Logic for Computer Science

12 – Games

Paige Randall North

(based on Wouter Swierstra's slides)

University of Utrecht

Welcome back

- This week we will continue with the last chapters of *Modelling Computing Systems*.
- From Thursday onwards, we will be using a separate set of lecture notes for the last two weeks or so. You can find these linked on the course homepage.
- In case you slacked off after the mid-term: please look at the chapters on induction! This is a very important topic that plays an important role in the final exam (and the rest of your degree).
- You should have received your marks for the midterm.

Last time

Proofs by induction

This lecture

Games

Modelling Computing Systems

The book is called Modelling Computing Systems – but so far we have studied:

- sets
- functions
- relations
- propositional and predicate logic
- induction

… but these are all very *static* concepts. There's hardly any *interaction*.

Games

In the lecture today, we will start study how to apply these ideas to model a structured series of *interactions*.

Rather than try to describe interactions between, for example, two computers connected over the internet, we'll start by studying a more restricted form of interaction, namely **games**.

Games

We can distinguish between different kinds of games:

- Games of chance – like roulette – where there is no good strategy for winning.
- Games of no chance – like tic-tac-toe or chess – where some moves are clearly better than others.

These are two extremes: many fun games have some element of luck, making them unpredictable, while leaving room for clever strategies.

Simplifying assumptions

We'll study the *games of no chance* – although many ideas from this lecture can be extended to games of (some) chance, but this requires probability theory.

We will study games between *two players*.

We assume that in these games both players have *perfect information* – each player knows everything there is to know about the game at any given point:

- chess is a game of perfect information
- poker is not – as players only know the cards in their *own* hand.

We will only study *finite* games that are guaranteed to end after some finite number of moves.

Example: 21

Two players count, starting from one. They take turns in saying one, two or three numbers. The player that says “twenty-one” wins.

Alice: One, two.

Bob: Three.

Alice: Four, five, six.

Bob: Seven, eight.

...

Example: 21

Two players count, starting from one. They take turns in saying one, two or three numbers. The player that says “twenty-one” wins.

Alice: One, two.

Bob: Three.

Alice: Four, five, six.

Bob: Seven, eight.

...

Anyone fancy a game?

How to win at 21?

How can player A always beat player B?

- If it's player A's turn and the current count is 18, 19, or 20, they have won. They simply count up to and including 21.
- If it's player A's turn, and the current count is 17 – they are in a *losing position*. Regardless of player A's move, their opponent can always reach 21—player A *cannot* win.
- By the same argument, 13 is also a losing position—your opponent can always reach 17.
- So 9, 5 and 1 are also losing positions.

So the first player has a *winning strategy:* always finish on a number that is 1 modulo 4.

How to win at 21?

How can player A always beat player B?

- If it's player A's turn and the current count is 18, 19, or 20, they have won. They simply count up to and including 21.
- If it's player A's turn, and the current count is 17 – they are in a *losing position*. Regardless of player A's move, their opponent can always reach 21—player A *cannot* win.
- By the same argument, 13 is also a losing position—your opponent can always reach 17.
- So 9, 5 and 1 are also losing positions.

So the first player has a *winning strategy:* always finish on a number that is 1 modulo 4.

Of course, we can generalize this game allowing for a number of steps or a different target number. (How does this change the winning strategy?)

Strategies

A *strategy* is a rule that computes the best move a player can make at any given point in the game.

A *winning strategy* is one that guarantees the player will win the game.

A *drawing strategy* is one that guarantees the player will win the game or draw (but not lose).

A position in a game is called a *winning position* if the current player has a winning strategy (such as 19 in the game Twenty-one).

If the other player has a winning strategy in the current position, this is known as a *losing position* (such as 17 in the game Twenty-one).

10 coins

There are 10 coins on a table. Players take turns by removing two or three coins. The player that takes the last coin wins; if one coin remains, the game has ended in a draw.

Which player has a winning strategy?

10 coins

There are 10 coins on a table. Players take turns by removing two or three coins. The player that takes the last coin wins; if one coin remains, the game has ended in a draw.

Which player has a winning strategy?

Like we did for twenty-one, we can work our way backwards through the game.

10 coins

If it is your turn and:

- there are no coins left, you are in a *losing position*;
- there is one coin left, you are in a *drawing position*;
- there are two coins left, you are in a *winning position*;
- there are three coins left, you are in a *winning position*;

What about four coins?

10 coins

If it is your turn and:

- there are no coins left, you are in a *losing position*;
- there is one coin left, you are in a *drawing position*;
- there are two coins left, you are in a *winning position*;
- there are three coins left, you are in a *winning position*;

What about four coins?

We have two options:

- if we take two coins, there are two coins left, leaving our opponent in a winning position;
- if we take three coins, there is one coin left, leaving our opponent in a drawing position.

Clearly the second choice is preferable, hence four coins is also a *drawing position*.

Beyond four coins

Question

What about five coins?

Beyond four coins

Question

What about five coins?

Whatever move we make, we leave our opponent in a winning position – hence five coins is a losing position.

Question

What about five coins?

Whatever move we make, we leave our opponent in a winning position – hence five coins is a losing position.

We can tabulate the different kinds of positions for each number of coins:

Coins left	0	1	2	3	4	5	6	7	8	9	10
Position	L	D	W	W	D	L	D	W	W	D	L

We can also visualize this game by drawing a *game tree* describing the possible moves and associated positions for each possible game state.

Game trees

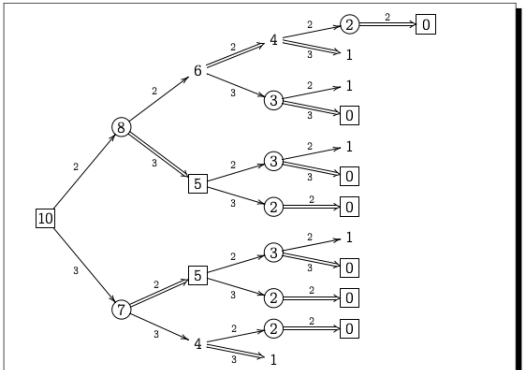


Figure 10.1: The game tree for a simple coin game.

Each node contains a number, corresponding to the current game state; each node has two subtrees, corresponding to the state after removing 2 or 3 coins.

Game trees

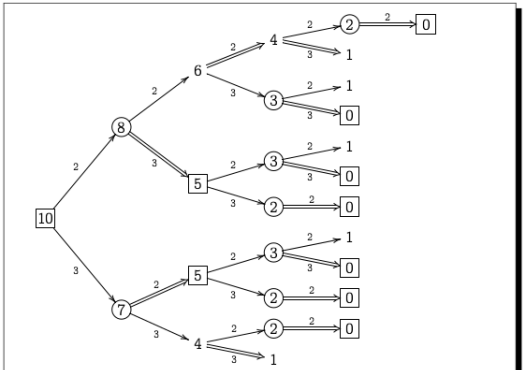


Figure 10.1: The game tree for a simple coin game.

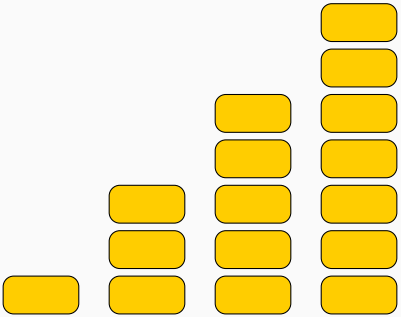
Furthermore, each node has been decorated with a circle if it is a *winning position* and square if it is a *losing position*; the double arrows indicate the best move to make at any given position.

Nim

Nim

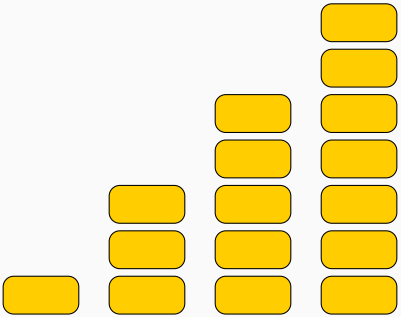
The game of Nim is a classic game of no chance that has a non-obvious winning strategy.

In a game of Nim, two players face a finite number of piles of coins – typically 1, 3, 5 and 7:



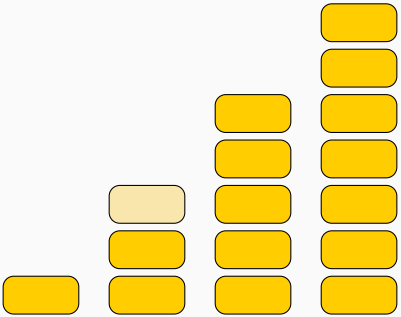
The players take turns in removing an arbitrary (nonzero) number of coins from *one* pile. The player that takes the last coin wins.

Nim



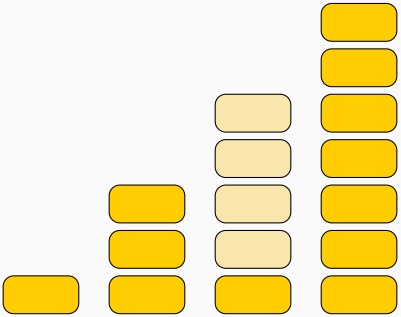
Starting from this position – the first player can take various different moves.

Nim



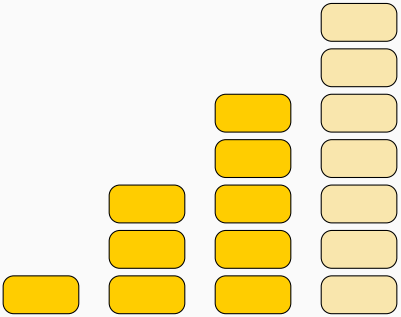
They could remove one coin from the second stack

Nim



Or remove some coins from the third stack.

Nim



Or remove all the coins from the last stack.

Nim

What move is best? What is a *winning strategy* for Nim?

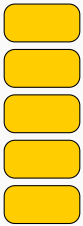
Nim

What move is best? What is a *winning strategy* for Nim?

It isn't at all obvious when one move is better than another.

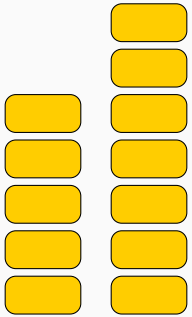
To find a winning strategy, it can help by studying degenerate instances of this game.

Two-stack Nim



If there is just one stack of coins, the first player has an obvious winning strategy – take all the coins and win.

Two-stack Nim

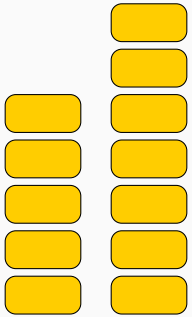


What about two stacks?

Question

Which player has a winning strategy? (Hint: try mirroring your opponent's moves)

Two-stack Nim



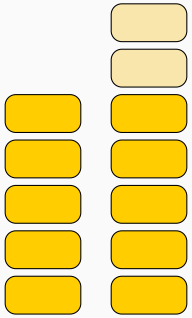
What about two stacks?

Question

Which player has a winning strategy? (Hint: try mirroring your opponent's moves)

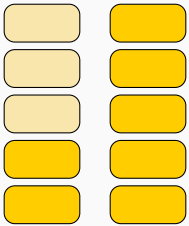
If the stacks have different heights, the first player has a *winning strategy*. . .

Two-stack Nim



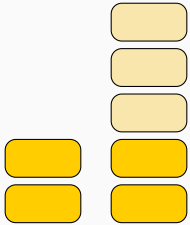
The first player can always balance the stacks, taking coins from taller stack.

Two-stack Nim



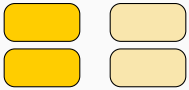
And whatever move the next player makes…

Two-stack Nim



Can be mirrored by the first player – bringing the stacks back to a balanced position.

Two-stack Nim



Whenever the second player takes the last coins from one stack, the first player has reached a winning position.

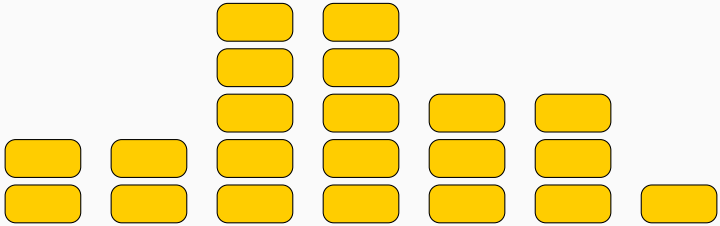
More than two stacks

What is the smallest generalization that we can make of this situation?

More than two stacks

What is the smallest generalization that we can make of this situation?

What if we have *n* pairs of stacks of equal height and one additional stack?



Then we clearly have a winning strategy – we can remove the odd stack and afterwards mirror the opponent's moves.

More generally

But what if we have an *arbitrary* number of stacks, storing an *arbitrary* number of coins?

To find a winning strategy, we need a more general notion of *balanced* position.

The key idea will be to define a notion such that:

- the empty board is balanced;
- there is always a balancing move from an unbalanced position;
- every move from a balanced position yields an unbalanced position.

These three statements together guarantee that the player that starts from an unbalanced position has a winning strategy.

So when is an *arbitrary* position balanced?

Balanced positions

We say that an arbitrary Nim position is *balanced* precisely when *there is an overall **even** number of ones in each digit place of the binary representation of the number of coins in each stack.*

For example: consider the position 2,3,5

Converting these to binary gives: 010, 011, 101. Now we can count the ones in each column:

```
0 1 0
0 1 1
1 0 1
------
1 2 2
```

Because there are an odd number of ones in the first digit, the position is *unbalanced*.

Question: What move would lead to a balanced position?

Balanced positions

Now we claim that this notion of balanced position satisfies these three properties:

- the empty board is balanced;
- there is always a balancing move from an unbalanced position;
- every move from a balanced position yields an unbalanced position.

Question:

Convince yourself that these properties hold.

Balanced positions

Now we claim that this notion of balanced position satisfies these three properties:

- the empty board is balanced;
- there is always a balancing move from an unbalanced position;
- every move from a balanced position yields an unbalanced position.

Question:

Convince yourself that these properties hold.

Let's go through these three properties one by one.

Balanced positions

Claim: The empty position is balanced

Balanced positions

Claim: The empty position is balanced

Clearly this is true: there are no coins left – hence the binary representation of the number of coins in each stack consists of zeros. So it is clear that the number of ones is even (as there are no ones).

Balanced positions

Claim: there is always a balancing move from an unbalanced position;

Balanced positions

Claim: there is always a balancing move from an unbalanced position;

A position that is unbalanced has an odd number of ones in at least one position. Choose the *most significant* position *i* with an odd number of ones and any stack that has at least 2^i coins.

By removing coins from this stack, we will balance position *i*.

By removing *just the right number* of coins from this stack, we can ensure that all the other positions also become balanced – this leads to an overall position that is balanced.

Balanced positions

Claim: every move from a balanced position yields an unbalanced position.

Balanced positions

Claim: every move from a balanced position yields an unbalanced position.

Suppose we are in a balanced position. Removing any number of coins from any stack will flip one of the bits in the binary representation from 1 to 0 – resulting in an unbalanced position overall.

Winning Nim

Claim: The player that starts an arbitrary game of Nim from an *unbalanced* position has a winning strategy.

This player can always make a move that leads to balanced position; every move their opponent makes leads to an unbalanced position.

The first player's last move will balance the board, leaving no coins left over.

Symmetrically, if the first player starts from a *balanced* position, their opponent has a winning strategy.

Back to logic

There are different ways of formalizing this method…

Back to logic

There are different ways of formalizing this method…

Let S be the set of states of Nim: that is all possible board positions.

Let B be the set $\{\texttt{balanced}, \texttt{unbalanced}\}$.

Back to logic

There are different ways of formalizing this method…

Let S be the set of states of Nim: that is all possible board positions.

Let B be the set $\{\texttt{balanced}, \texttt{unbalanced}\}$.

- We have constructed a function $f : S \to B$.
- We can also think of this as a predicate `isBalanced` on S.
- We can also think about S as being *quotiented* by the equivalence relation given by f (i.e., we take the set of equivalence classes of the relation $s \sim t$ iff s and t are both balanced or both unbalanced).

Then we think about this as a game with only *two states*: balanced and unbalanced.

Chomp

Chomp

Consider an $n \times m$ chocolate bar, where the top-left square is covered in ketchup.

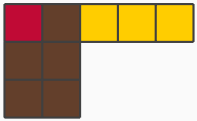
Two players take turns taking bites from the chocolate bar. With each bite, a player designates an uneaten square of chocolate and eats it, together with all the other squares below and to the right of the square they have chosen.

The aim of the game is to make your opponent eat the square with ketchup on it.

Chomp - example

Chomp - example

Chomp - example



Chomp - example

Chomp - example

Chomp - example

Chomp - example

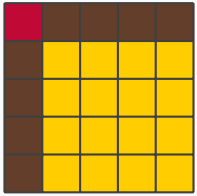
Chomp - example

Bon appétit!

Chomp - analysis

Let's consider a few degenerate cases for Chomp to try and find a winning strategy.

- 1×1 - the first player loses immediately;
- $1 \times n$ and $n \times 1$ - the first player wins by eating the entire chocolate bar, except the last square;
- $n \times n$ - the first player can eat the bottom $(n - 1) \times (n - 1)$ part of the chocolate bar:



Now the first player can copy each of their opponent's moves, forcing them to eat the last square.

Chomp - analysis

Surprisingly - there is no *general* strategy that tells us how to win an *arbitrary* game of Chomp. This is an open mathematical problem!

But we **can** prove that a winning strategy must exist!

Proving this requires we develop a general theory of games first…

The fundamental theorem of finite games

The fundamental theorem

Theorem

In any finite two-player game of no chance with perfect information, without draws, one of the players has a winning strategy.

The fundamental theorem

Theorem

In any finite two-player game of no chance with perfect information, without draws, one of the players has a winning strategy.

Proof sketch

Starting from the initial state of the game, we can construct the *game tree*, which captures every possible move from this initial state. At each node in the tree, we repeatedly branch for every possible move, yielding new smaller states.

As we know the game is finite, this process ends eventually, producing a game tree of finite depth.

In each leaf – when there are no moves possible – we know who has won; there is no draw possible.

Proof sketch (continued)

We can now work our way up through the tree, figuring out who is in a winning position in each node in the game tree.

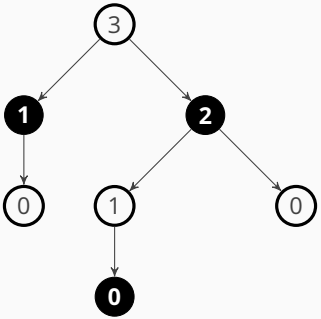
If is player A's turn, he can choose which move to make. If there is **any** subtree that brings player B into a losing position, this node is a winning position for player A.

If **each** subtree brings player B into a winning position, this node is in a losing position for player A.

By repeating this through the entire tree, we learn whether the root of our tree is in a winning or a losing position for player A.

In this fashion, we can determine that for one of the two players, a winning strategy must exist – as we know for the root of the tree which player is in a winning position.

Example

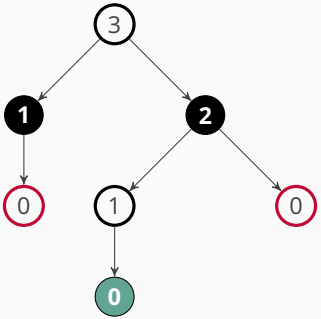


Here is the game tree for a coin taking game with three coins. Players take turns removing 1 or 2 coins. If you take the last coin, you win.

Each node is labelled with the number of remaining coins.

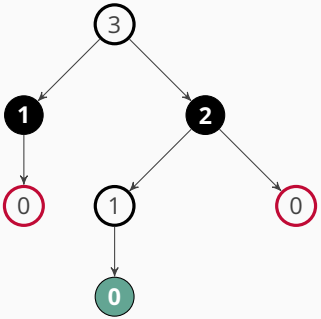
I'll refer to the players as White and Black—as in chess—where White moves first.

Example



If White has won, we colour the leaf green; if Black has won, we colour the node red.

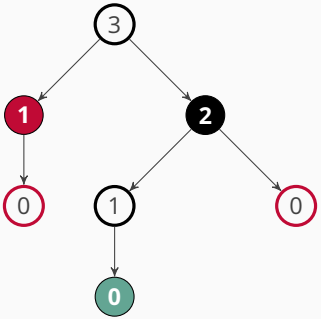
Example



We can now work our way up the tree:

- in a node where White cannot make a move to a losing state for Black, then White is in a losing position;
- in a node where Black cannot make a move to a losing state for White, then Black is in a losing position.

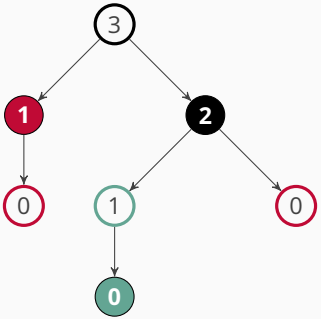
Example



We can now work our way up the tree:

- in a node where White cannot make a move to a losing state for Black, then White is in a losing position;
- in a node where Black cannot make a move to a losing state for White, then Black is in a losing position.

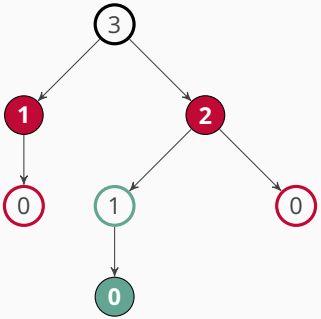
Example



We can now work our way up the tree:

- in a node where White cannot make a move to a losing state for Black, then White is in a losing position;
- in a node where Black cannot make a move to a losing state for White, then Black is in a losing position.

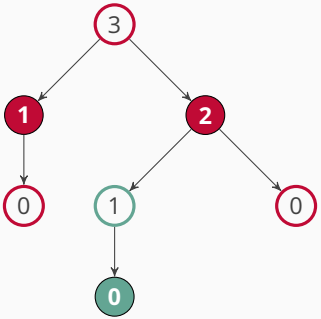
Example



We can now work our way up the tree:

- in a node where White cannot make a move to a losing state for Black, then White is in a losing position;
- in a node where Black cannot make a move to a losing state for White, then Black is in a losing position.

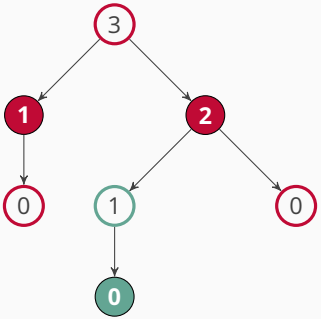
Example



We can now work our way up the tree:

- in a node where White cannot make a move to a losing state for Black, then White is in a losing position;
- in a node where Black cannot make a move to a losing state for White, then Black is in a losing position.

Example



This example is kept simple to fit on the slide – but it shows that black can force white into a losing position.

But crucially, this technique works *for any finite game of no chance with perfect information* – even if this is not tractable for some games (like chess or go).

An alternative proof

The fundamental theorem shows that (in a game with no draws) if the first player does not have a winning strategy, the second player does.

Can we make this make this more precise?

An alternative proof

The fundamental theorem shows that (in a game with no draws) if the first player does not have a winning strategy, the second player does.

Can we make this make this more precise?

Suppose player A has a winning strategy – then there is a series of moves a_1, a_2,…, a_n that player A can make in response to the moves b_1, b_2, …, b_n that player B may make. These moves satisfy:

$\exists a_1\ \forall b_1\ \exists a_2\ \forall b_2 \ldots \exists a_n\ \forall b_n$ player A has won.

We know that $P \vee \neg P$ holds, for any statement P. So in particular, player A has a winning strategy or $\neg$(player A has a winning strategy).

$\neg(\exists a_1\ \forall b_1\ \exists a_2\ \forall b_2 \ldots \exists a_n\ \forall b_n$ player A has won).

Let's look at this last statement more carefully.

An alternative proof

Consider the following proposition:

$\neg(\exists a_1\ \forall b_1\ \exists a_2\ \forall b_2\ \ldots\ \exists a_n\ \forall b_n$ player A has won).

Using the de Morgan laws, this is equivalent to:

$\forall a_1\ \exists b_1\ \forall a_2\ \exists b_2\ \ldots\ \forall a_n\ \exists b_n$ $\neg$(player A has won).

But as there is no draw possible, player A not winning means player B has won:

$\forall\ a_1\ \exists b_1\ \forall a_2\ \exists b_2\ \ldots\ \forall a_n\ \exists b_n$ player B has won

But this statement says that player B has a winning strategy!

In other words, either player A or player B has a winning strategy – exactly what we set out to prove.

Chomp revisited

This theorem tells us that one of the two players in Chomp has a winning strategy…

…even if we do not know what it is.

For any particular instance of Chomp, we can construct the game tree and calculate which player has a winning strategy.

The book uses a slightly different formulation of the fundamental theorem:

Theorem In a two-player game-of-no-chance of perfect information, either of the players has a winning strategy or they both have drawing strategies.

Corollary If a game cannot end in a draw, one of the two players has a winning strategy.

Both formulations are equivalent however.

Other games

There are lots of other examples of games to be found in the book, including Fibonacci Nim, Hex, and Bridg-it.

I won't cover these in the lecture – but they're fun to read about and learn.

The key ideas are the same over and over again:

- analyse what constitutes a winning position;
- show that the initial state of the board is a winning position for one of the two players.

Oftentimes there is some creativity involved, identifying properties of winning positions that are preserved throughout the game (like the balanced positions in Nim).

Games in computer science

Why study these games?

- Many interactions between computers – such as network protocols – can be defined in terms of a ‘game’ between sender and receiver.
- We can use the idea of games to assign semantics to (propositional) logic – one player tries to prove a statement, while the other tries to falsify it.
- Many security properties of cryptographic functions can be formulated as a game with an attacker. An encryption mechanism is ‘safe’ if the chance of the attacker winning the game (by, for example, guessing the plain text) is very small.

The applications of these ideas are not just restricted to the study of Nim!

Material

- Modelling Computing Systems Chapter 10