Exam-1 Program Verification 2019/2020

BBG-169, 7th Oct 2019, 8:30 - 10:30

Lecturers: Wishnu Prasetya, Anna-Lena Lamprecht

1 Predicate Transformer [2pt]

1. [0.6 pt] Give the weakest pre-conditions of the following statements. **Show** your calculation.

   (a) $\{* \; ? \; *\}$ **if** $\mathtt{x} \geq \mathtt{10}$ **then** $\{\mathtt{x} := \mathtt{x} - \mathtt{10} \; ; \; \mathtt{y} := \mathtt{y} * \mathtt{x}\}$ **else** $\mathtt{y} := -\mathtt{1} \;\; \{* \; \mathtt{y} > \mathtt{0} \; *\}$

   Solution: (not showing the calculation)

   $(x \geq 10 \;\wedge\; y(x-10) > 0) \vee (x < 10 \;\wedge -1 > 0)$

   The 2nd conjunct is not satisfiable so we can drop it; so simplifying the pre-cond to just:

   $(x \geq 10 \;\wedge\; y(x-10) > 0)$

   (b) $\{* \; ? \; *\} \;\; a[i{-}1] := a[0] + 1 \; ; \; a[0] := 0 \;\; \{* \; \mathtt{a}[\mathtt{i}{-}\mathtt{1}] \geq \mathtt{a}[\mathtt{0}] \; *\}$

   Do reduce/remove all **repby** sub-expressions from your result.

   Solution: the wlp before the 2nd assignment is:

   $(i{-}1{=}0 \;\rightarrow\; 0 \mid a[i{-}1]) \geq 0$

   The wlp before the 1st assignment is then:

   $(i{-}1{=}0 \;\rightarrow\; 0 \mid a[0]{+}1) \geq 0$

   Because $0 \geq 0$, the left arm of the if-then-else is always true. So we can simplify above to:

   $i{-}1{\neq}0 \;\Rightarrow\; a[0]{+}1 \geq 0$

2. [0.3 pt] Suppose we have a programming language that has a concept of statements, for which we can define Hoare triples: if S is a statement of this programming language, $\{P\}\; S \;\{Q\}$ means that if we execute S on a state satisfying the pre-condition P, and if the statement terminates, it will terminate in a state satisfying Q.

   Suppose we define how to calculate **wlp** over the statements of this programming language, and claim that this **wlp** is complete. What does this mean?

3. [0.5 pt] The calculation of **wlp** over $a[e_1] := e_2$ as defined in the PV Lecture Notes assumes that the array a is of infinite size. Suppose we now want to reason over realistic arrays that must have a finite size,

and moreover a should not be null. Propose how to calculate the **wlp** of such an assignment when a is a realistic array.

You can assume that e_1 and e_2 do not crash/throw any exception.

Solution: in this solution we will not allow exception to be thrown when an array is indexed outside its valid range.

$$\textbf{wlp}\ (a[e_1] := e_2)\ Q\ =\ (Q[a(e_1\ \textbf{repby}\ e_2)/a]) \wedge a \neq \mathsf{null} \wedge 0 \leq e_1 < \#a$$

where $\#a$ denotes the length of the array a.

4. [0.5 pt] Suppose we want to have N-dimensional arrays, N>0. E.g. we can now write $a[0][1][0]$. Give a rule to calculate the **wlp** over an assignmentr $target := expr$ where the $target$ is a 3-dimensional array expression. For example, an assignment such as $a[0][1][0] := 0$.

   You can assume that arrays have infinite size.

   Solution:

   In general, for N-dimensional array:

   $$\textbf{wlp}\ (a[e_0]...[e_{k-1}] := f)\ Q\ =\ Q[\alpha/a]$$

   where α_i is defined recursively as follows:

   $$\begin{array}{lcll} \alpha_0 & = & a(e_0\ \textbf{repby}\ \alpha_1) & \\ \alpha_i & = & a[e_0]...a[e_{i-1}](e_i\ \textbf{repby}\ \alpha_i) & , \text{for } 0 < i < k \\ \alpha_k & = & f & \end{array}$$

   So, for 3-dimensional array, α would be:

   $$a(e_0\ \textbf{repby}\ a[e_0](e_1\ \textbf{repby}\ a[e_0][e_1](e_2\ \textbf{repby}\ f)))$$

   Another way to define it is by introducing a new 'repby', let's call it $\textbf{repby}_\textbf{3}$. We define α to be $a(e_0, e_1, e_2\ \textbf{repby}_3\ f)$ where $\textbf{repby}_3$ is defined/characterized as:

   $$a(e_0, e_1, e_2\ \textbf{repby}_3\ f)[i][j][k]\ =\ (i{=}e_0 \wedge j{=}e_1 \wedge k{=}e_2\ \rightarrow f \mid a[i][j][k])$$

2 Dealing with Loop [2 pt]

1. [1.8 pt] Consider the specifications of the programs below. Give for each a loop-invariant that would be good enough to prove the validity of the corresponding specification. It should be an invariant that is consistent, strong enough to realize the asked post-condition, and realistic to be established by the pre-condition or initialization of the loop. Use the *partial* correctness interpretation of Hoare triples.

   Unless stated otherwise, all variables are of type int.

(a) $\{* \texttt{x}\in\{1,2,3\} *\}$

while $\texttt{x}>0$ **do** { **if** $\texttt{x}=1$ **then** $\texttt{x} := \texttt{x}+1$ **else** $\texttt{x} := \texttt{x}-2$ }

$\{* \texttt{x} = 0 *\}$

Solution: $x \geq 0$ will do.

(b) $\{* \texttt{x}=10 \wedge \texttt{y}=0 *\}$

while $\texttt{x}>0$ **do** $\{\texttt{x} := \texttt{x}-2 \; ; \; \texttt{y} := \texttt{y}+20 \}$

$\{* \texttt{x}+\texttt{y} = 100 *\}$

Solution: $x \geq 0 \wedge 10x+y = 100 \wedge even(x)$

(c) Below, the variable `a` is an array of int, and `sorted` is a boolean. The program checks if the array segment $\texttt{a}[0..\texttt{N}]$ is sorted.

$\{* \texttt{k}=1 \;\; \wedge \texttt{N}>0 \;\wedge\; \texttt{sorted}=\textbf{true} *\}$

while $\texttt{k}<\texttt{N} \wedge \texttt{sorted}$ **do** { $\texttt{sorted} := (\texttt{a}[\texttt{k}-1] \leq \texttt{a}[\texttt{k}])$; $\texttt{k}:=\texttt{k}+1$ }

$\{* \texttt{sorted} \;=\; (\forall \texttt{i} : 1 \leq \texttt{i} < \texttt{N} : \texttt{a}[\texttt{i}-1] \leq \texttt{a}[\texttt{i}]) *\}$

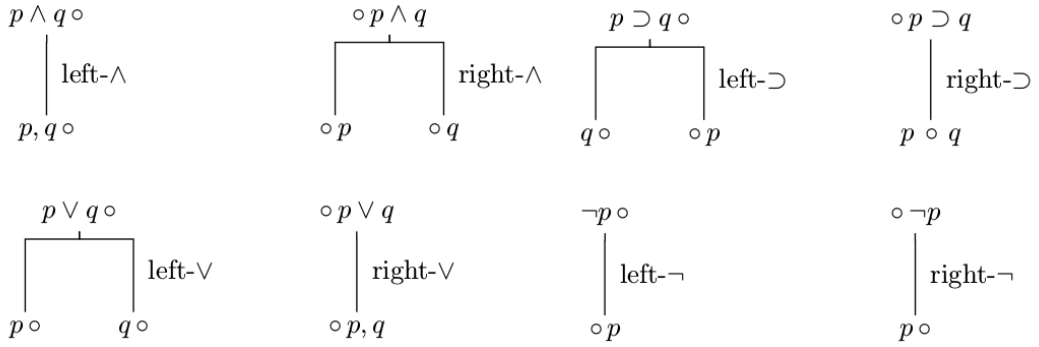
2. [challenging 0.2pt] Suppose this specification is valid under partial correctness:

$$\{* \texttt{I} *\} \;\; \textbf{while } g \textbf{ do } S \;\; \{* \texttt{Q} *\}$$

Furthermore I is also an invariant of the loop above. Prove (it does not have to be a formal proof) that $(g \wedge I) \vee (\neg g \wedge Q)$ is also invariant.

3 Propositional Theorem Proving [1 pt]

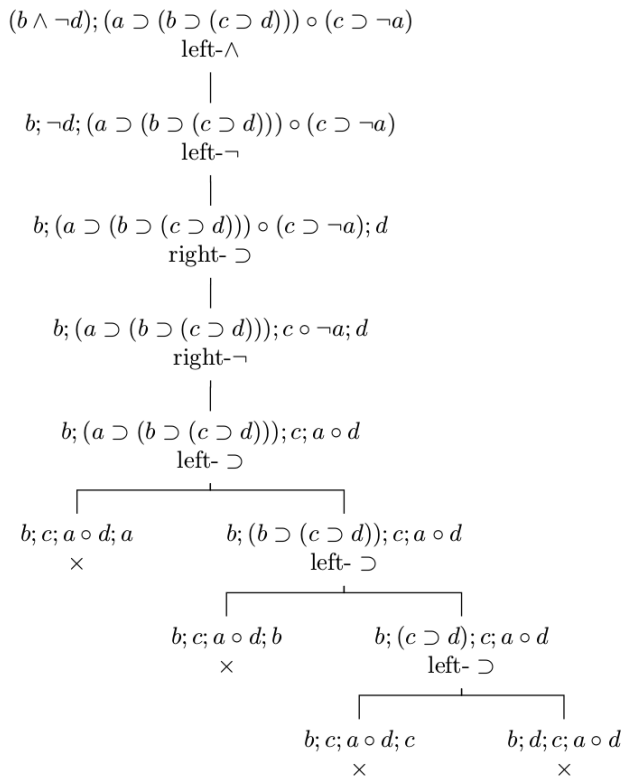
[1 pt] Construct refutation trees for the sequents below. Specify a counterexample if the sequent turns out to be invalid. Use the standard analytic refutation rules from the lecture:



1. $k \supset (e \wedge o)\ ;\ o \supset (b \wedge t) \quad \vdash \quad \neg k \wedge t$
2. $b \wedge \neg d\ ;\ a \supset (b \supset (c \supset d)) \quad \vdash \quad c \supset \neg a$

Solutions:

1. Invalid sequent. Countermodel: w, with $w(t) = w(k) = w(o) = 0$.
2. Valid sequent. Derivation:



4 First-Order Theorem Proving [2.5 pt]

1. [0.5 pt] Construct a refutation tree for the following sequent:

$$(\exists x)((p \supset qx)) \quad \vdash \quad (p \supset (\exists y)(qy))$$

Specify a counterexample if the sequent turns out to be invalid. Use the standard analytic refutation rules (see above) and the additional rules for first-order logic:

$(\forall x)\phi \circ$
$\phi[t/x] \circ$
left-$\forall$
Term t free for x in ϕ

$\circ (\forall x)\phi$
$\circ \phi[y/x]$
right-$\forall$
y does not occur in the current tree

$(\exists x)\phi \circ$
$\phi[y/x] \circ$
left-$\exists$
y does not occur in the current tree

$\circ (\exists x)\phi$
$\circ \phi[t/x]$
right-$\exists$
Term t free for x in ϕ

2. [1 pt] The sequent $\vdash (\forall x)(\exists y)(\forall z)(\exists w)(rx, y \vee \neg rw, z)$ is valid. After a number of steps, we can close the refutation tree as follows (because $rv_2, \varsigma_2(v_1)$ is unifiable with $r\varsigma_1, v_3$):

$$\begin{array}{c} \vdots \\ | \\ rv_2, \varsigma_2(v_1); rv_4, \varsigma_3(v_3) \circ \phi; \psi; r\varsigma_1, v_1; r\varsigma_1, v_3 \\ \times \text{ with } \sigma = \{\varsigma_1/v_1, \varsigma_1/v_2, \varsigma_2(\varsigma_1)/v_3\} \end{array}$$

   Can it also be closed with another substitution? (explain)

3. [1 pt] Does the unification algorithm always halt? Explain your answer. (Hint: Consider the total number of variables in t_1 and t_2.)

Solutions:

1. Valid. Here is a failed refutation (= proof):

$$\begin{array}{c} (\exists x)((p \supset qx)) \circ (p \supset (\exists y)(qy)) \\ \text{right-}\supset \\ | \\ (\exists x)((p \supset qx)); p \circ (\exists y)(qy) \\ \text{right}^{+}\text{-}\exists \\ | \\ (\exists x)((p \supset qx)); p \circ (\exists y)(qy); qc_0 \\ \text{left-}\exists \\ | \\ (p \supset qc_1); p \circ (\exists y)(qy); qc_0 \\ \text{right}^{+}\text{-}\exists \\ | \\ (p \supset qc_1); p \circ (\exists y)(qy); qc_1; qc_0 \\ \text{left-}\supset \end{array}$$

$$\begin{array}{cc} \mathbf{p} \circ (\exists y)(qy); qc_1; qc_0; \mathbf{p} & \mathbf{qc_1}; p \circ (\exists y)(qy); \mathbf{qc_1}; qc_0 \\ \times & \times \end{array}$$

2. Yes. The formula $rv_4, \varsigma_3(v_3)$ can be unified with the formula $r\varsigma_1, v_1$ by means of the substitution $\tau = \{\varsigma_3(\varsigma_1)/v_1, \varsigma_1/v_3, \varsigma_1/v_4\}$.

3. Yes, it always halts. The total number of variables is finite. The algorithm chooses a pair of terms in each iteration. It will either halt the procedure (because not reasonable substitution is possible), or substitute (eliminate) one of the variables to make the terms equal,

possibly up until the point where the input terms are unified. Thus, if it never halts within the iteration, the loop condition will eventually be false and the algorithm will thus terminate.

5 Clause Sets and Resolution [2.5 pt]

1. [0.5 pt] Describe a polynomial-time algorithm for determining wether a CNF is a tautology.

2. [0.5 pt] Use (binary) resolution to prove that the following proposition is unsatisfiable. Convert to clause sets first.

$$(r \vee \neg u) \wedge (\neg r \vee s) \wedge (u \vee s) \wedge (\neg s \vee v) \wedge (\neg v \vee \neg s)$$

3. [1.5 pt] 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?

Solutions:

1. A CNF is a tautology iff all terms possess complementary literals. So, we will check for each term (linear in the number of terms) wether it contains at least one pair of complementary literals (will be at most quadratic in the number of literals in the term). So this whole procedure is in $O(n^3)$ even in a naive version.

2. One possible solution:

$$\begin{aligned}
\neg\phi &\equiv \{\{r, \neg u\}, \{\neg r, s\}, \{u, s\}, \{\neg s, v\}, \{\neg v, \neg s\}\} && \{r, \neg u\}, \{\neg r, s\} \rightsquigarrow \{\neg u, s\} \\
&\equiv \{\{r, \neg u\}, \{\neg r, s\}, \{u, s\}, \{\neg s, v\}, \{\neg v, \neg s\}, \{\neg u, s\}\} && \{u, s\}, \{\neg u, s\} \rightsquigarrow \{s\} \\
&\equiv \{\{r, \neg u\}, \{\neg r, s\}, \{u, s\}, \{\neg s, v\}, \{\neg v, \neg s\}, \{\neg u, s\}, \{s\}\} && \{\neg s, v\}, \{\neg v, \neg s\} \rightsquigarrow \{\neg s\} \\
&\equiv \{\{r, \neg u\}, \{\neg r, s\}, \{u, s\}, \{\neg s, v\}, \{\neg v, \neg s\}, \{\neg u, s\}, \{s\}, \{\neg s\}\} && \{s\}, \{\neg s\} \rightsquigarrow \square \\
&\equiv \{\{r, \neg u\}, \{\neg r, s\}, \{u, s\}, \{\neg s, v\}, \{\neg v, \neg s\}, \{\neg u, s\}, \{s\}, \{\neg s\}, \square\}
\end{aligned}$$

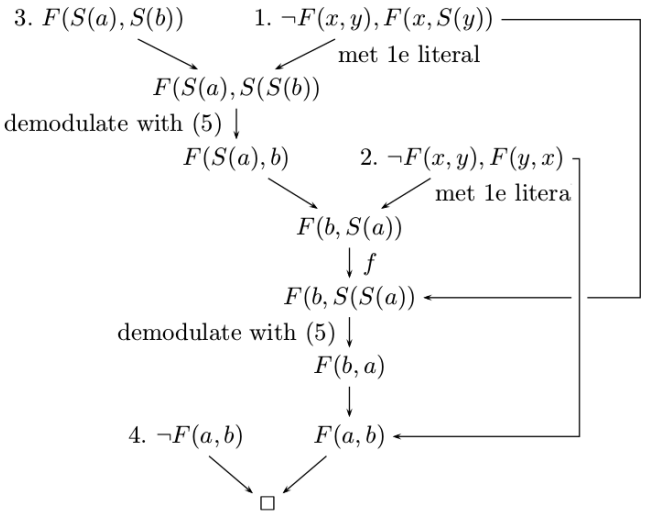
3. We translate the above natural-language expressions into the language of first-order logic as follows. For friendship we take the two-place predicate letter $F/2$, and for spouse we take the one-place predicate letter $S/1$. Alice and Betty are the constants a and b, respectively.

   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.