \circ \\
\text{right-}\forall\text{, } \textbf{introducing } \mathbf{c_1} \text{ [}\textbf{backtrack point B}\text{]} \\
| \\
\circ p\,c_1, c_0 \\
\text{left}^+\text{-}\forall \text{ with } c_0 \\
| \\
(\exists y)(p\,c_1, y) \circ \\
\text{right}^+\text{-}\exists \text{ with } c_0 \\
| \\
\circ(\forall x)(p\,x, c_1) \\
\text{left-}\exists\text{, } \textbf{reusing } \mathbf{c_0} \text{ [}\textbf{backtrack point C}\text{]} \\
| \\
p\,c_1, c_0 \circ \\
\times
\end{array}
$$

Again we fail to find a counterexample, so we backtrack to C to try c_1:

$$
\begin{array}{c}
\vdots \\
| \\
\circ(\forall x)(p\,x, c_1) \\
\text{left-}\exists\text{, } \textbf{reusing } \mathbf{c_1} \text{ [}\textbf{backtrack point C}\text{]} \\
| \\
p\,c_1, c_1 \circ \\
\text{right-}\forall\text{, } \textbf{reusing } \mathbf{c_1} \text{ [}\textbf{backtrack point D}\text{]} \\
| \\
\circ p\,c_0, c_1 \\
\lightning
\end{array}
$$

No more rules can be applied here and the branch remains open, so that we have found a finite countermodel M this time with domain $D = \{0, 1\}$ and with $R = \{(0, 0), (1, 1)\}$.

It is possible to represent the entire search process in one tree:



The splits in this tree are of a different kind than the splits we have seen thus far. This representation is also possible here, because the three separate trees do not split themselves—otherwise, we would get a hypertree). Please do not confuse the two tree-types.

The "try-old-constants-first" method makes sense only if a finite counterexample exists. Else, the only swallows extra time.

1, p. 64: The proof is not much different than what happens with the propositional case (p. 64). Therefore, we only prove

> $\ldots, (\forall x)p, \ldots \circ \ldots$ on a saturated branch then, for all ground terms t, the sequent $\ldots, p\,[t/x], \ldots \circ \ldots$ occurs on that branch as well

This almost immediately follows from the definition of saturation, provided that the algorithm that generates the branch is *fair*. An algorithm is said to be fair if every formula of every sequent on every branch branch is eventually analyzed. An example of a fair algorithm is to expand all formulas of every new sequent simultaneously, mark the sequent as "used" [or "expanded," or "old"] and traverse the rest of the tree in a breadth-first manner.

2, p. 64: Hint: consult Exercise 2 on page 64. Atomic formulas are $P(t_1, \ldots, t_n)$, with $t_1, \ldots, t_n$ ground; atomic sequents are sequents with atomic formulas only.) The proof is not much different

3, p. 64: Same as with Exercise 4 on page 43.

1, p. 72: True. The formula $rv_4, \varsigma_3(v_3)$ can be unified with the formula $r\varsigma_1, v_1$ by means of the substitution $\tau = \{\varsigma_3(\varsigma_1)/v_1, \varsigma_1/v_3, \varsigma_1/v_4\}$.

2a, p. 72: Two possible solutions:

$$\begin{array}{c} (\forall x)(p\,x) \circ (\forall x)(p\,x) \\ \text{right-}\forall \text{ (no free variables)} \\ \vert \\ (\forall x)(p\,x) \circ p\varsigma_1 \\ \text{left-}\forall \text{ (introducing new variable } v_1) \\ \vert \\ \phi, pv_1 \circ p\varsigma_1 \\ \times \text{ with } \sigma = \{\varsigma_1/v_1\} \end{array} \qquad \text{and} \qquad \begin{array}{c} (\forall x)(p\,x) \circ (\forall x)(p\,x) \\ \text{left-}\forall \text{ (introducing new variable } v_1) \\ \vert \\ \phi, pv_1 \circ (\forall x)(p\,x) \\ \text{right-}\forall \text{ (no free variables in the RHS)} \\ \vert \\ \phi, pv_1 \circ p\varsigma_1 \\ \times \text{ with } \sigma = \{\varsigma_1/v_1\} \end{array}$$

in both refutations, $\phi = (\forall x)(p\,x)$.

2b, p. 72:

$$\begin{array}{c} \circ\, (\exists x)[p\,x \supset (\forall x)(p\,x)] \\ \text{right-}\exists \text{ (introducing new variable } v_1) \\ \vert \\ \circ\, \phi, pv_1 \supset (\forall x)(p\,x) \\ \text{right-}\supset \\ \vert \\ pv_1 \circ \phi, (\forall x)(p\,x) \\ \text{right-}\forall \text{ (no free variables in the RHS)} \\ \vert \\ pv_1 \circ \phi, p\varsigma_1 \\ \times \text{ with } \sigma = \{\varsigma_1/v_1\} \end{array}$$

where $\phi = (\exists x)[p\,x \supset (\forall x)(p\,x)]$.

2c, p. 72:

$$\begin{array}{c} (\forall x, y)(p\,x \wedge p\,y) \circ (\exists x, y)(p\,x \vee p\,y) \\ \text{left-}\forall \text{ (introducing new variable } v_1) \\ \vert \\ \phi_1, (\forall y)(pv_1 \wedge p\,y) \circ (\exists x, y)(p\,x \vee p\,y) \\ \text{left-}\forall \text{ (introducing new variable } v_2) \\ \vert \\ \phi_1, \phi_2, pv_1 \wedge pv_2 \circ (\exists x, y)(p\,x \vee p\,y) \\ \text{right-}\exists \text{ (introducing new variable } v_3) \\ \vert \\ \phi_1, \phi_2, pv_1 \wedge pv_2 \circ \psi_1, (\exists y)(pv_3 \vee p\,y) \\ \text{right-}\exists \text{ (introducing new variable } v_4) \\ \vert \\ \phi_1, \phi_2, pv_1 \wedge pv_2 \circ \psi_1, \psi_2, pv_3 \vee pv_4 \\ \text{left-}\wedge \text{ and right-}\vee \\ \vert \\ \phi_1, \phi_2, pv_1, pv_2 \circ \psi_1, \psi_2, pv_3, pv_4 \\ \times \text{ with } \sigma = \{v_1/v_3\} \end{array}$$

where

$$\phi_1 = (\forall x, y)(p\,x \wedge p\,y), \quad \psi_1 = (\exists x, y)(p\,x \vee p\,y),$$
$$\phi_2 = (\forall y)(pv_1 \wedge p\,y), \quad \psi_2 = (\exists y)(pv_3 \vee p\,y).$$

2d, p. 72: Two possible solutions:

$$(\forall x, y)(p\,x \wedge p\,y) \circ (\forall x, y)(p\,x \vee p\,y)$$
left-$\forall$ (introducing new variable v_1)
$$\phi_1, (\forall y)(pv_1 \wedge p\,y) \circ (\forall x, y)(p\,x \vee p\,y)$$
left-$\forall$ (introducing new variable v_2)
$$\phi_1, \phi_2, pv_1 \wedge pv_2 \circ (\forall x, y)(p\,x \vee p\,y)$$
right-$\forall$ (no free variables in the RHS)
$$\phi_1, \phi_2, pv_1 \wedge pv_2 \circ (\forall y)(p\varsigma_1 \vee p\,y)$$
right-$\forall$ (no free variables in the RHS)
$$\phi_1, \phi_2, pv_1 \wedge pv_2 \circ p\varsigma_1 \vee p\varsigma_2$$
left-$\wedge$, right-$\vee$
$$\phi_1, \phi_2, pv_1, pv_2 \circ p\varsigma_1, p\varsigma_2$$
$\times$ with $\sigma = \{\varsigma_1/v_1, \varsigma_2/v_2\}$

and

$$(\forall x, y)(p\,x \wedge p\,y) \circ (\forall x, y)(p\,x \vee p\,y)$$
right-$\forall$ (two times)
$$(\forall x, y)(p\,x \wedge p\,y) \circ p\varsigma_1 \vee p\varsigma_2$$
left-$\forall$ (two times)
$$\phi_1, \phi_2, pv_1 \wedge pv_2 \circ p\varsigma_1 \vee p\varsigma_2$$
left-$\wedge$, right-$\vee$
$$\phi_1, \phi_2, pv_1, pv_2 \circ p\varsigma_1, p\varsigma_2$$
$\times$ with $\sigma = \{\varsigma_1/v_1, \varsigma_2/v_2\}$

where $\phi_1 = (\forall x, y)(p\,x \wedge p\,y)$ and $\phi_2 = (\forall y)(pv_1 \wedge p\,y)$ for both refutations.

1a, p. 74: $f(x, a); f(b, y)\ -\{b/x\} \rightarrow f(b, a); f(b, y)\ -\{a/y\} \rightarrow f(b, a); f(b, a)$. MGU is $\{b/x, a/y\}$.

1b, p. 74: $g(x, a); g(b, x)\ -\{b/x\} \rightarrow g(b, a); g(b, b)$. Not unifiable at second coordinate.

1c, p. 74: $h(x, a, w, y); h(b, a, z, c)\ -\{b/x\} \rightarrow h(b, a, w, y); h(b, a, z, c)\ -\{z/w\} \rightarrow h(b, a, z, y); h(b, a, z, c)\ -\{c/y\} \rightarrow h(b, a, z, c); h(b, a, z, c)$. MGU is $\{b/x, z/w, c/y\}$.

1d, p. 74: $f(x, w, w, x); f(w, z, d, a)\ -\{w/x\} \rightarrow f(w, w, w, w); f(w, z, d, a)\ -\{z/w\} \rightarrow f(z, z, z, z); f(z, z, d, a)\ -\{d/z\} \rightarrow f(d, d, d, d); f(d, d, d, a)$. Not unifiable at fourth coordinate.

1e, p. 74: $g(x, y, w, z); g(w, f(z), x, y)\ -\{w/x\} \rightarrow g(w, y, w, z); g(w, f(z), w, y)\ -\{f(z)/y\} \rightarrow g(w, f(z), w, z); g(w, f(z), w, f(z))$. Occurs check: variable z occurs in $f(z)$. Not unifiable at fourth coordinate.

1f, p. 74: $h(x, y, y, z); h(x, y, z, x)\ -\{z/y\} \rightarrow h(x, z, z, z); h(x, z, z, x)\ -\{x/z\} \rightarrow h(x, x, x, x); h(x, x, x, x)$. MGU is $\{z/y, x/z\}$. (Cannot be simplified to $\{x/y\}$!)

1a, p. 75: $\circ\ (\forall x)((\forall y)(x = y\ \supset y = x))$
(replace quantified x by a new constant, ς_1)
$$\circ\ (\forall y)(\varsigma_1 = y\ \supset y = \varsigma_1)$$
(replace quantified y by a new constant, ς_2)

$\circ\ \varsigma_1 = \varsigma_2\ \supset \varsigma_2 = \varsigma_1$
(right-$\supset$)
|
$\varsigma_1 = \varsigma_2\ \circ \varsigma_2 = \varsigma_1$
(right replacement of ς_1 with $\varsigma_1 = \varsigma_2$)
|
$\varsigma_1 = \varsigma_2\ \circ \varsigma_2 = \varsigma_2$
(left replacement of ς_1 with $\varsigma_1 = \varsigma_2$)
|
$\varsigma_2 = \varsigma_2\ \circ \varsigma_2 = \varsigma_2$
$\times$

1b, p. 75: $\circ\ (\forall x)((\exists y)(x = y))$
(replace quantified x by a new constant, ς_1)
|
$\circ\ (\exists y)(\varsigma_1 = y)$
(replace quantified y by the same constant, ς_1.
[We work with conventional tableaus.])
|
$\circ\ \varsigma_1 = \varsigma_1$
(left-= with ς_1)
|
$\varsigma_1 = \varsigma_1 \circ \varsigma_1 = \varsigma_1$
$\times$

2a, p. 75: $\circ\ (\forall u)((\forall v)((\forall w)((\forall x)((u = v \wedge w = x) \supset (Puw \supset Pvx)))))$
(replace quantified variables by new constants, $\varsigma_1, \ldots, \varsigma_4$)
|
$\circ\ (\varsigma_1 = \varsigma_2 \wedge \varsigma_3 = \varsigma_4) \supset (P\varsigma_1\varsigma_3 \supset P\varsigma_2\varsigma_4)$
(right-$\supset$)
|
$\varsigma_1 = \varsigma_2 \wedge \varsigma_3 = \varsigma_4 \circ P\varsigma_1\varsigma_3 \supset P\varsigma_2\varsigma_4$
(left-$\wedge$)
|
$\varsigma_1 = \varsigma_2;\ \varsigma_3 = \varsigma_4, P\varsigma_1\varsigma_3 \circ P\varsigma_2\varsigma_4$
(left replacement of ς_1 with $\varsigma_1 = \varsigma_2$)
|
$\varsigma_1 = \varsigma_2;\ \varsigma_3 = \varsigma_4, P\varsigma_2\varsigma_3 \circ P\varsigma_2\varsigma_4$
(left replacement of ς_3 with $\varsigma_3 = \varsigma_4$)
|
$\varsigma_1 = \varsigma_2;\ \varsigma_3 = \varsigma_4, P\varsigma_2\varsigma_4 \circ P\varsigma_2\varsigma_4$
$\times$

3a, p. 75: $\circ\ (\forall x)(\exists y)(\exists z)(y = f(x) \wedge z = g(y))$
(replace quantified variable x by new constant, ς_1)
|
$\circ\ (\exists y)(\exists z)(y = f(\varsigma_1) \wedge z = g(y))$
(free y and z
[We work with the postponed substitution version of tableaux])
|
$\circ\ y = f(\varsigma_1) \wedge z = g(y)$
(right-$\wedge$)
| |
$\circ\ y = f(\varsigma_1)$ $\circ\ z = g(y)$

$$\begin{array}{cc}
\text{(left-= with } f(\varsigma_1)) & \text{(left-= with } g(y) \\
| & | \\
f(\varsigma_1) = f(\varsigma_1) \circ y = f(\varsigma_1) & g(y) = g(y) \circ z = g(y) \\
\text{(close with } [f(\varsigma_1)/y]) & \text{(close with } [g[y]/z]) \\
\times & \times
\end{array}$$

3b, p. 75: $\circ\ (\forall x)(\forall y)(x = y \supset f(x) = f(y))$
(replace quantified variables by new constants)
$$\begin{array}{c}
| \\
\circ\ \varsigma_1 = \varsigma_2 \supset f(\varsigma_1) = f(\varsigma_2) \\
\text{(right-}\supset) \\
| \\
\varsigma_1 = \varsigma_2 \circ f(\varsigma_1) = f(\varsigma_2) \\
\text{(right replacement with } \varsigma_1 = \varsigma_2) \\
| \\
\varsigma_1 = \varsigma_2 \circ f(\varsigma_2) = f(\varsigma_2) \\
\text{(left-= with } f(\varsigma_2)) \\
| \\
f(\varsigma_2) = f(\varsigma_2), \varsigma_1 = \varsigma_2 \circ f(\varsigma_2) = f(\varsigma_2) \\
\times
\end{array}$$

3c, p. 75: $\circ\ (\forall x)(\forall y)(\forall z)(y = f(x) \wedge z = f(x) \supset y = z)$
(replace quantified variables by new constants)
$$\begin{array}{c}
| \\
\circ\ \varsigma_2 = f(\varsigma_1) \wedge \varsigma_3 = f(\varsigma_1) \supset \varsigma_2 = \varsigma_3 \\
\text{(right-}\supset) \\
| \\
\varsigma_2 = f(\varsigma_1) \wedge \varsigma_3 = f(\varsigma_1) \circ \varsigma_2 = \varsigma_3 \\
\text{(left-}\wedge) \\
| \\
\varsigma_2 = f(\varsigma_1), \varsigma_3 = f(\varsigma_1) \circ \varsigma_2 = \varsigma_3 \\
\text{(right replacement with left identities)} \\
| \\
\varsigma_2 = f(\varsigma_1), \varsigma_3 = f(\varsigma_1) \circ f(\varsigma_1) = f(\varsigma_1) \\
\text{(left-= with } f(\varsigma_1)) \\
\varsigma_2 = f(\varsigma_1), \varsigma_3 = f(\varsigma_1), f(\varsigma_1) = f(\varsigma_1) \circ f(\varsigma_1) = f(\varsigma_1) \\
\times
\end{array}$$

4, p. 76: First rule:

$$\begin{array}{c}
s = t \circ \\
\text{(left-= with } s) \\
| \\
s = s, s = t \circ \\
\text{(left replacement of } s \text{ in } s = s \text{ with } s = t) \\
| \\
t = s, s = s, s = t \circ
\end{array}$$

Second rule:

$$\begin{array}{c}
s = t \circ P(\dots, t, \dots) \\
\text{(previous rule)} \\
| \\
t = s, s = t \circ P(\dots, t, \dots)
\end{array}$$

$$\begin{array}{c}\text{(right replacement of } t \text{ in } P(\dots,t,\dots) \text{ with } t=s)\\ \Big|\\ t=s, s=t\circ P(\dots,t,\dots), P(\dots,s,\dots)\end{array}$$

5b, p. 88: Suppose a CNF C is equivalent to (4). Suppose further that there is a clause is which proposition letter p does not occur. This is impossible, for choose a model m such that $m \not\models p$. Flip valuation of p. Clause is still false, while (4) becomes true. If all proposition letter must occur in every clause, then it is easy to see that there must be as many clauses as there are rows in the corresponding truth table with a 1. Hence, C must be exponentially large.

9, p. 88: Answers: (a) clause set, equivalent with p; (b) clause set, equivalent with `false`; (c) mix of literals and clauses but no clause set; (d) clause set, equivalent with $\texttt{false} \wedge \neg q$; (e) empty clause set, equivalent with `true`; (f) meaningless expression; (g) clause, equivalent with `false`; (h) clause set, equivalent with $(\neg p \vee p\,) \wedge (\neg p \vee p \vee q) \wedge \texttt{false}$.

10, p. 88: Rules can be rewritten into $(\neg p \vee r \vee \neg s \vee t) \wedge (t \vee p\,) \wedge (r \vee \neg q \vee \neg s) \wedge (\neg q \vee \neg p\,)$, which is equivalent with $\{\{\neg p, r, \neg s, t\}, \{t, p\}, \{r, \neg q, \neg s\}, \{\neg q, \neg p\}\}$.

11a, p. 88: Fair enough: $\{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$!

11b, p. 88: Definitely harder than the first one! First, write ϕ as $a \vee (b \vee (c \vee (d \vee e)))$. (Alternatively you may write ϕ as $(((a \vee b) \vee c) \vee d) \vee e$, or $(((a \vee b)) \vee (c \vee d))) \vee e$, or whatever. In each case you end up with another Tseitin derivative.) Then decompose ϕ in subformulas where each formula receives its own letter:

$$\phi \equiv (a \vee r) \wedge (r \equiv (b \vee s)) \wedge (s \equiv (c \vee t)) \wedge (t \equiv (d \vee e)) \tag{B.2}$$

We now use Table 4.1 on page 86 to rewrite each equivalence of the form $u \equiv v \vee p$ into a clause set of the form $\{\{u, \neg v\}, \{u, \neg p\}, \{\neg u, v, p\}\}$. In this way, ϕ is equivalent to

$$\begin{array}{rcl}\phi & \equiv & \{\{a, r\}\} \cup\\ & & \{\{r, \neg b\}, \{r, \neg s\}, \{\neg r, b, s\}\} \cup\\ & & \{\{s, \neg c\}, \{s, \neg t\}, \{\neg s, c, t\}\} \cup\\ & & \{\{t, \neg d\}, \{t, \neg e\}, \{\neg t, d, e\}\}.\end{array}$$

Hence,

$$\phi \equiv \{\{a, r\}, \{r, \neg b\}, \{r, \neg s\}, \{\neg r, b, s\}, \{s, \neg c\}, \{s, \neg t\}, \{\neg s, c, t\}, \{t, \neg d\}, \{t, \neg e\}, \{\neg t, d, e\}\}. \tag{B.3}$$

12, p. 88: How a Tseitin transformation is executed is shown in Example 4.3 on page 85. The size of the clause set can be determined as follows. To prove

$$(p \equiv q) \equiv (q \vee \neg p) \wedge (\neg q \vee p)), \tag{B.4}$$

we will have to transform the negation of (B.4) to $\leq$3CNF. Formula (B.4) consists of nine subformulas: itself, three negations, two equivalences, one conjunction, and two disjunctions. [Construct a syntactical tree of (B.4) if necessary.] In a Tseitin transformation, each negation yields two clauses, each equivalence yields four clauses, while conjunctions, disjunctions and implications each yield three clauses. Together this gives $1 + 3\cdot 2 + 2\cdot 4 + 1\cdot 3 + 2\cdot 3 = 24$ clauses.

13, p. 88: This could be true. It would be a good research question.

1, p. 95: The pair with the smallest frequency, which is $(p, \neg p)$: clause set (4.16) on page 94 would then have only $1 \times 7 = 7$ new clauses of the form $\{C_i, D_j\}$.

2, p. 95: The disjunction $\{\{a,b\},\{c,d\}\} \vee \{\{e,f\},\{g,h\}\}$ reduces to

$$\{\{a,b,e,f\},\{a,b,g,h\},\{c,d,e,f\},\{c,d,g,h\}\}\}.$$

All four clauses are of type $\{C_i, D_j\}$ in the clause set (4.16) on page 94.

3a, p. 95: The classic Davis-Putnam procedure consists of the DP-rule (Eq. 4.14- 4.16 on page 94), supplemented with four simplification rules, viz. OLR, MVF, tautology, and subsumption (Sec. 4.2 on page 89.) The way of working is as follows. First, apply simplification rules until no simplification rule can be applied. Then apply DP with the variable for which the number of positive occurrences times the number of negative occurrences is as small as possible.

None of the four simplification rules can be applied to S, so that the DP-rule must be applied immediately. The product of the number of occurrences of q and $\neg q$ is minimal (viz. 2 times 1 is 2). T It makes therefore sense to apply DP to q and $\neg q$ first. A choice to split on other variables yields more work.

$$\begin{aligned}
S &= \{\{\neg r,\neg s\},\{p,r\},\{\neg s,t\},\{\neg p,\neg r\},\{s,r\},\{q,s\},\{\neg q,\neg t\},\{p,t\},\{\neg p,q\}\}\\
&\equiv \{\{\neg r,\neg s\},\{p,r\},\{\neg s,t\},\{\neg p,\neg r\},\{s,r\},\{p,t\}\} \wedge \{\{q,s\},\{q,\neg p\},\{\neg q,\neg t\}\}\\
&\equiv_v \{\{\neg r,\neg s\},\{p,r\},\{\neg s,t\},\{\neg p,\neg r\},\{s,r\},\{p,t\}\} \wedge (\{\{s\},\{\neg p\}\} \vee \{\{\neg t\}\})\\
&\equiv \{\{\neg r,\neg s\},\{p,r\},\{\neg s,t\},\{\neg p,\neg r\},\{s,r\},\{p,t\}\} \wedge \{\{s,\neg t\},\{\neg p,\neg t\}\}\\
&= \{\{\neg r,\neg s\},\{p,r\},\{\neg s,t\},\{\neg p,\neg r\},\{s,r\},\{p,t\},\{s,\neg t\},\{\neg p,\neg t\}\}
\end{aligned}$$

None of the four simplification rules may be applied to this clause set, so we apply DP again, this time with p:

$$\begin{aligned}
&= \{\{\neg r,\neg s\},\{p,r\},\{\neg s,t\},\{\neg p,\neg r\},\{s,r\},\{p,t\},\{s,\neg t\},\{\neg p,\neg t\}\}\\
&= \{\{\neg r,\neg s\},\{\neg s,t\},\{s,r\},\{s,\neg t\}\} \wedge \{\{p,t\},\{p,r\},\{\neg p,\neg r\},\{\neg p,\neg t\}\}\\
&\equiv_v \{\{\neg r,\neg s\},\{\neg s,t\},\{s,r\},\{s,\neg t\}\} \wedge \{\{t,\neg r\},\{t,\neg t\},\{r,\neg r\},\{r,\neg t\}\}
\end{aligned}$$

Simplification with the tautology rule yields:

$$\begin{aligned}
&\equiv_v \{\{\neg r,\neg s\},\{\neg s,t\},\{s,r\},\{s,\neg t\}\} \wedge \{\{t,\neg r\},\{r,\neg t\}\}\\
&= \{\{\neg r,\neg s\},\{\neg s,t\},\{s,r\},\{s,\neg t\},\{t,\neg r\},\{r,\neg t\}\}
\end{aligned}$$

The variable r occurs two times positively and two times negatively. Applying DP with r yields:

$$\equiv_v \{\{\neg s,t\},\{s,\neg t\},\{\neg s,s\},\{t,\neg s\},\{\neg t,s\},\{\neg t,t\}\}$$

Cleaning up with the tautology rule yields:

$$\equiv \{\{\neg s,t\},\{s,\neg t\},\{t,\neg s\},\{\neg t,s\}\}$$

At this point we miay apply DP either s or t. We choose s:

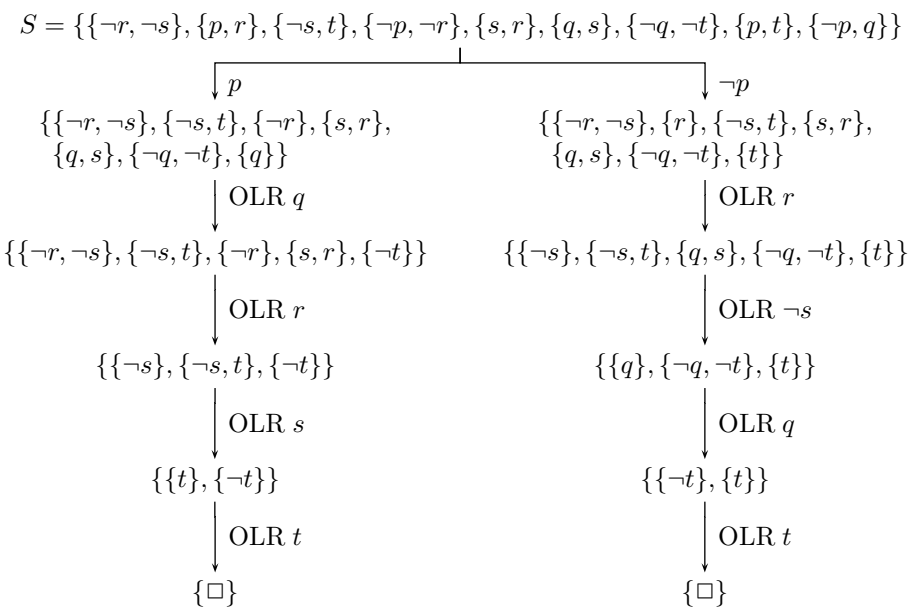
$$\equiv_v \{\{\neg t,t\},\{\neg t\},\{t\},\{t,\neg t\}\}$$

Applying the one-literal rule with t yields:

$$\equiv_v \{\{\neg t,t\},\{t,\neg t\},\square\}$$

We have inferred the empty clause. This means that S is unsatisfiable.

3b, p. 95:



Since OLR is satisfiability-invariant, $S \cup \{p\}$ and $S \cup \{\neg p\}$ are unsatisfiable. Hence, S is unsatisfiable.

4a, p. 95: $\{\{\neg a, b\}, \{\neg b, c\}, \{\neg c, d\}, \{\neg d\}\}$
OLR with $\neg d$: $\{\{\neg a, b\}, \{\neg b, c\}, \{\neg c\}\}$
OLR with $\neg c$: $\{\{\neg a, b\}, \{\neg b\}\}$
OLR with $\neg b$: $\{\{\neg a\}\}$
satisfiable with $\neg a$.

Hence, original clause set satisfiable with $\neg a$, and with $\neg b$, $\neg c$, $\neg d$.

4b, p. 95: $\{\{a\}, \{\neg a, b\}, \{\neg b, \neg c, d\}, \{\neg d, e\}, \{e\}\}$
OLR with a $\{\{b\}, \{\neg b, \neg c, d\}, \{\neg d, e\}, \{e\}\}$
OLR with b $\{\{\neg c, d\}, \{\neg d, e\}, \{e\}\}$
OLR with e $\{\{\neg c, d\}\}$
satisfiable with $\neg c$, d.

Hence, original clause set satisfiable with $\neg c$, d, and e, b, a.

4c, p. 95: This clause set is an exhaustive enumeration of all "possible worlds" that can be made with a, b, and c. Clearly, no simple reduction rule can be applied. Therefore, we will have to split. Since this clause set is symmetrical the choice of the splitting variable does not matter.

$\{\{a, b, c\}, \{a, b, \neg c\}, \{a, \neg b, c\}, \{a, \neg b, \neg c\}\} \cup$
$\{\{\neg a, b, c\}, \{\neg a, b, \neg c\}, \{\neg a, \neg b, c\}, \{\neg a, \neg b, \neg c\}\}$
SPLIT on a, branch a: $\{\{b, c\}, \{b, \neg c\}, \{\neg b, c\}, \{\neg b, \neg c\}\}$
SPLIT on b, branch b: $\{\{c\}, \{\neg c\}\}$
OLR with c: $\{\square\}$

Hence, the original clause set is unsatisfiable in branch $a - b - c$. So this branch does not yield a counter-model. Similarly, other branches (like $a - b - \neg c$, $a - \neg b - c$, and $a - \neg b - \neg c$) will not yield counter-models either. We can conclude that the original clause set is unsatisfiable.

4d, p. 95: We aim at applying the OLR at a second reduction. therefore, we may split with either b or d.

$$\begin{array}{ll} & \{\{\neg a, \neg b, c\}, \{\neg a, \neg c, d\}, \{a, b, d\}, \{\neg b, \neg d\}, \{\neg a, b, d\}\} \\ \text{SPLIT on } b\text{, in branch } b\text{:} & \{\{\neg a, c\}, \{\neg a, \neg c, d\}, \{\neg d\}\} \\ \text{OLR with } \neg d\text{, in branch } b\text{:} & \{\{\neg a, c\}, \{\neg a, \neg c\}\} \\ \text{MTV with } \neg a\text{, in branch } b\text{:} & \{\{c\}, \{\neg c\}\} \\ \text{OLR with } c\text{, in branch } b\text{:} & \{\square\} \end{array}$$

Hence, the branch where b is assumed true does not yield a counter-model. We proceed after the split on b in the branch where $\neg b$ is the case.

$$\begin{array}{ll} & \{\{\neg a, \neg b, c\}, \{\neg a, \neg c, d\}, \{a, b, d\}, \{\neg b, \neg d\}, \{\neg a, b, d\}\} \\ \text{SPLIT on } b\text{, in branch } \neg b\text{:} & \{\{\neg a, \neg c, d\}, \{a, d\}, \{\neg a, d\}\} \\ \text{MTV with } d\text{, in branch } \neg b\text{:} & \varnothing \end{array}$$

This branch is satisfiable with b false and d true. (We don't care about the values of the other variables.)

5, p. 95:

$$\begin{aligned} &\{\{p, q\}, \{\neg p, D_1\}, \ldots, \{\neg p, D_l\}, \ E_1, \ldots, E_m\} \\ &\quad = \{\{p, C\}, \{\neg p, D_1\}, \ldots, \{\neg p, D_l\}, \ E_1, \ldots, E_m\} \\ &\quad = \{\{C, D_1\}, \ldots, \{C, D_l\}, \ E_1, \ldots, E_m\} && \text{by (4.16) on page 94} \\ &\quad = \{\{q, D_1\}, \ldots, \{q, D_l\}, \ E_1, \ldots, E_m\}. \end{aligned}$$

6, p. 95: Let

$$\{\{p, L_1\}, \ldots, \{p, L_n\}, \ C_1, \ldots, C_m\}$$

be given. First, we separate clauses containing $\neg p$ from clauses that do not contain $\neg p$:

$$\{\{p, L_1\}, \ldots, \{p, L_n\}, \ \{\neg p, D_1\}, \ldots, \{\neg p, D_l\}, \ E_1, \ldots, E_k\}.$$

Applying DPP yields

$$\begin{array}{llll} \{\{L_1, D_1\}, & \ldots, & \{L_1, D_l\}, \\ \vdots & & \vdots & \\ \{L_n, D_1\}, & \ldots, & \{L_n, D_l\}, E_1, \ldots, E_k\}. \end{array}$$

Mirroring this into the main diagonal yields

$$\begin{array}{llll} \{\{L_1, D_1\}, & \ldots, & \{L_n, D_1\}, \\ \vdots & & \vdots & \\ \{L_1, D_l\}, & \ldots, & \{L_n, D_l\}, \\ E_1, & & & \\ \vdots, & & & \\ E_k\} & & & \end{array}$$

Since no D_i contains a $\neg p$ we may write

$$\begin{array}{llll} \{\{\neg p, D_1\}[\neg p/L_1], & \ldots, & \{\neg p, D_1\}[\neg p/L_n], \\ \vdots & & \vdots & \\ \{\neg p, D_l\}[\neg p/L_1], & \ldots, & \{\neg p, D_l\}[\neg p/L_n], \\ E_1, & & & \\ \vdots, & & & \\ E_k\} & & & \end{array}$$

which is just

$$\begin{array}{l}\{C_1[\neg p/L_1], \quad \ldots, \quad C_1[\neg p/L_n],\\ \quad\vdots \qquad\qquad\qquad\quad \vdots\\ \quad C_l[\neg p/L_1], \quad \ldots, \quad C_l[\neg p/L_n],\\ \quad C_{l+1},\\ \quad\vdots,\\ \quad C_m\}.\end{array}$$

Applying the distributive law for disjunction yields

$$\begin{array}{l}\{C_1[\neg p/L],\\ \quad\vdots\\ \quad C_l[\neg p/L],\\ \quad C_{l+1},\\ \quad\vdots,\\ \quad C_m\}.\end{array}$$

Because no C_i contains a $\neg p$, for $i > l$, this amounts to

$$\{C_1[\neg p/L], \ldots, C_m[\neg p/L]\}.$$

5a, p. 100: Suppose there are n different letters. This makes at most $2n$ different literals. (I.e., literals and their negations of which some do not occur in the clause set.) Each literal can be coupled to $2n$ other literals. (If it is the same literal, we have a unit clause, if it is an opposite literal, we have the empty clause.) This makes less than $2n^2 = 4n^2$ possible different clauses. (This number includes doublures since inconsistent clauses all reduce to the empty clause and all non-unit clauses are counted double.) In any case, $|R|$ depends polynomially on n.

5b, p. 100: Hint: use a standard form of resolution. Then try to deduce from the resolution strategy a maximum number of resolution steps.

1, p. 102: Try $\{\{p\}, \{\neg q\}, \{\neg p, q\}\}$. This clause set is unsatisfiable, while a non-restricted binary resolution refutation would use the mixed clause $\{\neg p, q\}$.

2, p. 102: The picture shows that an input resolution per sé need not be linear. However, all input resolution *refutations* are linear. With an (informal) inductive argument, it is easy to see that an input resolution refutation is actually a linear refutation in which every side-clause is an input clause.

1a, p. 106: One possible refutation:

1.	$\{\neg d, e\}$	[input]
2.	$\{a, \neg d, \neg e, f\}$	[input]
3.	$\{d\}$	[input]
4.	$\{\neg a, \neg d, g\}$	[input]
5.	$\{\neg e, \neg f, g\}$	[input]
6.	$\{\neg g\}$	[input]
7.	$\{e\}$	[hyper,1,3]
8.	$\{a, f\}$	[hyper,2,3,7]
9.	$\{a, g\}$	[hyper,8,5,7]
10.	$\{a\}$	[hyper,9,6]
11.	$\{g\}$	[hyper,10,4,3]
12.	$\square$	[binary,11,6]

1b, p. 106: One possible refutation:

1.	$\{ d, a, \neg f, g\}$	[input]
2.	$\{ \neg a, \neg f\}$	[input]
3.	$\{ \neg d\}$	[input]
4.	$\{ f, b\}$	[input]
5.	$\{ \neg g, a, b\}$	[input]
6.	$\{ \neg b, d\}$	[input]
7.	$\{ d, g, a, b\}$	[hyper,1,4]
8.	$\{ d, a, b\}$	[hyper,7,5]
9.	$\{ a, b\}$	[hyper,8,3]
10.	$\{ a, d\}$	[hyper,9,6]
11.	$\{ a\}$	[hyper,10,3]
12.	$\{ b\}$	[hyper,11,2,4]
13.	$\{ d\}$	[hyper,12,6]
14.	$\square$	[binary,13.1,3.1]

1c, p. 106: One possible refutation:

1.	$\{ -g, d\}$	[input]
2.	$\{ g, b, -c\}$	[input]
3.	$\{ -d, b, -c\}$	[input]
4.	$\{ -b, g\}$	[input]
5.	$\{ -d, -b\}$	[input]
6.	$\{ g, c\}$	[input]
7.	$\{ c, -d\}$	[input]
8.	$\{ c, d\}$	[hyper,6,1]
9.	$\{ c\}$	[hyper,8,7]
10.	$\{ g, b\}$	[hyper,2,9]
11.	$\{ b, d\}$	[hyper,10,1]
12.	$\{ b\}$	[hyper,3,11,9]
13.	$\{ g\}$	[hyper,12,4]
14.	$\{ d\}$	[hyper,13,1]
15.	$\square$	[hyper,14,5,12]

2, p. 106: One possible refutation:

1.	$\{ -g, d\}$	[input]
2.	$\{ g, b, -c\}$	[input]
3.	$\{ -d, b, -c\}$	[input]
4.	$\{ -b, g\}$	[input]
5.	$\{ -d, -b\}$	[input]
6.	$\{ g, c\}$	[input]
7.	$\{ c, -d\}$	[input]
8.	$\{ \neg b, \neg g\}$	[neg. hyper,5,1]
9.	$\{ \neg b\}$	[neg. hyper,8,4]
10.	$\{ \neg d, \neg c\}$	[neg. hyper,3,9]
11.	$\{ \neg c, \neg g\}$	[neg. hyper,10,1]
12.	$\{ \neg c\}$	[neg. hyper,11,2,9]
13.	$\{ \neg d\}$	[neg. hyper,12,7]
14.	$\{ \neg g\}$	[neg. hyper,13,1]
15.	$\square$	[neg. hyper,14,6,12]

1a, p. 108: We follow the prescribed steps:

1. The denial of $\vdash (\forall x)(px \wedge \neg px)$ in formula-form is $\neg(\forall x)(px \wedge \neg px)$.
2. Nothing to do: there are no exotic connectives here unequal to $\neg$, $\wedge$ and $\vee$.
3. Nothing to do: every quantifiers has its own variable.
4. Pulling all quantifiers to the outside yields $(\exists x)\neg(px \wedge \neg px)$.
5. Skolemization yields $\neg(pa \wedge \neg pa)$.
6. Nothing to do here: no universal quantifiers.
7. Rewriting in CNF yields $\neg pa \vee pa$.
8. Writing this as a clause set gives $\{\{\neg pa, pa\}\}$.
9. Nothing to do: no variables.

This clause set is satisfiable. (In this case it is even a tautology.) Hence, no sound resolution method should be able to derive the empty clause from this clause set.

1b, p. 108: We follow the prescribed steps:

1. The denial of $\vdash (\exists x)(px \vee \neg px)$ in formula-form is $\neg(\exists x)(px \vee \neg px)$.
4. Pulling all quantifiers to the outside yields $(\forall x)\neg(px \vee \neg px)$.
5. Nothing to do here: no existential quantifiers.
6. Deleting all (universal) quantifiers yields $\neg(px \vee \neg px)$
7. Rewriting in CNF yields $\neg px \wedge px)$.
8. Writing this as a clause set gives $\{\{px\}, \{\neg px\}\}$.
9. Standardizing variables apart yields $\{\{px\}, \{\neg py\}\}$.

This clause says that for every x and y: px and $\neg py$. This is obviously not true if $x = y$. Hence, every decent resolution method should unify the literals of both clauses to apply a resolution step in which the empty clause is formed.

4, p. 108: A possible rectification of

$$(\forall x)(rx, y \supset (\exists z)(\neg(\exists x)(rx, z))) \wedge (\forall z)((\forall n)(tn, x, z \supset ((\forall z \exists u \exists u)px, u, u)))$$

is

$$(\forall v)(rv, y \supset (\exists s)(\neg(\exists q)(rq, s))) \wedge (\forall z)((\forall n)(tn, x, z \supset ((\forall w \exists r \exists u)px, u, u))).$$

Moving all quantifiers to the front yields

$$(\forall v \exists s \forall q \forall z \forall w \exists r \exists u \exists n)((rv, y \supset \neg rq, s) \wedge (tn, x, z \supset px, u, u)).$$

Skolemizing s, r, u and n yields

$$(\forall v \forall q \forall z \forall w)((rv, y \supset \neg rq, \varsigma_s(v)) \wedge (t\varsigma_n(v, q, z, w), x, z \supset px, \varsigma_u(v, q, z, w), \varsigma_u(v, q, z, w))).$$

The quantifier for r has disappeared, because it is irrelevant. Dropping all quantifiers yields

$$(rv, y \supset \neg rq, \varsigma_s(v)) \wedge (t\varsigma_n(v, q, z, w), x, z \supset px, \varsigma_u(v, q, z, w), \varsigma_u(v, q, z, w)).$$

Writing this in CNF yields

$$(\neg rv, y \vee \neg rq, \varsigma_s(v)) \wedge (\neg t\varsigma_n(v,q,z,w), x, z \vee px, \varsigma_u(v,q,z,w), \varsigma_u(v,q,z,w)),$$

which is

$$\{\{\neg rv, y \vee \neg rq, \varsigma_s(v)\}, \{\neg t\varsigma_n(v,q,z,w), x, z \vee px, \varsigma_u(v,q,z,w), \varsigma_u(v,q,z,w)\}\}$$

in clause set notation. Standardizing variables apart yields

$$\{\{\neg rv, y \vee \neg rq, \varsigma_s(v)\}, \{\neg t\varsigma_n(v_2,q_2,z,w), x, z \vee px, \varsigma_u(v_2,q_2,z,w), \varsigma_u(v_2,q_2,z,w)\}\}.$$

3, p. 120: We write $\bar{x} = x_1, \ldots, x_n$ for the sake of brevity. First, let us suppose that $(\forall\bar{x})(\exists y)(\phi)$ is satisfiable. We'll have to prove that $(\forall\bar{x})\phi[\varsigma(\bar{x})/y]$ is satisfiable as well, for some function symbol ς. Well, since $(\forall\bar{x})(\exists y)(\phi)$ is satisfiable, there is a model M and a valuation s, such that

$$(M,s) \models (\forall\bar{x})(\exists y)(\phi). \tag{B.5}$$

Why not use (M,s) to prove the satisfiability of $(\forall\bar{x})\phi[\varsigma(\bar{x})/y]$? According to (B.5) we have

$$\text{for all } \bar{d} \in D^n \text{ there is a } d* \in D \text{ such that } (M, s[\bar{d}/\bar{x}][d^*/y]) \models \phi. \tag{B.6}$$

Thus, it is possible to introduce a new function symbol ς and a new function ς_M in M:

$$\varsigma_M : D^n \to D : \bar{d} \longmapsto d^*$$

If, as the notation suggests, the function symbol ς is interpreted as ς_M, then indeed $(M,s) \models (\forall\bar{x})(\phi[\varsigma(\bar{x})/y])$:

$(M,s) \models (\forall\bar{x})\phi[\varsigma(\bar{x})/y]$ iff
for all $\bar{d} \in D^n$: $(M, s[\bar{d}/\bar{x}]) \models \phi[\varsigma(\bar{x})/y]$ iff [substitution lemma, p. 56]
for all $\bar{d} \in D^n$: $(M, s[\bar{d}/\bar{x}][\ (\varsigma(\bar{x}))_{M,s[\bar{d}/\bar{x}]}/y\ ]) \models \phi$ iff
for all $\bar{d} \in D^n$: $(M, s[\bar{d}/\bar{x}][\varsigma_M(\bar{d})/y]) \models \phi$ only if $(B.6)$.

The last expression is true, since $\varsigma_M(\bar{d})$ is defined in accordance with (B.6).

Conversely, suppose $(\forall\bar{x})\phi[\varsigma(\bar{x})/y]$ is satisfiable, for some function symbol ς. Then, for some M and s,

$$(M,s) \models (\forall\bar{x})\phi[\varsigma(\bar{x})/y].$$

Hence, by way of the substitution lemma (56) we have that

$$\text{for all } \bar{d} \in D^n : (M, s[\bar{d}/\bar{x}][\varsigma_M(\bar{d})/y]) \models \phi.$$

Thus,

$$\text{for all } \bar{d} \in D^n \text{ there is a } d^* \in D : (M, s[\bar{d}/\bar{x}][d^*/y]) \models \phi,$$

so that

$$(M,s) \models (\forall\bar{x})(\exists y)(\phi).$$

Hence, $(\forall\bar{x})(\exists y)(\phi)$ is satisfiable.

4, p. 120:

	Demodulants
(1)	$D(f(x,x),a)$
(2)	$D(a)$
(3)	$D(f(a,a),f(a,x))$
(4)	$D(f(f(a,a),f(a,x)))$
(5)	$D(f(a,a),f(a,x))$
(6)	$D(f(f(a,a),f(a,x)))$

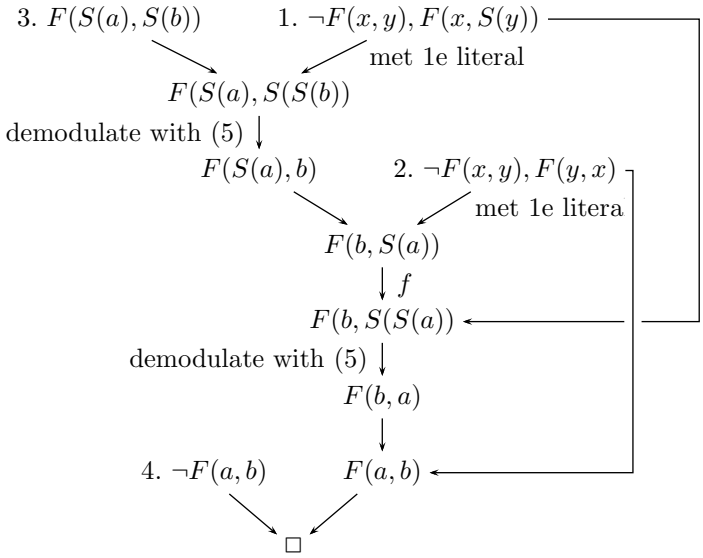
5, p. 120: We translate the above natural-language expressions into the language of first-order logic as follows. For friendship we take the two-place predicate letter $F/2$, and for spouse we take the one-place predicate letter $S/1$. Alice and Betty are the constants a and b, respectively.

Translation:

1.	$F(x,y) \supset F(x,S(y))$	If two persons are friends, then the first person is also ...
2.	$F(x,y) \supset F(y,x)$	Friendship is symmetrical.
3.	$F(S(a),S(b))$	Alice's and Betty's spouses are friends.
4.	$F(a,b)$?	Are Alice and Betty friends?
5.	$S(S(x)) = x$	Married persons are the spouse of their spouse.

In clause set notation:

1.	$\{\neg F(x,y), F(x,S(y))\}$	If two persons are friends, then ...
2.	$\{\neg F(x,y), F(y,x)\}$	Friendship is symmetrical.
3.	$\{F(S(a),S(b))\}$	Alice's and Betty's spouses are friends.
4.	$\{\neg F(a,b)\}$	The denial of the question "Are Alice and Betty friends?"
5.	$\{S(S(x)) = x\}$	Married persons are the spouse of their spouse.



All nodes with two parents are the result of binary resolution.

6, p. 120: Otter starts a left-most inner-most reduction using $f(x,y) \to x + 2y$: $D(f(1+2, 7-3)) \to D(f(3, 7-3)) \to D(f(3,4)) \to D(3 + 2 \cdot 4) \to D(11)$:

```
=========== start of search ===========

given clause #1: (wt=8) 1 [] D(f($SUM(1,2),$DIFF(7,3))).
```

```
  0 [1] D(f($SUM(1,2),$DIFF(7,3))).
   demod term: 1.
   demod term: 2.
   demod term: $SUM(1,2).
   demod term: 7.
   demod term: 3.
   demod term: $DIFF(7,3).
   demod term: f(3,4).
   --> result: $SUM(3,$PROD(2,4))   demod<2>
   demod term: 2.
   demod term: $PROD(2,4).
   demod term: $SUM(3,8).
   demod term: D(11).
  after demodulation: 0 [1,demod,2] D(11).
** KEPT (pick-wt=2): 3 [1,demod,2] D(11).    [The one and
                                              only clause
Search stopped because sos empty.             generated and kept]

============ end of search ============
```

7, p. 121: Starting with the unit clause $P(2,3)$, Otter will repeatedly select a clause of the form $P(x,y)$ to produce (and keep) a clause of the form $P(x+y,xy)$:

```
=========== start of search ===========

given clause #1: (wt=3) 2 [] P(2,3).
** KEPT (pick-wt=3): 3 [hyper,2,1,demod] P(5,6).

given clause #2: (wt=3) 3 [hyper,2,1,demod] P(5,6).
** KEPT (pick-wt=3): 4 [hyper,3,1,demod] P(11,30).

given clause #3: (wt=3) 4 [hyper,3,1,demod] P(11,30).
** KEPT (pick-wt=3): 5 [hyper,4,1,demod] P(41,330).

given clause #4: (wt=3) 5 [hyper,4,1,demod] P(41,330).
** KEPT (pick-wt=3): 6 [hyper,5,1,demod] P(371,13530).

given clause #5: (wt=3) 6 [hyper,5,1,demod] P(371,13530).
** KEPT (pick-wt=3): 7 [hyper,6,1,demod] P(13901,5019630).

... and so forth.
```

This process is never-ending unless you set bounds on the maximum number of clauses that may be selected, e.g., `assign(max_given,10)` or on the maximum number of rewrites that will be applied when demodulating a clause, includes $-symbol evaluation, e.g., `demod_limit,10)`.

9, p. 122: Voor de eenvoud uitgewerkt met binaire resolutie (i.p.v. hyperresolutie). Oefening (9) is ten behoeve van de lengte met hyperresolutie gedaan.

(1)

1.	$\neg P \mid Q(a,x) \mid Q(y,b)$	input
2.	P	input
3.	$Q(a,x) \mid Q(y,b)$	binair, 2.1, 1.1
4.	$Q(a,b)$	factor, 3.1.2

N.B. “binair, 2.1, 1.1” betekent dat de clause op deze regel verkregen is door literal 1 van parent clause 2 (2.1) te clashen met literal 1 van parent clause 1 (1.1).

(2)

1.	$\neg P \mid Q(a,x) \mid Q(y,b)$	input
2.	$\neg Q(a,b)$	input
3.	P	input
4.	$Q(a,x) \mid Q(y,b)$	binair, 3.1, 1.1
5.	$Q(a,b)$	factor, 4.1.2
6.	$\square$	binair, 5.1, 2.1

(3)

1.	$\neg P \mid Q(a,x) \mid Q(y,b)$	input
2.	P	input
3.	$\neg Q(a,b)$	input
4.	$Q(a,x)$	binair, 3.1, 1.3, en dan OLR met 2
5.	$\square$	binair, 4.1, 3.1

(4)

1.	$\neg P \mid Q(a,x)$	input
2.	$P \mid Q(x,b)$	input
3.	$Q(x,b) \mid Q(a,y)$	binair, 2.1, 1.1
4.	$Q(a,b)$	factor, 3.1.2

(5)

1.	$\neg P \mid Q(a,x)$	input
2.	$\neg Q(a,b)$	input
3.	$P \mid Q(x,b)$	input
4.	$Q(x,b) \mid Q(a,y)$	binair, 3.1, 1.1
5.	P	binair, 3.2, 2.1
6.	$Q(a,b)$	factor, 4.1.2
7.	$\square$	binair, 6.1, 2.1

(6)

1.	$P(a,x) \mid P(y,b)$	input
2.	$\neg P(a,b) \mid Q$	input
3.	$Q \mid P(a,x)$	binair, 2.1, 1.2
4.	$Q \mid P(x,b)$	binair, 2.1, 1.1
5.	Q	binair, 3.2, 2.1, factor simplificatie

Factor simplificatie wil zeggen dat een clause gegenereerd werd door het factoriseren van een input clause. Als bijvoorbeeld $P(x) \mid P(a)$ een input clause is, dan factor-simplificeert deze tot $P(a)$.

(7)

1.	$P(a,x) \mid P(y,b)$	input
2.	$\neg Q$	input
3.	$\neg P(a,b) \mid Q$	input
4.	$P(a,x)$	binair, 3.1, 1.2, en dan OLR met 2
5.	$P(x,b)$	binair, 3.1, 1.1, en dan OLR met 2
6.	$\neg P(a,b)$	binair, 3.2, 2.1
7.	$\square$	binair, 6.1, 5.1

(8)

1.	$P(a,x) \mid P(y,b)$	input
2.	Q	input
3.	$\neg P(a,b) \mid \neg Q \mid R$	input
4.	$R \mid P(a,x)$	binair, 3.1, 1.2, en dan OLR met 2
5.	$R \mid P(x,b)$	binair, 3.1, 1.1, en dan OLR met 2
6.	$\neg P(a,b) \mid R$	binair, 3.2, 2.1
7.	R	binair, 6.1, 5.2, factor simplificatie

(9) Met hyperresolutie:

1.	$P(a,x,y,z) \mid P(u,b,y,z)$	input
2.	$\neg P(x,y,c,z) \mid \neg P(x,y,u,d)$	input
3.	$P(a,x,c,y) \mid P(a,z,u,d)$	hyper, 2, 1, 1
4.	$P(a,x,c,y) \mid P(z,b,u,d)$	hyper, 2, 1, 1
5.	$P(x,b,c,y) \mid P(a,z,u,d)$	hyper, 2, 1, 1
6.	$P(x,b,c,y) \mid P(z,b,u,d)$	hyper, 2, 1, 1
7.	$P(a,x,c,d)$	factor, 3.1.2
8.	$P(x,b,c,d)$	factor, 6.1.2
9.	$\square$	hyper, 7, 2, 7

(10)

1.	$\neg P \mid Q(x) \mid Q(y)$	input
2.	$\neg Q(u) \mid \neg Q(v)$	input
3.	P	input
4.	$Q(x)$	binair, 3.1, 1.1, factor simplificatie
5.	$\neg Q(x)$	binair, 4.1, 2.2
6.	$\square$	binair, 5.1, 4.1

(12)

1.	$\neg P \mid Q(x) \mid Q(y) \mid R$	input
2.	$\neg Q(u) \mid \neg Q(v)$	input
3.	$\neg R$	input
4.	P	input
5.	$Q(x)$	binair, 4.1, 1.1, en dan OLR met 3, factor simplificatie
6.	$\neg Q(x)$	binair, 5.1, 2.2
7.	$\square$	binair, 6.1, 5.1

(14)

1.	$P(a) \mid P(b)$	input
2.	$\neg P(x) \mid \neg P(y)$	input

10, p. 122: (i) $Pf(a)$ reduceert naar $Pg(a,y)$. Laatste is niet verder reduceerbaar.

(ii) De formule $Pg(a,b)$ is niet reduceerbaar. (Demodulatie werkt van links naar rechts, niet omgekeerd.)

(iii) $Pa \to Pf(a) \to Pf(f(a)) \to \ldots$ Reductie termineert niet.

11, p. 122: (c) Nee.

12, p. 122: Zie Fig. 5.1 on page 118.

13, p. 122: (i) 2e generatie: Pa; 3e generatie: Pb.

(ii) 2e generatie: Qd; 3e generatie: Qc.

(iii) 2e generatie: Rf, f en Re, e; 3e generatie: Re, f en Rf, e.

14, p. 122: Alvorens de clause $h(x) = b \mid Rx$ in de clause $Sb \mid Th(x)$ te paramoduleren halen we ze "uit elkaar," door alle voorkomens van x in $Sb \mid Th(x)$ te vervangen door y.

De opdracht is nu $h(x) = b \mid Rx$ te paramoduleren in $Sb \mid Th(y)$.

1. Eerst bekijken we of er termen voorkomend in literals van $Sb \mid Th(y)$ te unificeren zijn met de rechterkant van de gelijkheid in $h(x) = b \mid Rx$. Dat kan: de term b in Sb kan door middel van de MGU (most general unifier) $\sigma = \varnothing$ met de rechterkant van $h(x) = b$ worden geünificeerd. We vervangen b voorkomend in $Sb \mid Th(y)$ door $h(x)$, en voegen daar de het restanten van de "vanuit"-clause bij; de eerste paramodulant wordt dus $Sh(x) \mid Th(y) \mid Rx$.

De clause $Sb \mid Th(y)$ bezit meer literals met b-unificeerbare termen. De term y in literal $Th(y)$ kan ook met de rechterkant van $h(x) = b$ worden geünificeerd, en wel door middel van de MGU $\tau = \{b/y\}$. We vervangen het tweede voorkomen van b in $Sb \mid Th(b)$ door $h(x)$, en voegen daar de het restanten van de "vanuit"-clause bij; een tweede paramodulant wordt dus $Sb \mid Th(h(x)) \mid Rx$. Met het eerste voorkomen van b gebeurd in dit geval niets: deze werd niet geparamoduleerd.

2. Vervolgens bekijken we of er termen voorkomend in literals van $Sb \mid Th(y)$ te unificeren zijn met de *linkerkant* van de gelijkheid in $h(x) = b \mid Rx$. Dat kan: de term $h(y)$ in $Th(y)$ kan worden geünificeerd met de linkerkant van het $=$-teken door middel van de MGU $\rho = \{x/y\}$. Vervolgens vervangen we $h(x)$ in $Sb \mid Th(x)$ door b, en voegen daar de het restanten van de "vanuit"-clause bij; de derde paramodulant wordt hiermee $Sb \mid Tb \mid Rx$.

   Behalve $h(y)$ bevat $Th(y)$ meer $h(x)$-unificeerbare termen, namelijk de variable y zélf. De variabele y in $Th(y)$ kan met de linkerkant van $h(x) = b$ worden geünificeerd door middel van de MGU $\mu = \{h(x)/y\}$. Door $h(x)$ in $Sb \mid Th(h(x))$ door b te vervangen daar de het restanten van de "vanuit"-clause bij te voegen ontstaat een vierde paramodulant.

Samenvattend:

$Sh(x) \mid Th(y) \mid Rx$,
$Sb \mid Th(h(x)) \mid Rx$,
$Sb \mid Tb \mid Rx$, en
$Sb \mid Th(b) \mid Rx$.

1a, p. 131: Ninety-nine. (Note the second literal of every clause: as from the fifth clause, the number of the second literal of every clause indicates the clause number.)

1b, p. 131: The truth value of p_1 must be 1 on pain of not satisfying the first five clauses. (If $p_1 = 0$, we would have $2 \supset 3$, $3 \supset 4$, $4 \supset \neg 2$, so that p_2 must be false. However, the 4th and the 5th clause say that p_2 must be true.)

1c, p. 131: To make the first five clauses true, we set $p_1 = 1$. To make the rest true, we set $p_i = 0$, $i > 1$.

1d, p. 131: If $p_1 = 1$ then GSAT would select p_1 as a candidate variable to flip. If p_1 would indeed be selected to flip, then GSAT would move in the wrong direction. If $p_1 = 0$ then GSAT would *not* select p_1 as a candidate variable to flip.

1e, p. 131: Take a truth assignment very close to the optimal solution.

5a, p. 144: Hint: showing that fuzzy-$\neg$ is surjective (onto) and continuous can be done in one turn. First show what it means for the graph of $\neg$ if fuzzy negation is a conservative extension of classical negation. Then show what it means for the graph of $\neg$ if fuzzy negation is self-invers. Finally show that these graphical constraints are violated if we assume that $\neg$ is discontinuous or non-surjective.

5c, p. 144: The function

$$f(x) = [(1 - \neg x) + x]/2$$

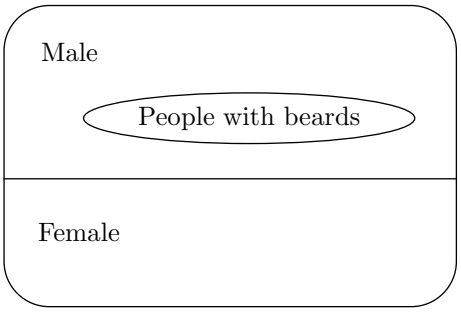
does the job. (If you don't believe, please verify.) The function f is a so-called *automorphism* on $[0, 1]$. An automorphism on $[0, 1]$ is a surjective function that respects the ordering on $[0, 1]$.

That is, if $x < y$ then $f(x) < f(y)$ [which, by the way, immediately implies $x < y \Leftrightarrow f(x) < f(y)$]. The relation $f(\neg x) = 1 - f(x)$, is more commonly written as

$$\neg = f^{-1}\alpha f, \tag{B.7}$$

where α is the "standard" negation $x \rightarrow 1 - x$. If we define an *anti-automorphism* as a surjective strictly decreasing function on $[0, 1]$, then we may say that, in the group of automorphisms and anti-automorphisms $Map([0, 1])$, every fuzzy negation $\neg$ is a so-called *conjugate* (Eq. B.7) of the "standard" negation α. This is an important result, because it says that all fuzzy negations are equal up to some stretching of $[0, 1]$.

1, p. 156: In the following Venn-diagram, most men don't wear beards while it is not the case that most people wearing a beard are not men:



To understand what is at stake here, it may be helpful to compare this defeasible implication with its categorical (material) counterpart, namely, Man $\supset \neg$Beard. The latter (unrealistic!) implication does not allow beard wearers to be men at all. With such a strong constraint, the contraposition vacuously holds: $\neg\neg$Beard $\supset \neg$Man, i.e., Beard $\supset$ Woman.

2, p. 156: Deze gevolgtrekking is niet waar. Intuïtief voorbeeld: "trouwen met Paula is Rechtmatig; trouwen met Quirine is Rechtmatig; maar trouwen met beiden is niet Rechtmatig" (polygamie). Merk op dat, voor elk drietal verzamelingen P, Q en R de gevolgtrekking wél geldt. Immers, als $P \subseteq R$ en $Q \subseteq R$, dan automatisch $P \cap Q \subseteq R$. Een tegenvoorbeeld in de ϵ-semantiek zal dus noodzakelijkerwijs een limiet-element moeten bezitten.

We zoeken $\{P_n \mid n \geq 0\}$, en $\{Q_m \mid m \geq 0\}$ en $\{R_k \mid k \geq 0\}$ zó dat, als n, m, en k naar oneindig gaan, de kansen $\Pr(R \mid P)$ en $\Pr(R \mid Q)$ naar 1 gaan, zonder dat de kans $\Pr(R \mid P, Q)$ naar 1 gaat. Dat kan als we $R_k \backslash (P_n \cap Q_m)$ vullen met één (statisch) element, en $R_k \cap P_n$ en $R_k \cap Q_m$ vullen met een aantal elementen dat toegroeit naar oneindig. Dat kan als $n = m = k$, en

$$\begin{array}{lll} P_n & =_{Def} & \{2k \mid k \in \mathbb{Z},\ 0 \leq k \leq n\} \qquad (= \{0, 2, 4, \ldots, 2n\}) \\ Q_n & =_{Def} & \{2k+1 \mid k \in \mathbb{Z},\ 0 \leq k \leq n\} \cup \{0\} \qquad (= \{1, 3, 5, \ldots, 2n+1\} \cup \{0\}) \\ R & =_{Def} & \mathbb{N} \qquad (= \{1, 2, 3, \ldots\}) \end{array} \tag{B.8}$$

Merk op dat $P_n \nsubseteq R$ en $Q_n \nsubseteq R$ (immers $0 \notin R$) maar wel

- $\Pr(R \mid P_n) = (2n - 1)/2n$.
- $\Pr(R \mid Q_n) = (2n - 1)/2n$.

Bovendien:

- $\Pr(R \mid P_n \cap Q_n) = |P_n \cap Q_n \cap R|/|P_n \cap Q_n| = 0/1 = 0$.

Als $n \rightarrow \infty$ ontstaat er een tegenvoorbeeld.

3a, p. 156: We prove the cumulativity rule by showing that

$$\frac{Pr(Q|P) \rightsquigarrow 1 \quad Pr(R|P) \rightsquigarrow 1}{Pr(R|P,Q) \rightsquigarrow 1} \tag{B.9}$$

for probabilistic events P, Q and R. We prove (B.9) by showing that, for small ϵ_1, ϵ_2:

$$\frac{Pr(Q|P) = 1-\epsilon_1 \quad Pr(R|P) = 1-\epsilon_2}{Pr(R|P,Q) \geq 1-\epsilon_2/(1-\epsilon_1)} \tag{B.10}$$

The first step is conditioning $Pr(R|P)$ on Q:

$$Pr(R|P) = Pr(R|P,Q)Pr(Q|P) + Pr(R|P,\bar{Q})Pr(\bar{Q}|P).$$

Isolate $Pr(R|P,Q)$, and fill in what you know about $Pr(Q|P)$ and $Pr(R|P)$:

$$\begin{aligned}
Pr(R|P,Q) &= \frac{Pr(R|P) - Pr(R|P\bar{Q})Pr(\bar{Q}|P)}{Pr(Q|P)} \\
&= \frac{Pr(R|P) - Pr(R|P\bar{Q})[1 - Pr(Q|P)]}{Pr(Q|P)} \\
&= \frac{(1-\epsilon_2) - Pr(R|P\bar{Q})[1-(1-\epsilon_1)]}{1-\epsilon_1} \\
&= \frac{(1-\epsilon_2) - \epsilon_1 \cdot Pr(R|P\bar{Q})}{1-\epsilon_1} \\
&\geq \frac{(1-\epsilon_2) - \epsilon_1 \cdot 1}{1-\epsilon_1} \\
&= 1 - \frac{\epsilon_2}{1-\epsilon_1}
\end{aligned}$$

Thus, if $Pr(Q|P) \rightsquigarrow 1$ and $Pr(R|P) \rightsquigarrow 1$ this means that $\epsilon_1 \rightsquigarrow 0$ and $\epsilon_2 \rightsquigarrow 0$. Hence, $\epsilon_2/(1-\epsilon_1) \rightsquigarrow 0$, so that $Pr(R|P,Q) \rightsquigarrow 1$.

3c, p. 156: (Due to Jeroen Snijders.) We prove this rule by showing that for small ϵ_1 and ϵ_2,

$$\frac{Pr(R|P) = 1-\epsilon_1 \quad Pr(R|Q) = 1-\epsilon_2}{Pr(R|P \cup Q) \geq 1-(\epsilon_1+\epsilon_2)} \tag{B.11}$$

To this end, suppose $Pr(R|P) = 1-\epsilon_1$ and $Pr(R|Q) = 1-\epsilon_2$. Hence, $Pr(\bar{R}|P) = \epsilon_1$ and $Pr(\bar{R}|Q) = \epsilon_2$. Accordingly,

$$\begin{aligned}
Pr(\bar{R}|P \cup Q) &= \frac{Pr(\bar{R} \cap (P \cup Q))}{Pr(P \cup Q)} \\
&= \frac{Pr((\bar{R} \cap P) \cup (\bar{R} \cap Q))}{Pr(P \cup Q)} \\
&= \frac{Pr(\bar{R} \cap P) + Pr(\bar{R} \cap Q) - Pr(P \cap Q \cap \bar{R})}{Pr(P \cup Q)} \\
&= \frac{Pr(\bar{R} \cap P)}{Pr(P \cup Q)} + \frac{Pr(\bar{R} \cap Q)}{Pr(P \cup Q)} - \underbrace{\frac{Pr(P \cap Q \cap \bar{R})}{Pr(P \cup Q)}}_{(*)}.
\end{aligned}$$

Since $Pr(P \cup Q) \geq Pr(P)$, $Pr(P \cup Q) \geq Pr(Q)$, and $(*) \geq 0$, we have

$$\begin{aligned} \ldots &\leq \frac{Pr(\bar{R} \cap P)}{Pr(P)} + \frac{Pr(\bar{R} \cap Q)}{Pr(Q)} - 0 \qquad (!) \\ &= Pr(\bar{R}|P) + Pr(\bar{R}|Q) \\ &= \epsilon_1 + \epsilon_2. \end{aligned}$$

Hence,

$$Pr(R|P \cup Q) \geq 1 - (\epsilon_1 + \epsilon_2).$$

Therefore, $1 - (\epsilon_1 + \epsilon_2) \rightsquigarrow 0$ if $(\epsilon_1, \epsilon_2) \rightsquigarrow (0, 0)$. (The latter may be argued with hand-waving.)

3d, p. 156: We prove the rule that says that we may distinguish cases by showing that, for small ϵ_1, ϵ_2:

$$\frac{Pr(R|P,Q) = 1 - \epsilon_1 \quad Pr(R|P,\bar{Q}) = 1 - \epsilon_2}{Pr(R|P) \geq 1 - (\epsilon_1 + \epsilon_2)} \qquad \text{(B.12)}$$

As before, we condition $Pr(R|P)$ on Q:

$$\begin{aligned} Pr(R|P) &= Pr(R|P,Q)Pr(Q|P) + Pr(R|P,\bar{Q})Pr(\bar{Q}|P) \\ &= (1 - \epsilon_1)Pr(Q|P) + (1 - \epsilon_2)Pr(\bar{Q}|P) \\ &= 1 - \epsilon_2 - \epsilon_1 Pr(Q|P) + \epsilon_2 Pr(Q|P) \\ &\geq 1 - \epsilon_2 - \epsilon_1 \cdot 1 + \epsilon_2 Pr(Q|P) \\ &\geq 1 - \epsilon_2 - \epsilon_1 \cdot 1 + \epsilon_2 \cdot 0 \\ &= 1 - (\epsilon_1 + \epsilon_2) \end{aligned}$$

Thus, if $Pr(R|P,Q) \rightsquigarrow 1$ and $Pr(R|P,\bar{Q}) \rightsquigarrow 1$ this means that $\epsilon_1 \rightsquigarrow 0$ and $\epsilon_2 \rightsquigarrow 0$. Hence, $\epsilon_1 + \epsilon_2 \rightsquigarrow 0$, so that $Pr(R|) \rightsquigarrow 1$.

1, p. 166: To obtain a long argument for which none of the rules are justified, we construct an argument

$$p_5 \leftarrow p_4 \leftarrow p_3 \leftarrow p_2 \leftarrow p_1$$

where the steps $p_{i+1} \leftarrow p_i$ unjustified. Thus, we further do not give reasons why, e.g., p_4 implies p_5, or why p_3 implies p_4. A practical (but arbitrary) reading of these steps could be the following: "you should take the bus this morning (p_5), because I need the car (p_4)," "I need the car (p_4), because I must bring the children (p_3)," "I must bring the children (p_3), because you do not seem to have time for it (p_2)", "you do not seem to have time for it (p_2), because you said so (p_1)". To be sure, these steps *could* be justified, be we were asked to give an example of an argument for which none of the rules are justified, and this is such an argument.

2, p. 166: To obtain a situation in which none of the rule antecedents is justified, we leave the statements p_i unjustified. Thus, we further do not give reasons why, e.g., p_1 should be believed, or why p_2 should be believed.

$$r_1: \quad c \leftarrow p_1 \quad r_2: \quad r_1 \leftarrow p_2 \quad r_3: \quad r_2 \leftarrow p_3 \quad r_4: \quad r_3 \leftarrow p_4, p_5$$

Example: "r_1: you should take the bus this morning (c), because I need the car (p_1)" "...because we can't drive together (p_2)" "...because we're having different destinations (p_3)" "...because the children's school is in Maarssen (p_4) and your work is in Woerden (p_5)".

3, p. 166: Example of a realistic argument for which it is plausible that none of the rules is called into question: "$x + 1 > 0$ because $x > 0$". No one would question or challenge this inference. It is still possible to question individual claims like, e.g., "$x > 0$".

4, p. 166: Example: "mr. Jones can be trusted because he is from company X". It is logical to ask questions about why the reliability of mr. Jones follows from the fact that he is paid by company X.

5, p. 166: Philosophical arguments such as: "r_1: I think (p_1) therefore I am (p_2)". "r_2: The experience than I think presents itself to me (p_3), therefore I think". "p_4: Things that are self-presenting involve thinking (r_2)." "p_5: Every object that has intellectual capabilities exists (r_1)." And so on.

6a, p. 167: If $\text{DOB}(P) = \text{DOB}(\neg P) = 1$, there are eight arguments for R:

```
% for R
PROGRAM searches arguments for R ...
... found 8
    which argument? [enter number] 1
R <-(0.75)-
    NOT P
    Q <-(0.75)- NOT P
    which argument? [enter number] 2
R <-(0.75)-
    NOT P
    Q <-(0.75)- P
    which argument? [enter number] 3
R <-(0.75)-
    P
    Q <-(0.75)- NOT P
    which argument? [enter number] 4
R <-(0.75)-
    P
    Q <-(0.75)- P
    which argument? [enter number] 5
R <-(0.75)-
    Q <-(0.75)- NOT P
    which argument? [enter number] 6
R <-(0.75)-
    Q <-(0.75)- P
    which argument? [enter number] 7
R <-(0.75)- NOT P
    which argument? [enter number] 8
R <-(0.75)- P
```

6b, p. 167: There is one argument for R, namely R itself.

1a, p. 169:

- $t \leftarrow\!(0.95)\!\vdash r \leftarrow\!(0.91)\!\vdash p$ [0.84]
  $q \leftarrow\!(0.78)\!\vdash p$ [0.84]

- $t \leftarrow\!(0.95)\!\vdash r \leftarrow\!(0.98)\!\vdash q \leftarrow\!(0.78)\!\vdash p$ [0.84]

- t ←(0.95)– s ←(0.99)– p [0.84]
  q ←(0.78)– p [0.84]

- t ←(0.97)– s ←(0.93)– p [0.84]
  r ←(0.91)– p [0.84]
  q ←(0.78)– p [0.84]

- t ←(0.97)– s ←(0.93)– p [0.84]
  r ←(0.98)– q ←(0.78)– p [0.84]

2, p. 170: An argument such as

t ←(0.95)– r ←(0.91)– p [0.84]
q ←(0.78)– p [0.84]

could in for example Lisp be represented as

```
(setq argument ('t 0.95 ('r 0.91 ('p (0.84) ('q 0.78 'p (0.84)))))
```

Or you can write

```
(setq argument ('t 0.95 ('r 0.91 ('p ('q 0.78 'p)))))
```

and then use property lists to set the DOS of individual propositions:

```
(putprop 'p 0.84 'dos)
```

The same representation can be used in Prolog:

```
argument([t, 0.95, [r, 0.91, [p, [q, 0.78, p]]]]).
dos(p, 0.84).
```

In Perl, you say

```
$argument = ["t", 0.95, ["r", 0.91, ["p", ["q", 0.78, "p"]]]];
$dos{"p"} = 0.84;
```

or

```
$argument = ["rule1", ["rule2", [0.84, "rule3", 0.84]]];
```

The last option assumes that you have named your rules.

3, p. 170: In Lisp this is easy. Say

```
(defun immediate-subargs (arg)
   (cddr arg))
```

and you have them all packed in a list. (Here we utilize the fact that, unlike many programming languages, Lisp can return lists.)

4, p. 170: In Lisp, the following two functions do the work:

```
(defun compute-circ-args-for (formula &optional ancestors)
   (cond
      ((factp formula) (list formula))
      ((null (rules-for formula)) nil)
      ((memberp formula ancestors) (list 'circ))
      (t
         (mapcan #'(lambda (rule)
                     (compute-circ-args-for-rule
                        rule
                        (cons formula ancestors)))
                 (rules-for formula)))))

(defun compute-circ-args-for-rule (rule &optional ancestors)
   (mapcar #'(lambda (antecedent)
               (cons (consequent-of rule)
                  (cons (implication-of rule)
                     antecedent)))
           (list-product-of
              (mapcar #'(lambda (formula)
                          (compute-circ-args-for
                             formula
                             ancestors))
                      (antecedent-of rule)))))
```

where `list-product` returns the Cartesian product of a list of lists. In Prolog, you might try

```
argument(P, P) :-
   dos(P, _).

argument(arg(C, Rule_strength, Subarguments), C) :-
   !, rule(C, Rule_strength, Antecedent),
   mapcar(argument, Subarguments, Antecedent).
```

where it is assumed that arguments are represented as

```
arg(c, 0.89, [
    a,
    arg(b, 0.99, [
        c,
        d,
        arg(e, 0.94, [f, g])
        ])
    ]).
```

1, p. 172: All but the last two.

2, p. 172: Since g and h have no attackers, they remain undefeated. Because g is undefeated, it defeats d and p. Because h is undefeated, it defeats a, e and p. Because both attackers of b (viz. d and e) are now defeated, b remains undefeated. Because q's attacker (viz. p) is defeated, q remains undefeated. Argument i is defeated iff j remains undefeated, and conversely. In the first case, n is defeated by j. In the other case, n is defeated by i. Hence, n is defeated so that f remains undefeated. (A conclusion of f is called a *floating conclusion* [64].) The arguments k, l and m form an odd loop. If k would be undefeated, l is defeated, so that m defeats k, which contradicts our assumption. Similarly, if k would be defeated, his would imply that l defeats m,

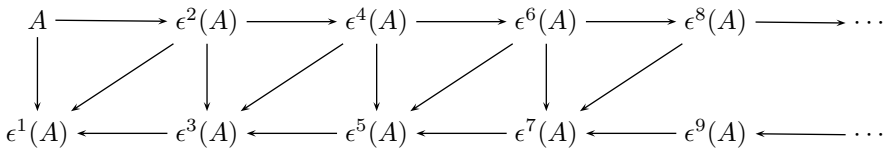
which again produces a contradiction. Hence, the status of k, l, m and hence c can therefore not be determined.

4b, p. 173: Hint: Define an argument to be undefeated if and only if it is in S. Define an argument to be defeated if and only if it is not in S. Then verify conditions (a,b) in Exercise 3 on page 172.

4c, p. 173: Hint: even/odd loop.

6c, p. 173: Result (6c) can be proven with the help of Result (5a). (Cf. Fig. B.14.) The vertical arrows follow from consistency and anti-monotonicity, while the horizontal arrows from self-defence and anti-monotonicity. From the inclusion diagram B.14 it also follows that

$$\limsup \{\epsilon^{2n}(A) \mid n \geq 0\} \subseteq \liminf \{\epsilon^{2n+1}(A) \mid n \geq 0\}. \tag{B.13}$$



(B.14)

8, p. 174: Consider the following six statements:

1. The argument a is not skeptically preferred.
2. The argument a is not contained in all preferred extensions.
3. There is a preferred extension, Q, such that $a \notin Q$.
4. There is a preferred extension, Q, such that every admissible set A that contains a is inconsistent with Q.
5. There is an admissible set, B, such that every admissible set A that contains a is inconsistent with B.
6. There is an admissible set, B, such that every minimally admissible set A that contains a is inconsistent with B.

The equivalences (1) $\Leftrightarrow$ (2) and (2) $\Leftrightarrow$ (3) are left as an exercise.

(3) $\Rightarrow$ (4) If A would be consistent with Q, then $Q \cup A$ would be an admissible set that is larger than Q. Contradiction.

(4) $\Rightarrow$ (3) If $a \in Q$, then Q would be inconsistent with Q. Contradiction.

(4) $\Rightarrow$ (5) A fortiori, since every preferred is admissible.

(5) $\Rightarrow$ (4) Let Q be the largest admissible set such that $B \subseteq Q$. Then Q is preferred. Obviously, every admissible set A that is inconsistent with A is inconsistent with Q as well.

(5) $\Rightarrow$ (6) A fortiori, since every admissible is minimally admissible.

(6) $\Rightarrow$ (5) True, because every admissible set A that contains a must contain a minimally admissible set A that contains a.

1a, p. 177: The difference between $\text{DOS}(r)$, s, and $\text{DOS}(r;q)$ is that $\text{DOS}(r)$ is the support that r receives from other propositions, that s is the strength of r, and $\text{DOS}(r;q)$ is the support that r gives to q. It is very well possible that $\text{DOS}(r) \neq s \neq \text{DOS}(r;q) \neq \text{DOS}(r)$.

1b, p. 177: Any combination of values where s is close to 1.00, and at least one of $\textsc{dos}(a)$, $\textsc{dos}(b)$, $\textsc{dos}(r)$ is close to 0.00.

1c, p. 177: This is impossible, since $\textsc{dos}(r; c) \leq s$.

2a, p. 177: $\textsc{dos}(g) = 0.005$.

2b, p. 177: $\textsc{dos}(g) = 0.005$.

2c, p. 177: $\textsc{dos}(g) = \max\{0.2, 0.005\} = 0.2$.

3a, p. 177: There are two arguments for a, viz.

$$A_1 = a \leftarrow\!(0.89)\!\!- \begin{array}{l} b \\ c \end{array} \qquad \text{and} \qquad A_2 = a \leftarrow\!(0.89)\!\!- \begin{array}{l} b \\ c \leftarrow\!(0.94)\!\!- \ d \end{array}$$

3b, p. 177: Arguments with indexed subconclusions. These indexes indicate the support of the subconclusions.

$$A_1 = a_{0.61} \leftarrow\!(0.89)\!\!- \begin{array}{l} b_{0.97} \\ c_{0.68} \end{array} \qquad \text{and} \qquad A_2 = a_{0.82} \leftarrow\!(0.89)\!\!- \begin{array}{l} b_{0.97} \\ c_{0.92} \leftarrow\!(0.94)\!\!- \ d_{0.98} \end{array}$$

1, p. 179:
$\textsc{gross}(d) = 0.10 \oplus \max\{0.40, 0.50\} = 0.10 \oplus 0.50 = (0.10 + 0.50)/(1 + 0.10 * 0.50) = 0.57$.

3, p. 179:

$$a \oplus b \oplus c \oplus d = \frac{(a + b + c + d) + (abc + abd + acd + bcd)}{1 + (ab + ac + ad + bc + bd + cd) + abcd}$$

4, p. 180: A set of representatives $x_1, \ldots, x_n$ is certainly independent. Hence, it is not heavier than the heaviest independent subset $\{r_1, \ldots, r_h\}$ of R. Hence, $\textsc{gross}'(q) \leq \textsc{gross}(q)$.

Let $R = \{r_1, r_2, r_3\}$ be such that all pairs of rules except r_1 and r_3 are dependent and $\textsc{dos}(r_i) = 0.2i$. The only independent partition of R is $\{R\}$. Hence, there is only one representant, viz. r_3, so that $\textsc{gross}'(q) = 0.9$. A heaviest independent subset of R is $R = \{r_1, r_3\}$. Hence, $\textsc{gross}(q) = 0.3 \oplus 0.9$, which is strictly greater than 0.9.

2a, p. 183: There is one argument for s, viz. $s \leftarrow\!(0.8)\!\!- r \leftarrow\!(0.4)\!\!- p$. There is one argument for $\neg r$, viz. $\neg r \leftarrow\!(1.0)\!\!- q \leftarrow\!(0.5)\!\!- p$. There is one argument for $\neg q$, viz. $\neg q \leftarrow\!(0.6)\!\!- p$.

2b, p. 183: To compute the nett support of s, it is necessary to compute the gross support and nett support of all propositions. (First gross, then nett.)

	p	$\neg p$	q	$\neg q$	r	$\neg r$	s	$\neg s$
gross support	1.0	0.0	0.5	0.6	0.4	0.0	0.32	0.0
nett support	1.0	0.0	0.0	0.6	0.4	0.0	0.32	0.0

The gross support of $\neg r$ is 0.0, because the gross support of $\neg r$ is based on the nett support of q, which is 0.0.

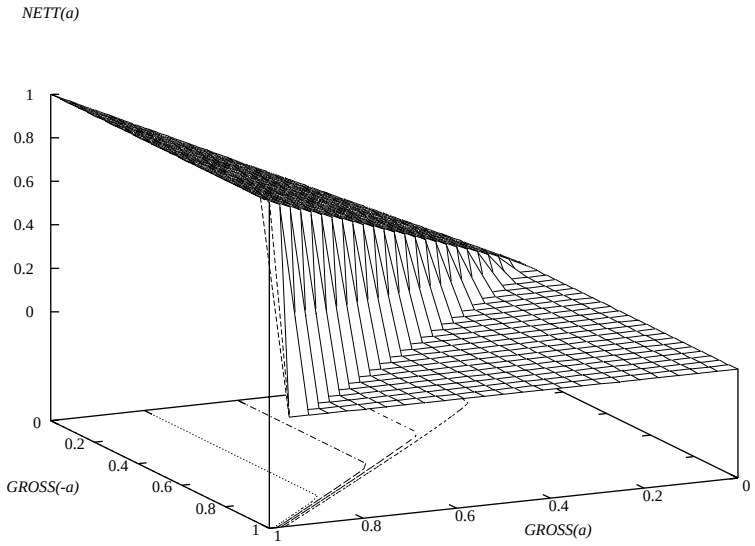
2c, p. 183: This time, support for q is blocked by support for $\neg q$:

	p	$\neg p$	q	$\neg q$	r	$\neg r$	s	$\neg s$
gross support	1.0	0.0	0.7	0.6	0.4	0.7	0.0	0.0
nett support	1.0	0.0	0.7	0.0	0.0	0.7	0.0	0.0

2d, p. 183: Since gross support for q is equal to gross support for $\neg q$, nett support is divided among both propositions:

	p	$\neg p$	q	$\neg q$	r	$\neg r$	s	$\neg s$
gross support	1.0	0.0	0.6	0.6	0.4	0.3	0.32	0.0
nett support	1.0	0.0	0.3	0.3	0.4	0.0	0.32	0.0

2e, p. 183: Either 0.0 or 0.32; no other values. The absence of other, intermediate, values is a typical feature of argumentation-like processes.



3a, p. 183:

Figure B.2: Plot of NETT

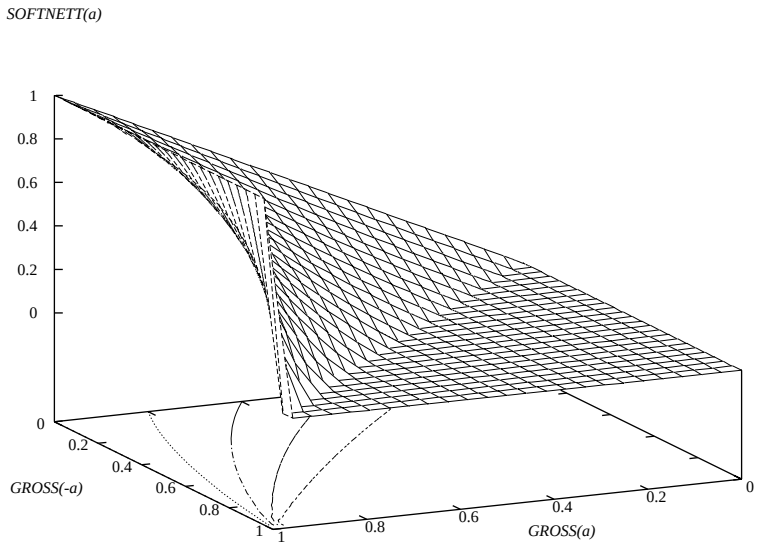
4c, p. 184: The graphs are extremely similar. In particular, the absolute distance between the two graphs is significant only near $(-1, 1)$ or $(1, -1)$. A difference between the two graphs is that f is continuously differentiable (or "smooth"), while g is not, which is probably an indication that that f is a more natural candidate than g.

2a, p. 204:

f	fire	a	alarm goes off	e	evacuation
s	smoke	t	one has tinkered with the alarm		

$$B = \{f \supset (s \wedge a), (f \vee t) \supset a, a \supset e\}$$

3, p. 204: It is actually simple: LPE's are minimal because $H' \subset H$ implies $H \vdash H'$. Here is the full argument. Suppose H is a least presumptive explanation. To show that H is a minimal explanation, suppose that $H' \subset H$ is an explanation as well (it does not even matter how the notion of explanation is actually defined). Because $H' \subset H$ it must be that $H \vdash H'$. Because H is least presumptive and $H \vdash H'$, it must be that $\Pr(D \mid B, H) > \Pr(D \mid B, H')$, which was to be proven.



4b, p. 183:

Figure B.3: Plot of SOFT_NETT

1, p. 212:

proposition	p	$\neg p$	q	$\neg q$	r	$\neg r$	s	$\neg s$
DOB	0.80	0.00	0.00	0.70	0.20	0.00	0.00	0.00
DOS	0.80	0.00	0.00	0.70	0.68	0.00	0.61	0.00

Strongest argument for s:

$$\begin{aligned} s &\leftarrow\!\!(0.90)\!\!- \neg q \\ &\qquad r \leftarrow\!\!(0.85)\!\!- p \end{aligned}$$

2a, p. 212: Quantity $\text{DOS}(p_1)$ is 0.45. Strongest argument: $p_2 \text{–}(0.90)\!\!\rightarrow p_1$. Length of strongest argument: 1. The search stops at p_8: p_8 gives 0.50, needs 0.94, so rules need > 0.94, but all rules emanating from p_8 have strength < 0.94.

2b, p. 212: Strongest argument: since all rules have strength x and all propositions except p have DOB y, and since we have the ability to look ahead in the rule-graph, we know with certainty that the algorithm will not find additional support further down the rules. Thus, for every x and y, $p_2 \text{–}(x)\!\!\rightarrow p_1$ is the strongest argument for p_1. Hence, $\text{DOS}(p_1) = xy$.

Search depth: p_2 needs 0, p_3 needs $\max\{0, y/x\} = y/x$, p_4 needs $\max\{y/x, y/x^2\} = y/x^2$, and so forth. Thus, p_n needs y/x^{n-2}. The search stops if the rule strength, x, becomes smaller than or equal to what is needed. Hence, the search stops when $x \leq y/x^{n-2}$.

$$\begin{aligned} x \leq y/x^{n-2} &\Leftrightarrow x^{n-1} \leq y \\ &\Leftrightarrow \log_x y \leq n-1 \\ &\Leftrightarrow \log_x y + 1 \leq n \\ &\Leftrightarrow \frac{\log y}{\log x} + 1 \leq n \end{aligned}$$

Thus, if $x < 1$, the search stops at node $\lceil (\log y)/(\log x) + 1 \rceil$ if $x < 1$. If $x = 1$, the search goes on forever.

For example, if $(x, y) = (0.90, 0.50)$ then the search stops at node $\lceil (\log 0.5)/(\log 0.9) + 1 \rceil = \lceil 7.57 \rceil = 8$, which corresponds to the above answer.

3, p. 212: Let

proposition	DOB	*proposition*	DOB
p_1	1.00	$p_1 \dashv(0.50)\rightarrow p_2$	$p_2 \dashv(1.00)\rightarrow p_3$
q_1	1.00	$q_1 \dashv(0.60)\rightarrow q_2$	$q_2 \dashv(0.60)\rightarrow \neg p_3$

An argument for p_3 is

$$A = p_3 \leftarrow\!(1.00)\!- p_2 \leftarrow\!(0.50)\!- p_1.$$

An argument against p_3 is

$$B = \neg p_3 \leftarrow\!(0.60)\!- q_2 \leftarrow\!(0.60)\!- q_1$$

According to the weakest link principle, B interferes with A (and A not with B), since $\min\{1.00, 0.60, 0.60\} > \min\{1.00, 0.50, 1.00\}$. According to the certainty-factor propagation principle, A interferes with B (and B not with A), since $1.00 \times 0.60 \times 0.60 < 1.00 \times 0.50 \times 1.00$.

4a, p. 213: Ronald may explain his late arrival with the argument

$$A = \text{too-late} \leftarrow\!(0.90)\!- \text{have-to-wait} \leftarrow\!(0.90)\!- \text{missed-the-bus } [0.75].$$

This argument supports "too-late" with DOS $= 0.75 \times 0.90 \times 0.90 = 0.61$.

4b, p. 213: An argument against A is

$$B = \neg\text{have-to-wait} \leftarrow\!(0.95)\!- \text{missed-the-bus } [0.75]$$
$$\text{high-bus-frequency} \leftarrow\!(0.80)\!- \text{it-is-morning } [1.00]$$

A gives B a DOS of $\min\{1.00 \times 0.80, 0.75\} \times 0.95$.

4c, p. 213: Ronald may explain his late arrival by

$$C = \text{too-late} \leftarrow\!(0.90)\!- \text{have-to-wait} \leftarrow\!(0.90)\!- \text{bridge-open } [0.75].$$

There are arguments against waiting, e.g.,

$$X = \neg\text{have-to-wait} \leftarrow\!(0.95)\!- \text{bridge-open } [0.75]$$
$$\text{there-is-a-detour } [0.00]$$

but all counterarguments are too weak to interfere with C.

4d, p. 213: Now DOB(there-is-a-detour) $= 1.00$, we know that

$$D = \neg\text{have-to-wait} \leftarrow\!(0.95)\!- \text{bridge-open } [0.75]$$
$$\text{there-is-a-detour } [1.00]$$

interferes with C, since D supports "have-to-wait" with DOS $= 0.75 \times 0.95$, and $0.75 \times 0.95 >$ DOS$(C) = 0.75 \times 0.90 \times 0.90$. D has no counterarguments, let alone that there are arguments that interfere with D. Hence, D defeats C.

4e, p. 213: Ronald's explanation

$$E = \text{too-late} \leftarrow\!(0.80)\!- \text{walk} \leftarrow\!(1.00)\!- \text{flat-tire } [0.50]$$

is defeated by

$$F = \text{flat-tire} \leftarrow\!(1.00)\!- \text{missed-the-bus } [0.75].$$

(When you missed the bus, you cannot be late because of a flat tire.)

5a, p. 214: A possible solution in the style of propositional logic is displayed in Table B.1 on the next page. a possible solution in the style of first-order logic is displayed in Table B.2. (Table B.1 could be extended with rules that express that a payment of a particular amount excludes payments of other amounts. This does not have to be specified in the first-order framework, because it follows from the fact that the "pay"-function assumes a unique value.

5b, p. 215: An explanation for a nett profit of \$190,000 in the propositional setting is

$$\begin{array}{l} e = \text{nett190} \leftarrow\!(1.00)\!- \text{cost10} \leftarrow\!(1.00)\!- \text{test } [1.00] \\ \qquad\qquad\qquad\qquad\quad \text{pay200} \leftarrow\!(1.00)\!- \text{soaked } [0.20] \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{drill } [1.00] \end{array}$$

The first-order version looks like

$$\begin{array}{l} e = \text{nett-profit}(x) = 190 \leftarrow\!(1.00)\!- \text{cost}(x) = 10 \leftarrow\!(1.00)\!- \text{test}(x) = \text{yes } [1.00] \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{pay}(x) = 200 \leftarrow\!(1.00)\!- \text{oil}(x) = \text{soaked } [0.20] \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{drill}(x) = \text{yes } [1.00] \end{array}$$

In both cases, "nett-profit$(x) = 190$" receives a DOS of 0.20. The argument e grounds in the assumptions that we have done a seismic test, that we drill, and that the ground is soaked with oil.

Argument e can be defeated, because it grounds in the questionable assumption "oil$(x) =$ soaked". A defeater of e is

$$\begin{array}{l} d = \text{nett-profit}(x) = 40 \leftarrow\!(1.00)\!- \text{cost}(x) = 10 \leftarrow\!(1.00)\!- \text{test}(x) = \text{yes } [1.00] \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{pay}(x) = 50 \leftarrow\!(1.00)\!- \text{oil}(x) = \text{wet } [0.30] \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{drill}(x) = \text{yes } [1.00] \end{array}$$

In both cases, "nett-profit$(x) = 40$" receives a DOS of 0.30. (There are other defeaters of e.)

1, p. 220: There are 100 closed integer-bounded intervals $[a, b]$ such that $a < b$ that start with 0, there are 99 similar intervals that start with 1, ..., there is one closed interval that starts with 99. Thus, $[0, 100]$ possesses $1 + \ldots + 100 = 100 \times 101/2 = 5050$ closed intervals. [There are other ways to count the possible number of intervals, for example by first choosing one point of the interval (101 choices) and then choosing another point of the interval, other than the first point (100 choices). We then obtain 101×100 possible intervals. Since we did not distinguish start and end points, we have counted intervals double, so that we have to divide 101×100 by two.]

In all, $[0, 100]^2$ possesses $5050^2 = 25502500$ different integer-bounded closed rectangles with an interior.

2, p. 220: The most general hypothesis is $H = [0, 100]^2$. It covers all instances.

2a, p. 220: The most general hypothesis is unique, at least in the hypothesis space $\mathcal{H}$ defined in 11.2 on page 219.

A priori data:

Proposition	DOB
dry	0.50
wet	0.30
soaked	0.20

Rules to determine cost:

Proposition	DOB
test –(1.00)→ cost10	1.00
¬test –(1.00)→ cost0	1.00

Rules to predict seismic patterns:

Proposition	DOB
dry, test –(0.10)→ closed	1.00
dry, test –(0.30)→ open	1.00
dry, test –(0.60)→ diffuse	1.00
dry, ¬test –(0.33)→ closed	1.00
dry, ¬test –(0.33)→ open	1.00
dry, ¬test –(0.33)→ diffuse	1.00

Proposition	DOB
wet, test –(0.30)→ closed	1.00
wet, test –(0.40)→ open	1.00
wet, test –(0.30)→ diffuse	1.00
wet, ¬test –(0.33)→ closed	1.00
wet, ¬test –(0.33)→ open	1.00
wet, ¬test –(0.33)→ diffuse	1.00

Rules to predict seismic patterns:

Proposition	DOB
soaked, test –(0.50)→ closed	1.00
soaked, test –(0.40)→ open	1.00
soaked, test –(0.10)→ diffuse	1.00
soaked, ¬test –(0.33)→ closed	1.00
soaked, ¬test –(0.33)→ open	1.00
soaked, ¬test –(0.33)→ diffuse	1.00

Rules to determine revenue:

Proposition	DOB
dry, drill –(1.00)→ pay-70	1.00
dry, ¬drill –(1.00)→ pay0	1.00
wet, drill –(1.00)→ pay50	1.00
wet, ¬drill –(1.00)→ pay0	1.00
soaked, drill –(1.00)→ pay200	1.00
soaked, ¬drill –(1.00)→ pay0	1.00

Rules to exclude disjoint attributes:

Proposition	DOB
closed –(1.00)→ ¬open	1.00
closed –(1.00)→ ¬diffuse	1.00
open –(1.00)→ ¬closed	1.00
open –(1.00)→ ¬diffuse	1.00
diffuse –(1.00)→ ¬open	1.00
diffuse –(1.00)→ ¬closed	1.00

Proposition	DOB
dry –(1.00)→ ¬wet	1.00
dry –(1.00)→ ¬soaked	1.00
wet –(1.00)→ ¬dry	1.00
wet –(1.00)→ ¬soaked	1.00
soaked –(1.00)→ ¬wet	1.00
soaked –(1.00)→ ¬dry	1.00

Rules to determine nett profit:

Proposition	DOB
cost0, pay-70 –(1.00)→ nett-profit-70	1.00
cost0, pay0 –(1.00)→ nett-profit0	1.00
cost0, pay50 –(1.00)→ nett-profit50	1.00
cost0, pay200 –(1.00)→ nett-profit200	1.00

Proposition	DOB
cost10, pay-70 –(1.00)→ nett-profit-80	1.00
cost10, pay0 –(1.00)→ nett-profit-10	1.00
cost10, pay50 –(1.00)→ nett-profit40	1.00
cost10, pay200 –(1.00)→ nett-profit190	1.00

Table B.1: Defeasible rules for the oil wildcatters example (propositional version).

A priori data:

Sentence	DOB
$\text{oil}(x) = \text{dry}$	0.50
$\text{oil}(x) = \text{wet}$	0.30
$\text{oil}(x) = \text{soaked}$	0.20

Rules to predict seismic patterns:

Sentence	DOB
$\text{oil}(x) = \text{dry},\ \text{test}(x) = \text{yes} \text{–}(0.10)\!\!\rightarrow \text{seismic}(x) = \text{closed}$	1.00
$\text{oil}(x) = \text{dry},\ \text{test}(x) = \text{yes} \text{–}(0.30)\!\!\rightarrow \text{seismic}(x) = \text{open}$	1.00
$\text{oil}(x) = \text{dry},\ \text{test}(x) = \text{yes} \text{–}(0.60)\!\!\rightarrow \text{seismic}(x) = \text{diffuse}$	1.00
$\text{oil}(x) = \text{dry},\ \text{test}(x) = \text{no} \text{–}(0.33)\!\!\rightarrow \text{seismic}(x) = \text{closed}$	1.00
$\text{oil}(x) = \text{dry},\ \text{test}(x) = \text{no} \text{–}(0.33)\!\!\rightarrow \text{seismic}(x) = \text{open}$	1.00
$\text{oil}(x) = \text{dry},\ \text{test}(x) = \text{no} \text{–}(0.33)\!\!\rightarrow \text{seismic}(x) = \text{diffuse}$	1.00
$\text{oil}(x) = \text{wet},\ \text{test}(x) = \text{yes} \text{–}(0.30)\!\!\rightarrow \text{seismic}(x) = \text{closed}$	1.00
$\text{oil}(x) = \text{wet},\ \text{test}(x) = \text{yes} \text{–}(0.40)\!\!\rightarrow \text{seismic}(x) = \text{open}$	1.00
$\text{oil}(x) = \text{wet},\ \text{test}(x) = \text{yes} \text{–}(0.30)\!\!\rightarrow \text{seismic}(x) = \text{diffuse}$	1.00
$\text{oil}(x) = \text{wet},\ \text{test}(x) = \text{no} \text{–}(0.33)\!\!\rightarrow \text{seismic}(x) = \text{closed}$	1.00
$\text{oil}(x) = \text{wet},\ \text{test}(x) = \text{no} \text{–}(0.33)\!\!\rightarrow \text{seismic}(x) = \text{open}$	1.00
$\text{oil}(x) = \text{wet},\ \text{test}(x) = \text{no} \text{–}(0.33)\!\!\rightarrow \text{seismic}(x) = \text{diffuse}$	1.00
$\text{oil}(x) = \text{soaked},\ \text{test}(x) = \text{yes} \text{–}(0.50)\!\!\rightarrow \text{seismic}(x) = \text{closed}$	1.00
$\text{oil}(x) = \text{soaked},\ \text{test}(x) = \text{yes} \text{–}(0.40)\!\!\rightarrow \text{seismic}(x) = \text{open}$	1.00
$\text{oil}(x) = \text{soaked},\ \text{test}(x) = \text{yes} \text{–}(0.10)\!\!\rightarrow \text{seismic}(x) = \text{diffuse}$	1.00
$\text{oil}(x) = \text{soaked},\ \text{test}(x) = \text{no} \text{–}(0.33)\!\!\rightarrow \text{seismic}(x) = \text{closed}$	1.00
$\text{oil}(x) = \text{soaked},\ \text{test}(x) = \text{no} \text{–}(0.33)\!\!\rightarrow \text{seismic}(x) = \text{open}$	1.00
$\text{oil}(x) = \text{soaked},\ \text{test}(x) = \text{no} \text{–}(0.33)\!\!\rightarrow \text{seismic}(x) = \text{diffuse}$	1.00

Rules to determine cost:

Sentence	DOB
$\text{test}(x) = \text{yes} \text{–}(1.00)\!\!\rightarrow \text{cost}(x) = 10$	1.00
$\text{test}(x) = \text{no} \text{–}(1.00)\!\!\rightarrow \text{cost}(x) = 0$	1.00

Rules to determine revenue:

Sentence	DOB
$\text{oil}(x) = \text{dry},\ \text{drill}(x) = \text{yes} \text{–}(1.00)\!\!\rightarrow \text{pay}(x) = -70$	1.00
$\text{oil}(x) = \text{dry},\ \text{drill}(x) = \text{no} \text{–}(1.00)\!\!\rightarrow \text{pay}(x) = 0$	1.00
$\text{oil}(x) = \text{wet},\ \text{drill}(x) = \text{yes} \text{–}(1.00)\!\!\rightarrow \text{pay}(x) = 50$	1.00
$\text{oil}(x) = \text{wet},\ \text{drill}(x) = \text{no} \text{–}(1.00)\!\!\rightarrow \text{pay}(x) = 0$	1.00
$\text{oil}(x) = \text{soaked},\ \text{drill}(x) = \text{yes} \text{–}(1.00)\!\!\rightarrow \text{pay}(x) = 200$	1.00
$\text{oil}(x) = \text{soaked},\ \text{drill}(x) = \text{no} \text{–}(1.00)\!\!\rightarrow \text{pay}(x) = 0$	1.00

Rule to determine nett profit:

Sentence	DOB
$\text{cost}(x) = \text{u},\ \text{pay}(x) = \text{v} \text{–}(1.00)\!\!\rightarrow \text{nett-profit}(x) = \text{v}-\text{u}$	1.00

Table B.2: Defeasible rules for the oil wildcatters example (first-order version).

2b, p. 220: 10 instances.

2c, p. 220: Only positive instances should be covered. Thus, four instances are classified correctly.

2d, p. 220: Number of instances classified correctly divided by the total number of instances = 4/10.

2e, p. 221: Number of positive instances covered divided by the total number of instances covered = 4/10.

3, p. 221: An (arbitrary) example of a most specific hypothesis is $H = [85, 86] \times [11, 12]$. There are more most specific hypotheses; in fact there are as many such hypotheses as there are unit squares on the $[0, 100]^2$-grid, namely 100×100. Let us proceed with H. This hypothesis classifies all 10 instances as negative, hence six correctly so. Further, H covers 0 instances, so that $match(H) = 6/10$ and $accuracy(H) = 0/0 =$ `undef`.

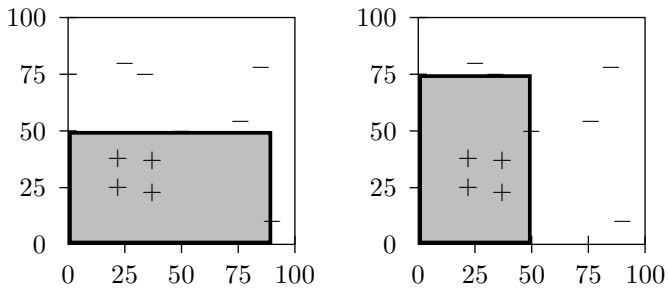
4, p. 221: $H = [20, 60]^2$.

Instance	Class	H's classification
$(22, 25)$	+	+
$(76, 54)$	-	-
$(37, 23)$	+	+
$(37, 37)$	+	+
$(25, 80)$	-	-
$(34, 75)$	-	-
$(85, 78)$	-	-
$(22, 38)$	+	+
$(90, 10)$	-	-
$(50, 50)$	-	+

H classifies 10 instances of which 9 are classified correctly. Thus, $match(H) = 9/10$. Further, H covers 5 instances, of which 4 are covered correctly. Thus, $accuracy(H) = 4/5$.

5, p. 221: $H = [20, 40]^2$ does the job. It covers all positive instances, and no negative instance. $match(H) = 10/10 = accuracy(H) = 4/4 = 1$. Of course, other solutions are possible.

6, p. 221: There are two most general hypothesis that are consistent with the data, viz. $H_1 = [0, 90] \times [0, 50]$ and $H_2 = [0, 50] \times [0, 75]$. Thus, a most general hypothesis that are consistent with the data is not unique.



7, p. 221: $H = [22, 37] \times [25, 38]$ is *the* most specific hypothesis. It is included in every other hypothesis that is consistent with (11.1).

1a, p. 223: An antecedent can be formed by any (possibly empty) combination of the remaining 49 literals. This number is equal to the number of subsets of a set of 49 literals, namely, $2^{49} \approx 10^{29}$. If antecedents are ordered, there are even more possibilities, namely $49 + 49 \cdot 48 + 49 \cdot 48 \cdot 47 + \ldots$.

1b, p. 223: In the first level there are 49 choices. At each node in the second level there are 48 choices, hence, $49 \cdot 48$ choices at the second level. Therefore, the entire search tree contains $49! \approx 10^6 4 > 10^2 9$ nodes. If beam search is applied, it will be clear that most nodes will not be visited.

1a, p. 231: If $a = 1$, $b = 0$ and $c = 0$,

$| \rightarrow x$, score: $9/15 = 0.60$
- $| a \rightarrow x$, score: $8/12 = 0.67$
  - $| a, b \rightarrow x$, score: $8/10 = 0.80$
    - $| a, b, c_1 \rightarrow x$, score: $1/1 = 1.00$ $\surd$
    - $| a, b, c_2 \rightarrow x$, score: $1/1 = 1.00$ $\surd$
    - $| \ldots$
    - $| a, b, c_8 \rightarrow x$, score: $1/1 = 1.00$ $\surd$
    - $| a, b, c_9 \rightarrow x$, score: $0/1 = 0.00$
    - $| a, b, c_{10} \rightarrow x$, score: $0/1 = 0.00$
    - $| a, b, c_{11} \rightarrow x$, score: $1/1 = 1.00$
    - $| a, b, c_{12} \rightarrow x$, score: $0/1 = 0.00$
    - $| a, b, c_{13} \rightarrow x$, score: $0/1 = 0.00$
    - $| a, b, c_{14} \rightarrow x$, score: $0/1 = 0.00$
    - $| a, b, c_{15} \rightarrow x$, score: $0/1 = 0.00$
- $| b \rightarrow x$, score: $9/12 = 0.75$
  - $| b, a \rightarrow x$, score: $8/10 = 0.80$
    - specializations as with $a, b \rightarrow x$
    - $\ldots$
- $| \neg a \rightarrow x$, score: $1/3 = 0.33$
  - $| \neg a, b \rightarrow x$, score: $1/2 = 0.50$
    - $| \neg a, b, c_{11} \rightarrow x$, score: $1/1 = 1.00$ $\surd$
  - $| \neg a, \neg b \rightarrow x$, score: $0/1 = 0.00$
- $| \neg b \rightarrow x$, score: $0/3 = 0.00$
- $| c_1 \rightarrow x$, score: $1/1 = 1.00$ $\surd$
- $| c_2 \rightarrow x$, score: $1/1 = 1.00$ $\surd$
- $| \ldots$
- $| c_8 \rightarrow x$, score: $1/1 = 1.00$ $\surd$
- $| c_9 \rightarrow x$, score: $0/1 = 0.00$
- $| c_{10} \rightarrow x$, score: $0/1 = 0.00$
- $| c_{11} \rightarrow x$, score: $1/1 = 1.00$ $\surd$
- $| c_{12} \rightarrow x$, score: $0/1 = 0.00$
- $| c_{13} \rightarrow x$, score: $0/1 = 0.00$
- $| c_{14} \rightarrow x$, score: $0/1 = 0.00$
- $| c_{15} \rightarrow x$, score: $0/1 = 0.00$

Thus, with $a = 1$, $b = 0$ and $c = 0$, rules are specialized into the extreme. Since the maximum score is 1.00, and several rules score 1.00, the best rule is among $a, b, c_1 \rightarrow x$, $a, b, c_2 \rightarrow x$, $a, b, c_8 \rightarrow x$, $\neg a, b, c_{11} \rightarrow x$, $c_1 \rightarrow x$, $c_2 \rightarrow x$, $\ldots$, $c_8 \rightarrow x$, and $c_{11} \rightarrow x$.

1b, p. 231: If $a = 1$, $b = 0$ and $c = 0.023$,

$| \rightarrow x$, score: $9/15 + 0.023 \times 15 = 0.945$

| $a \to x$, score: $8/12 + 0.023 \times 12 = 0.946$
  | $a, b \to x$, score: $8/10 + 0.023 \times 10 = 1.03$ $\surd$
    | $a, b, c_1 \to x$, score: $1/1 + 0.023 \times 1 = 1.02$ stop: $\downarrow$
    | $a, b, c_2 \to x$, score: $1/1 + 0.023 \times 1 = 1.02$ stop: $\downarrow$
    | ...
    | $a, b, c_8 \to x$, score: $1/1 + 0.023 \times 1 = 1.02$ stop: $\downarrow$
    | $a, b, c_9 \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ stop: $\downarrow$
    | $a, b, c_{10} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ stop: $\downarrow$
    | $a, b, c_{11} \to x$, score: $1/1 + 0.023 \times 1 = 1.02$ stop: $\downarrow$
    | $a, b, c_{12} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ stop: $\downarrow$
    | $a, b, c_{13} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ stop: $\downarrow$
    | $a, b, c_{14} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ stop: $\downarrow$
    | $a, b, c_{15} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ stop: $\downarrow$
| $b \to x$, score: $9/12 + 0.023 \times 12 = 1.026$
  | $b, a \to x$, score: $8/10 + 0.023 \times 10 = 1.03$ $\surd$
    specializes further as with $a, b \to x$
    ...
| $\neg a \to x$, score: $1/3 + 0.023 \times 3 = 0.399$
  | $\neg a, b \to x$, score: $1/2 + 0.023 \times 2 = 0.546$
    | $\neg a, b, c_{11} \to x$, score: $1/1 + 0.023\times = 1.02$ $\surd$
  | $\neg a, \neg b \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ stop: $\downarrow$
| $\neg b \to x$, score: $0/3 + 0.023 \times 3 = 0.069$ $\surd$ (all specializations decrease the score)
| $c_1 \to x$, score: $1/1 + 0.023 \times 1 = 1.02$ $\surd$ (*id.*)
| $c_2 \to x$, score: $1/1 + 0.023 \times 1 = 1.02$ $\surd$ (*id.*)
| ...
| $c_8 \to x$, score: $1/1 + 0.023 \times 1 = 1.02$ $\surd$ (*id.*)
| $c_9 \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ $\surd$ (*id.*)
| $c_{10} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ $\surd$ (*id.*)
| $c_{11} \to x$, score: $1/1 + 0.023 \times 1 = 1.02$ $\surd$ (*id.*)
| $c_{12} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ $\surd$ (*id.*)
| $c_{13} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ $\surd$ (*id.*)
| $c_{14} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ $\surd$ (*id.*)
| $c_{15} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ $\surd$

"Stop: $\downarrow$" means that the algorithm stops with specializing the rule, because the score decreases. This time, the maximum score is 1.03, which is for $a, b \to x$ and $b, a \to x$. If order in the antecedent does not matter, there is precisely one best rule, viz. $a, b \to x$.

1c, p. 231: To reach an optimum score at $a, b \to x$ and $b, a \to x$, we reason as follows. To ensure that $\to x$, $a \to x$ and $b \to x$ are specialized into $a, b \to x$ and $b, a \to x$, the score of $\to x$, $a \to x$ and $b \to x$ must be less than the score of $a, b \to x$ and $b, a \to x$:

$$9/15 + 15c < 8/12 + 12c,$$
$$8/12 + 12c < 8/10 + 10c, \text{ and}$$
$$9/12 + 12c < 8/10 + 10c,$$

which is equivalent to

$$c < 0.0233, \; c < 0.0433, \text{ and } c < 0.0250. \tag{B.15}$$

To ensure that $a, b \to x$ and $b, a \to x$ need not be specialized further, the score of all specializations of $a, b \to x$ and $b, a \to x$ must be less than the score of $a, b \to x$ and $b, a \to x$. This amounts to

$$8/10 + 10c > 1 + c,$$

which is equivalent to

$$c > 0.0222. \tag{B.16}$$

Combining (B.15) and (B.16) yields $0.222 < c < 0.233$. If $c \geq 0.233$ the search will stop too early; if $c \leq 0.222$, the search will overshoot its goal.

2a, p. 236: The network tries to express that lions are more similar to tigers than, e.g, lions to fishes. Fishes are dissimilar to mammals (-1). Pigs are neither similar nor dissimilar to lions and tigers. I thought that makes sense...

2b, p. 236: The node values express a selection of a coherent subset of "lion," "tiger," "fish," "pig," with indifference as regarding to whether "pig" should be included in this set.

2c, p. 236: Local coherence between "lion" and "fish" is $1 \times -1 \times -1 = 1$.

2d, p. 236: Global coherence is $0 + 0 + 0 + 1 + 1 + 1 = 3$.

2e, p. 236: Change node value of fish from -1 into 1. Local coherency between tiger and lion decreases with 2 each, so that global coherency decreases with 4 to -1. (Other solutions are possible.)

2f, p. 236: If we are searching for an optimal solution, we are allowed to change only the node values. Trying out different values (!) yields that the global coherency can be increased to 4 if "pig" is set to 1.

2g, p. 236: Yes: a perfect solution is an assignment of activation values for which every (in)coherence relation between two propositions is satisfied. There are four non-zero links, so a perfect solution would have all 4 such links satisfied. The previous item shows that this is possible.

3, p. 236: If $P \sim Q \sim R \nsim P$, for instance, then accepting P implies accepting Q, and accepting Q implies accepting R. Accepting R, however, implies rejecting P. A similar impossible situation arises if we assume P rejected. Thus, the highest coherence that can be obtained for this network is $1 + 1 + (-1) = 1$. (The more simple combination $P \sim Q \nsim P$ does not qualify as an example of a non-perfect network because it isn't a coherence network.)

4, p. 236: Suppose nodes can take on values from $\{-1, 0, 1\}$, and links $\{-1, 0, 1\}$. (Links with value 0 can be considered as not to exist.) The first node can take on three values, the connecting link can take on three values, and the second node can take on three values. This makes for $3^3 = 27$ combinations. Due to symmetry, there are actually $3^2 \cdot 2 = 18$ properly different combinations, since the value of the second node must be chosen such that it is unequal to the value of the first node.

2a, p. 240: One solution is.

```
C -> D
B contradicts D
A -> B
A,B -> C
observation A
A -> B ~ C -> D
```

Many other solutions are possible.

1, p. 245:

$$a \oplus b \oplus c \oplus d = \frac{(a+b+c+d)+(abc+abd+acd+bcd)}{1+(ab+ac+ad+bc+bd+cd)+abcd}$$

Bibliography

[1] E.W. Adams. The logic of 'almost all'. *Journal of Philosophical Logic*, 3:3–17, 1974.

[2] E.W. Adams. On the logic of high probability. *Journal of Philosophical Logic*, 15:255–279, 1986.

[3] E.W. Adams and H.P. Levine. On the uncertainties transmitted from premises to conclusions in deductive inferences. *Synthese*, 30:429–460, 1975.

[4] S.K. Andersen et al. Hugin: a shell for building bayesian belief universes for expert systems. In *Proc. of the 11th Int. Joint Conf. on Artificial Intelligence*, pages 1080–1085, Los Altos, CA, 1989.

[5] R. Audi. *Epistemology: A Contemporary Introduction to the Theory of Knowledge.* Routledge, London, 1998.

[6] M. Aydede. Language of thought: The connectionist contribution. *Minds and Machines*, 7(1):57–101, 1997.

[7] J. Barwise and J. Etchemendy. *The Language of First-Order Logic.* Cambridge UP, 1990.

[8] M. Böhm and E. Speckenmeyer. A fast parallel SAT-solver — efficient workload balancing. *Annals of Mathematics and Artificial Intelligence*, 17:381– 400, 1996.

[9] I. Bratko. *Prolog Programming for Artificial Intelligence.* Int. Computer Science Series. Addison-Wesley, 3rd edition, 2000.

[10] G. Brewka. Preferred subtheories: An extended logical framework for default reasoning. In *Proc. of the 11th Int. Joint Conf. on Artificial Intelligence*, pages 1043–1048, Detroit, Michigan, August 1989.

[11] B. Buchanan and E.H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison Wesley, Reading, Massachusetts, 1984.

[12] Marco Cadoli and Andrea Schaerf. Compiling problem specifications into SAT. *Lecture Notes in Computer Science*, 2028:387–402, 2001.

[13] C. Chang and R.C. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, Boston, 1973.

[14] P. Cheeseman. Discussion of a paper by Lauritzen and Spiegelhalter. In *Journal of the Royal Statistic Society, B*, pages 50–203, 1988.

[15] V. Chvátal. On the computational complexity of finding a kernel. Technical Report CRM-300, Centre de Recherches de Mathématiques, Université de Montréal, 1973.

[16] W.F. Clocksin and C.S. Mellish. *Programming In Prolog.* Springer-Verlag, Berlin, 1994.

[17] I.M. Copi and C. Cohen. *Introduction to Logic*. Macmillan, New York, 1953.

[18] James M. Crawford and Andrew B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proc. of the Twelfth National Conf. on Artificial Intelligence (AAAI-94)*, volume 2, pages 1092–1097, Seattle, Washington, USA, 1994. AAAI Press/MIT Press.

[19] M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors. *Handbook of Tableau Methods*. Kluwer, Dordrecht, 1999.

[20] M. D'Agostino and M. Mondadori. The taming of the cut: Classical refutations with analytic cut. *Journal of Logic and Computation*, 4:285–319, 1994.

[21] Marcello D'Agostino. Are tableaux an improvement on truth-tables? cut-free proofs and bivalence, 1992.

[22] Marcello D'Agostino et al. Winke: A pedagogical tool for teaching logic and reasoning. *Intelligent Tutoring Systems*, page 605, 1998.

[23] M. Davis, G. Longemann, and D. Loveland. A machine program for theorem proving. *Comm. of the ACM*, 5(7):394–397, 1962.

[24] M. Devitt. *Coming to our Senses: A Naturalistic Program for Semantic Localism*. Cambridge University Press, 1996.

[25] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming, and n-person games. *Artificial Intelligence*, 77(2):321–357, 1995.

[26] N. Everitt and A. Fisher. *Modern Epistemology: A New Introduction*. McGraw-Hill, 1995.

[27] M. Fitting. *First-Order Logic and Automated Theorem Proving*. Graduate Texts in Computer Science. Springer-Verlag, Berlin, 2nd edition, 1996.

[28] John Fox. Three arguments for extending the framework of probability. In Laveen N. Kanal and John F. Lemmer, editors, *UAI '85: Proc. of the First Annual Conf. on Uncertainty in Artificial Intelligence*. Elsevier, 1988.

[29] J.H. Gallier. *Logic for Computer Science*. Harper Row, New York, 1986.

[30] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman and Company, New York, 1983.

[31] I. Gent and T. Walsh. The search for satisfaction, 1999.

[32] C.J. Gerhardt. *Die Philosopphisen Schriften Von G.W. Leibniz*. Bnd. VII. Hildesheim, 1978.

[33] M.L. Ginsberg, editor. *Readings in Nonmonotonic Reasoning*, Los Altos, CA, 1987. Morgan Kaufmann.

[34] M.X. Goemans and D.P Williamson. .878-approximation algorithms for MAXCUT and MAX2SAT. In *Proc. Of the 26th Ann.L ACM STOC*, pages 422–431, Montreal, 1994.

[35] A.J. Gonzalez and D.D. Dankel. *The Engineering of Knowledge-based Systems: Theory and Practice*. Prentice Hall, Englewoods Cliffs, New Jersey, 1993.

[36] T.F. Gordon, N. Karaçapilidis, H. Voss, and A. Zauke. Computer-mediated cooperative spatial planning. In H. Timmermans, editor, *Decision Support Systems in Urban Planning*, pages 299–309. E & FN SPON Publishers, London, 1997.

[37] Thomas F. Gordon. *The Pleadings Game: An Artificial Intelligence Model of Procedural Justice*. Kluwer, Dordrecht, 1995.

[38] J.F. Groote. The propositional formula checker HeerHugo. Technical Report ???, CWI, Amsterdam, 1996.

[39] J.F. Groote and J. Warners. The propositional formula checker HeerHugo. Technical report, CWI, 1999.

[40] I. Hacking. *The emergence of probability*. Cambridge U.P., London, 1975.

[41] R. Hähnle et al. The even more liberalized delta-rule in free variable semantic tableaux. In G. Gottlob, editor, *Computational logic and proof theory*, volume 713 of *LNCS*, pages 108–119. Springer-Verlag, 1993.

[42] R. Hähnle et al. The liberalized delta-rule in free variable semantic tableaux. *Journal of Automated Reasoning*, 13(2):211–221, 1994.

[43] J. Herbrand. *Logical Writings*. Harvard University Press, Cambridge, MA, 1971. Translation of *Écrits Logiques*, Jean van Heijenoort (Ed.), Presses Univrsitaires de France, Paris.

[44] C. M. Hoadley, M. Ranney, and P. Schank. WanderECHO: A connectionist simulation of limited coherence. In A. Ran and K. Eiselt, editors, *Proc. Of the 16th Ann. Conf. Of the Cognitive Science Society*, pages 421–426, Hillsdale, NJ, 1994. Erlbaum.

[45] Thomas Hoos, Holger H.; Stützle. SATLIB: An online resource for research on SAT. In Ian Gent, Hans Van Maaren, and Toby Walsh, editors, *SAT 2000*. IOS Press, 2000.

[46] J.F. Horty and R.H. Thomason. Mixing strict and defeasible inheritance. *Proc. of the AAAI*, pages 427–432, 1988.

[47] R.E. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

[48] J. Kalman. *Automated Reasoning with Otter*. Rinton Press Inc., New Jersey, 2001.

[49] G.J. Klir. Where do we stand on measures of uncertainty, ambiguity, fuzziness, and the like? *Fuzzy Sets and Systems*, pages 141–160, 24.

[50] William Kneale and Martha Kneale. *The Development of Logic*. Oxford UP, Oxford, 1962.

[51] S. Kraus, K. Sycara, and A. Evenchik. Reaching agreements through argumentation: a logical model and implementation. *Artificial Intelligence*, 104:1–69, 1998.

[52] S.L. Lauritzen and D.J. Spiegelhalter. Local computations with probabilities on graphical structures and their applications to expert systems. *Journal of the Royal Statistical Society, B*, 50:157–224, 1988.

[53] V. Lifschitz. Benchmark problems for formal nonmonotonic reasoning, version 2.00. In M. Reinfrank et al., editor, *Proc. of the Second Workshop in Non-monotonic Reasoning*, Lecture Notes in Artificial Intelligence, pages 202–219. Springer Verlag, Grassau, FRG, 1988.

[54] Chao-Lin Liu and Michael Wellman. Incremental tradeoff resolution in qualitative probabilistic networks. In *Proc. of the Fourteenth Annual Conf. on Uncertainty in Artificial Intelligence (UAI–98)*, pages 338–345, San Francisco, CA, 1998. Morgan Kaufmann Publishers.

[55] Ronald P. Loui. Benchmark problems for nonmonotonic systems. Handout, 1989. Distributed at the Workshop on Defeasible Reasoning with Specificity and Multiple Inheritance.

[56] Ronald P. Loui. Process and policy: Resource-bounded nondemonstrative reasoning. *Computational Intelligence*, 14(1):1–38, 1998.

[57] D. W. Loveland. *Automated Theorem Proving: A Logical Basis*. North-Holland, Amsterdam, etc. 1978.

[58] D.W. Loveland. *Automated Theorem Proving: A Quarter-Century Review*, volume 29 of *Contemporary Mathematics*, pages 1–45. American Mathemtical Society, 1984.

[59] P.J.F. Lucas. Symbolic diagnosis and its formalisation. *The Knowledge Engineering Review*, 12(2):109–146, 1997.

[60] E.L. Lusk. Controlling redundancy in large search spaces: Argonne-style theorem proving through the years. In *Proc. Of the LPAR*, pages 96–106, Berlin, 1992. Springer-Verlag. Springer LNAI.

[61] D. MacKenzie. The automation of proof, a historical and sociological exploration. *IEEE Annals of the History of Computing*, 17:7–29, 1995.

[62] D. MacKenzie. The automation of proof: a historical and sociological exploration. *Annals of the History of Computing*, 17(3):7–29, 1995.

[63] A.L. Madsen and F.V. Jensen. Lazy propagation: a junction tree inference algorithm based on lazy evaluation. *Artificial Intelligence*, 113:203–245, 1999.

[64] D. Makinson and K. Schlechta. Floating conclusions and zombie paths: two deep difficulties in the 'directly skeptical' approach to inheritance nets. *Artificial Intelligence*, 48:199–209, 1991.

[65] Witold Marciszewski and Roman Murawski. *Mechanization of Reasoning in a Historical Perspective*. Poznań Studies in the Philosophy of the Sciences and the Humantities. Rodopi, 1995.

[66] W.W. McCune. *Otter 3.0 Reference Manual and Guide*. Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, revision a, august 1995 edition, 1994.

[67] E. Mendelson. *Introduction to Mathematical Logic*. Van Nostrand Company, New York, 1964.

[68] E. Milgram. Harman's hardness arguments. *Pacific Philosophical Quarterly*, 72:181–202, 1991.

[69] W. Molesworth. *The English Works of Thomas Hobbes of Malmesbury, now first collected and edited by Sir William Molesworth, 11 vols., 1839-45*. Scienta Aalen, 1962.

[70] A. Newell, J.C. Shaw, and H.A. Simon. Empirical explorations with the logic theory machine: A case study in heuristics. Reprinted in Computers and Thought, ed. Feigenbaum and Feldman, McGraw-Hill, NY, 1963, pp. 109-133; also reprinted in Automation of Reasoning - Classical Papers on Computational Logic, Vol. I, 1957-1966, ed. J. Siekmann and G. Wrightson, Springer-Verlag, Berlin, 1983, pp. 49-73., 1957.

[71] A. Newell and H. Simon. *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, 1972.

[72] S. Parsons. A proof theoretic approach to qualitative probabilistic reasoning. *Int. Journal of Approximate Reasoning*, 19:265–297, 1998.

[73] S. Parsons et al. A review of uncertainty handling formalisms. In S. Parsons et al., editors, *Applications of Uncertainty Formalisms*, pages 8–37. Springer-Verlag, 1998.

[74] S. Parsons, C. Sierra, and N.R. Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8:261–292, 1998.

[75] J. Pearl. System z: A natural ordering of defaults with tractable aplications to nonmonotonic reasoning. In R. Parikh, editor, *Proc. of the Third Conf. on Theoretical Aspects of Reasoning About Knowledge.*, pages 121–140, Pacific Grove, CA, March 1990. Morgan Kaufmann Publishers.

[76] J. Pearl. Epsilon-semantics. In S.C. Shapiro et al., editors, *Encyclopedia of Artificial Intelligence*, chapter Epsilon-semantics, pages 468–475. John Wiley and Sons, New York, 1992.

[77] J. Pearl. *Causality : Models, Reasoning, and Inference*. Cambridge UP, 2000.

[78] J. Pearl and H. Geffner. Probabilistic semantics for a subset of default reasoning. Technical Report Technical report CSD-870052 R-93-III, Cognitive Systems Laboratory, CS Dept., University of California, Los Angeles, CA, 1988.

[79] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, Inc., Palo Alto CA, 2 edition, 1994.

[80] J. Pelletier. Completely non-clausal, completely heuristically driven, automatic theorem proving. Master's thesis, Dept. of Computing Science, University of Alberta, Edmonton, Alberta, 1982. Tech. Report TR82-7.

[81] J. Pelletier. Seventy-five problems for testing automatic theorem provers. *Journal of Automated Reasoning*, 2:191–216, 1986.

[82] Fernando C. N. Pereira and David Warren. Definite clause grammars for language analysis — a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231–278, 1980.

[83] G.W. Pieper and R. Veroff, editors. *Automated Reasoning and its Applications : Essays in Honor of Larry Wos*. MIT Press, 1997.

[84] J. Pitt and J. Cunningham. Theorem proving and model building with the calculus ke. *Journal of the IGPL*, 4(1):129–150, 1996.

[85] John L. Pollock. *Cognitive Carpentry. A Blueprint for How to Build a Person*. MIT Press, Cambridge, MA, 1995.

[86] K.R. Popper. *The Logic of Scientific Discovery*. Hutchinson, London, 1934.

[87] K.R. Popper. What is dialectic? *Mind*, 49, 1940.

[88] E.L. Post. Introduction to a general theory of elementary propositions. *Amer. J. Math.*, 43:163–185, 1921.

[89] H. Prakken. On formalising burden of proof in legal argument. In *Proc. of the Twelfth Int. Conf. on Legal Knowledge-Based Systems (JURIX'99)*, pages 85–97, Leuven, Belgium, December 9-10 1999.

[90] H. Prakken and S. Renooij. Reconstructing causal reasoning about evidence: a case study. In Ronald P. Loui Bart Verheij, Arno R. Lodder and Antoinette J. Muntjwerff, editors, *Legal Knowledge and Information Systems. Jurix2001: The 14th Annual Conf.*, pages 131–137. Amsterdam: IOS Press, 2001.

[91] H. Prakken and G. Sartor. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-classical Logics*, 7:25–75, 1997.

[92] Henry Prakken and Gerard A.W. Vreeswijk. Logics for defeasible argumentation. In D.M. Gabbay et al., editors, *Handbook of Philosophical Logic*, pages 219–318. Kluwer Academic Publishers, Dordrecht, 2002.

[93] H. Putnam. *Representation and Reality*. MIT Press, Cambridge, Massachusetts, 1988.

[94] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[95] S. Renooij and L.C. Van der Gaag. Decision making in qualitative inference diagrams. In *Proc. Of the Ninth Dutch Conf. on Artificial Intelligence (NAIC)*, pages 93–102, 1997.

[96] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–49, 1965.

[97] Uwe Schöning. *Logic for Computer Scientists*. Progress in Computer Science and Applied Logic. Birkhäuser, Boston, etc., 1989.

[98] R. Sebastiani. Applying GSAT to non-clausal formulas (research note). *Journal of Artificial Intelligence Research*, 1:309–314, 1994.

[99] B. Selman, H.A. Kautz, and D.A. McAllester. Ten challenges in propositional reasoning and search. In *Proc. of the Fifteenth Int. Joint Conf. on Artificial Intelligence (IJCAI'97)*, pages 50–54, 1997.

[100] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proc. of theTenth National Conf. on Artificial Intelligence (AAAI-92)*, pages 440–446, San Jose, CA, 1992.

[101] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proc. of the Twelfth National Conf. on Artificial Intelligence (AAAI'94)*, pages 337–343, Seattle, 1994.

[102] J. H. Siekmann and G. Wrightson, editors. *The Automation of Reasoning*, volume I an II. Springer-Verlag, Berlin, 1983.

[103] L. Siklóssy, A. Rich, and V. Marinov. Breadth-first search: Some surprising results. *Artificial Intelligence*, 4:1–27, 1973.

[104] M. Smithson. *Ignorance and Uncertainty: Emerging Paradigms*. Springer-Verlag, New York, 1989.

[105] P. Smolensky. Constituent structure and explanation in an integrated Connectionist/Symbolic cognitive architecture. In C. Macdonald and G. Macdonald, editors, *Connectionism: Debates on Psychological Explanation*. Basil Blackwell, Oxford, UK, 1995.

[106] R.M. Smullyan. *First-Order Logic*. Springer-Verlag, 1968. Revised edition issued by Dover in 1994.

[107] E. Sosa. Two conceptions of knowledge. *The Journal of Philosophy*, 67:59–66, 1970.

[108] Ernest Sosa. The raft and the pyramid: Coherence versus foundations in the theory of knowledge. In Linda Martín Alcoff, editor, *Epistemology: The Big Questions*, pages 187–210. Blackwell,, Oxford, 1998.

[109] M. Stefik. *Introduction to Knowledge Systems*. Morgan Kaufmann, San Francisco, CA, 1995.

[110] K. Sterelny. *The Representational Theory of Mind*. MIT Press, Cambridge, Massachusetts, 1990.

[111] G. Sutcliffe et al. Progress in automated theorem proving, 1997-1999. In *Workshop on Empirical Methods in Artificial Intelligence, 17th Int. Joint Conf. on Artificial Intelligence*, pages 53–60, 2001.

[112] Paul Thagard. Explanatory coherence. *Behavioral and Brain Sciences*, 12:435–467, 1989.

[113] Paul Thagard. *Conceptual Revolutions*. Princeton University Press, Princeton, 1992. Paperback edition, 1993. Italian translation published by Guerini e Associati, 1994.

[114] Paul Thagard. *Coherence in Thought and Action*. MIT Press, Cambridge, MA, 2000.

[115] Paul Thagard and E.Millgram. Inference to the best plan: A coherence theory of decision. In A. Ram and D.B. Leake, editors, *Goal-Driven Learning*, pages 439–454. MIT Press, Cambridge, MA, 1995.

[116] Paul Thagard et al. Knowledge and coherence. In R. Elio, editor, *Common Sense, Reasoning, and Rationality*. Oxford UP, 1997.

[117] David Thomas and Andrew Hunt. *Programming Ruby: The Pragmatic Programmer's Guide*. Addison-Wesley, 2000. Aka "the Pickaxe book" (because the cover shows a pickaxe mining rubies). Available online at http://www.rubycentral.com/book/.

[118] Stephen Toulmin. *The Uses of Argument*. Cambridge University Press, 1985.

[119] M.A. Trick. *Second DIMACS Challenge Test Problems*, volume 26 of *DIMACS Series in Discrete Mathematics and Compter Science*, pages 653–657. American Mathematical Society, 1996.

[120] K. Verbeurgt and Paul Thagard. Coherence as constraint satisfaction. *Cognitive Science*, 22:1–24, 1998.

[121] H. Bart Verheij. *Rules, Reasons, Arguments: Formal studies of argumentation and defeat*. Department of Metajuridica, University of Limburg, Maastricht., Maastricht, The Netherlands, 1996. dissertation.

[122] H. Bart Verheij. Automated argument assistance for lawyers. In *Proc. of the Seventh Int. Conf. on Artificial Intelligence and Law*, pages 43–52, New York, 1999. ACM.

[123] Gerard A. W. Vreeswijk. Argumentation in bayesian belief networks. In Iyad Rahwan, Pavlos Moraitis, and Chris Reed, editors, *Argumentation in Multi-Agent Systems, First International Workshop, ArgMAS 2004*, volume 3366 of *Lecture Notes in Computer Science*, pages 111–129, 2005.

[124] Gerard A.W. Vreeswijk. IACAS: An implementation of Chisholm's principles of knowledge. In C. Witteveen, W. van der Hoek, J.-J. Ch. Meyer, and B. van Linder, editors, *The Proc. of the 2nd Dutch/German Workshop on Nonmonotonic Reasoning*, pages 225–234, Utrecht, 1995. Delft University of Technology, University of Utrecht.

[125] Gerard A.W. Vreeswijk. Interpolation of benchmark problems in defeasible reasoning. In M. DeGlas, editor, *Proc. of the Second World Conf. on the Fundamentals of Artificial Intelligence*, pages 453–468, Paris, 1995. Angkor, Paris. Formerly presented at the Second Symposium on AI and Mathematics, 1992, Fort Lauderdale, Florida.

[126] J. Warners. *Nonlinear Approaches to Satisfiability Problems*. PhD thesis, Techincal University Eindhoven, Eindhoven, The Netherlands, 1999.

[127] M.P. Wellman. Fundamental concepts of qualitative probabilistic networks. *Artificial Intelligence*, 44:257–303, 1990.

[128] A. Whitehead and B. Russell. *Principia Mathematica*, volume Vol. I. Cambridge UP, Cambridge, 1910.

[129] L. Wos. *Automated Reasoning: Thirty-Three Basic Research Problems*. Prentice Hall, 1988.

[130] L. Wos et al. *Automated Reasoning: Introduction and Applications*. Prentice-Hall, 1984.

[131] S.V. Yablonsky. Functional constructions in the k-valued logic. *Reports of the Steklov's Math. Institute, Russian Academy of Science*, 51:5–142, 1958. (In Russian.).

Index