From heaps of matches to the limits of computability

Urban Larsson, joint with Johan Wästlund
Chalmers University of Technology,
CANT 2011

May 24, 2011

Table of contents

A children's game

Games of more than one heap

Cellular automata

An undecidable class of games of two heaps

Invariance

Undecidability of game equivalence?

Place a heap of tokens on a table

Figure: Two player's alternate in removing tokens. Last player wins.

The move options

Figure: Rules: remove "one", "two" or "three" tokens.

How to win

Figure: The Previous player's safe positions are divisible by four.

A winning move

Figure: P-positions divisible by four.

The new move options are loosing

Figure: P-positions divisible by four.

Periodic P-positions

- If the set of numbers allowed to remove from the heap is finite,

Periodic P-positions

- If the set of numbers allowed to remove from the heap is finite,
- the set of P-positions will ultimately become periodic.

Periodic P-positions

- If the set of numbers allowed to remove from the heap is finite,
- the set of P-positions will ultimately become periodic.
- The game cannot embrace any mysteries.

Several heaps

- A fixed number d of heaps,

Several heaps

- A fixed number d of heaps,
- a position can be regarded as a d-dimensional vector $x = (x_1, \ldots, x_d)$ of non-negative integers.

Several heaps

- A fixed number d of heaps,
- a position can be regarded as a d-dimensional vector $x = (x_1, \ldots, x_d)$ of non-negative integers.
- The rules of the game are encoded by a finite set $\mathcal{M}$ of integer vectors that specify the permitted moves.

Several heaps

- A fixed number d of heaps,
- a position can be regarded as a d-dimensional vector $x = (x_1, \dots, x_d)$ of non-negative integers.
- The rules of the game are encoded by a finite set $\mathcal{M}$ of integer vectors that specify the permitted moves.
- If $(m_1, \dots, m_d) \in \mathcal{M}$, then from position $(x_1, \dots, x_d)$, a player can move to position $(x_1 + m_1, \dots, x_d + m_d)$, provided all of the numbers $x_1 + m_1, \dots, x_d + m_d$ are non-negative.

Invariant games and termination

- Invariant games,

Invariant games and termination

- Invariant games,
- The move options given by $\mathcal{M}$ are independent of the position played from,

Invariant games and termination

- Invariant games,
- The move options given by $\mathcal{M}$ are independent of the position played from,
- Implicit in the idea of heaps of tokens, we should not have to count the remaining tokens in order to play by the rules.

Invariant games and termination

- Invariant games,
- The move options given by $\mathcal{M}$ are independent of the position played from,
- Implicit in the idea of heaps of tokens, we should not have to count the remaining tokens in order to play by the rules.
- If some moves involve adding tokens to heaps, the game does not necessarily have to terminate.

Invariant games and termination

- Invariant games,
- The move options given by $\mathcal{M}$ are independent of the position played from,
- Implicit in the idea of heaps of tokens, we should not have to count the remaining tokens in order to play by the rules.
- If some moves involve adding tokens to heaps, the game does not necessarily have to terminate.
- In each move, the total number of tokens decreases.

Can we understand the P-positions?

- Each game $\mathcal{M}$ has a set $\mathcal{P}(\mathcal{M})$ of P-positions,

Can we understand the P-positions?

- Each game $\mathcal{M}$ has a set $\mathcal{P}(\mathcal{M})$ of P-positions,
- From the list $\mathcal{M}$ of permitted moves, is it possible to completely understand $\mathcal{P}(\mathcal{M})$?

Can we understand the P-positions?

- Each game $\mathcal{M}$ has a set $\mathcal{P}(\mathcal{M})$ of P-positions,
- From the list $\mathcal{M}$ of permitted moves, is it possible to completely understand $\mathcal{P}(\mathcal{M})$?
- Yes, at least in one sense:

Can we understand the P-positions?

- Each game $\mathcal{M}$ has a set $\mathcal{P}(\mathcal{M})$ of P-positions,
- From the list $\mathcal{M}$ of permitted moves, is it possible to completely understand $\mathcal{P}(\mathcal{M})$?
- Yes, at least in one sense:
- For any given position $x = (x_1, \ldots, x_d)$ we can determine recursively whether or not x is a P-position by first computing the status of all positions with fewer tokens.

Can we understand the P-positions?

- Each game $\mathcal{M}$ has a set $\mathcal{P}(\mathcal{M})$ of P-positions,
- From the list $\mathcal{M}$ of permitted moves, is it possible to completely understand $\mathcal{P}(\mathcal{M})$?
- Yes, at least in one sense:
- For any given position $x = (x_1, \ldots, x_d)$ we can determine recursively whether or not x is a P-position by first computing the status of all positions with fewer tokens.
- The position x is in $\mathcal{P}(\mathcal{M})$ iff there is no move from x to a position in $\mathcal{P}(\mathcal{M})$.

Patterns and P-positions

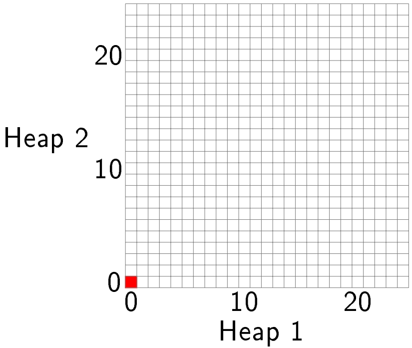


Figure: The P-positions of the game $\mathcal{M} = \{(-1,-3),(-2,1)\}$.

Patterns and P-positions

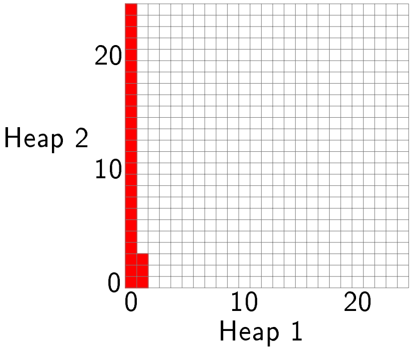


Figure: The P-positions of the game $\mathcal{M} = \{(-1,-3),(-2,1)\}$.

Patterns and P-positions

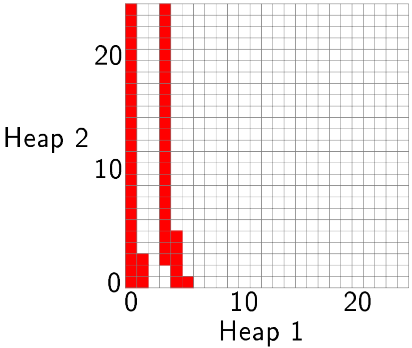


Figure: The P-positions of the game $\mathcal{M} = \{(-1,-3),(-2,1)\}$.

Patterns and P-positions

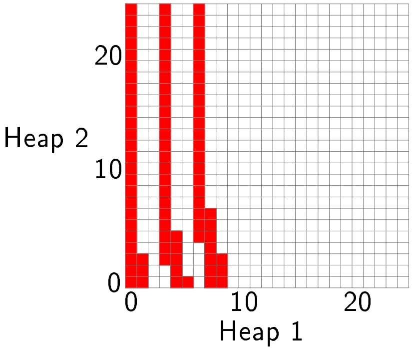


Figure: The P-positions of the game $\mathcal{M} = \{(-1,-3),(-2,1)\}$.

Patterns and P-positions

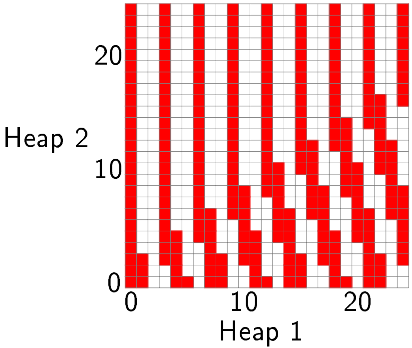


Figure: The P-positions of the game $\mathcal{M} = \{(-1,-3),(-2,1)\}$.

Patterns and P-positions

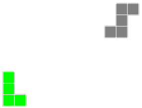


Figure: A pattern that occurs in $\mathcal{P}$ an one that does not

Patterns and P-positions

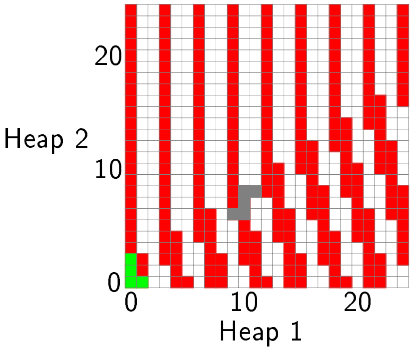


Figure: Pattern occurrence

Patterns and P-positions

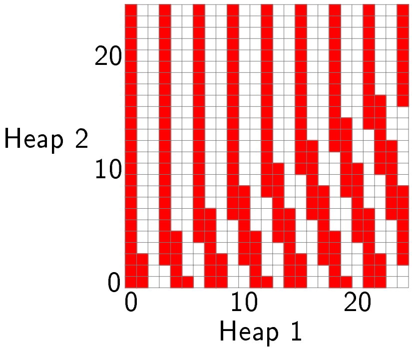


Figure: Pattern occurrence.

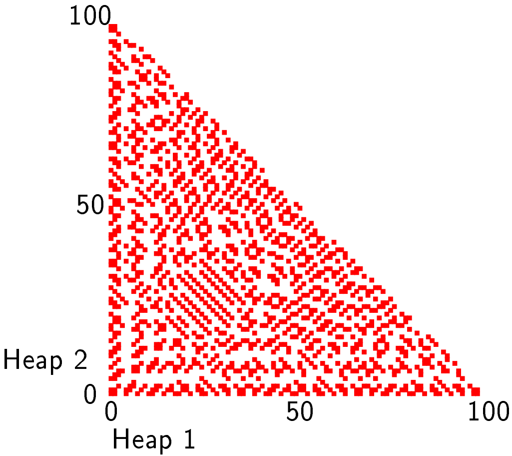
Undecidable games

- In general, the P-positions of a given game are much harder to understand.

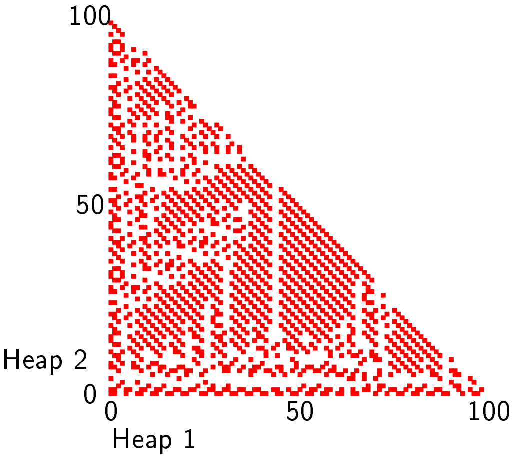
Undecidable games

- In general, the P-positions of a given game are much harder to understand.
- If we cannot answer questions of pattern occurrence, we cannot claim to have full understanding of the set of P-positions of a game.

$\mathcal{M} = \{(0,-2), (-2,0), (2,-3), (-3,2), (-3,-3)\}$



$$\mathcal{M} = \{(0,-2),(-2,0),(2,-3),(-3,2),(-5,4),(-5,-2),(-4,-3),(-1,-4)\}$$



Undecidable games

From the list $\mathcal{M}$ of permitted moves, it is not possible to completely understand $P(\mathcal{M})$ in the following sense:

Undecidable games

From the list $\mathcal{M}$ of permitted moves, it is not possible to completely understand $P(\mathcal{M})$ in the following sense:

Theorem

There is no algorithm that, given as input the set of moves $\mathcal{M}$ and a finite pattern, decides whether the given pattern occurs in $P(\mathcal{M})$.

Cellular automata

- Cellular automata give rise to similar 2-dimensional patterns.

Cellular automata

- Cellular automata give rise to similar 2-dimensional patterns.
- It is known that some questions including pattern-occurrence are algorithmically undecidable.

Cellular automata

- Cellular automata give rise to similar 2-dimensional patterns.
- It is known that some questions including pattern-occurrence are algorithmically undecidable.
- A restricted class of CA's for which the undecidability results are known to still hold.

Cellular automata

- Cellular automata give rise to similar 2-dimensional patterns.
- It is known that some questions including pattern-occurrence are algorithmically undecidable.
- A restricted class of CA's for which the undecidability results are known to still hold.
- Two states (0 and 1), and the state of cell i at time t is denoted by $x_{t,i}$.

Cellular automata

- Cellular automata give rise to similar 2-dimensional patterns.
- It is known that some questions including pattern-occurrence are algorithmically undecidable.
- A restricted class of CA's for which the undecidability results are known to still hold.
- Two states (0 and 1), and the state of cell i at time t is denoted by $x_{t,i}$.
- The starting configuration is ...000111... (the first '1' at position 0).

Cellular automata

- The update rule, a number n and a Boolean function f taking n bits of input.

Cellular automata

- The update rule, a number n and a Boolean function f taking n bits of input.
- The states are updated by

$$x_{t+1,i} = f(x_{t,i-n+1}, x_{t,i-n+2}, \dots, x_{t,i}),$$

Cellular automata

- The update rule, a number n and a Boolean function f taking n bits of input.
- The states are updated by

$$x_{t+1,i} = f(x_{t,i-n+1}, x_{t,i-n+2}, \dots, x_{t,i}),$$

- $x_{t+1,i}$ depends on $x_{t,i}$ and the $n-1$ cells immediately to the left of $x_{t,i}$.

Cellular automata

- The update rule, a number n and a Boolean function f taking n bits of input.
- The states are updated by

$$x_{t+1,i} = f(x_{t,i-n+1}, x_{t,i-n+2}, \ldots, x_{t,i}),$$

- $x_{t+1,i}$ depends on $x_{t,i}$ and the $n-1$ cells immediately to the left of $x_{t,i}$.
- We require that $f(0,\ldots,0) = 0$, so that $x_{t,i} = 0$ whenever $i < 0$.

Example

- $n = 2$ and letting the Boolean function be

$$f(x, y) = x \oplus y,$$

Example

- $n = 2$ and letting the Boolean function be

$$f(x, y) = x \oplus y,$$

- $f(x, y)$ is equal to 0 if $x = y$ and 1 if $x \neq y$.

Example

- $n = 2$ and letting the Boolean function be

$$f(x, y) = x \oplus y,$$

- $f(x, y)$ is equal to 0 if $x = y$ and 1 if $x \neq y$.
- The 1's are represented by red squares.

Example

- $n = 2$ and letting the Boolean function be

$$f(x, y) = x \oplus y,$$

- $f(x, y)$ is equal to 0 if $x = y$ and 1 if $x \neq y$.
- The 1's are represented by red squares.
- The leftmost column represents $x_{t,0}$.

Example

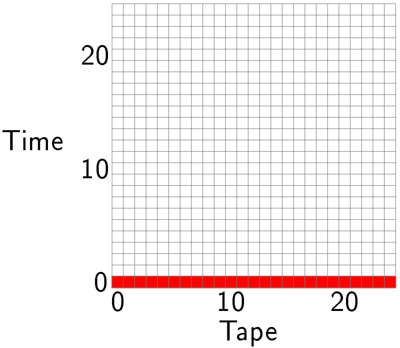


Figure: The CA given by $f(x, y) = x \oplus y$ (Wolfram's rule 90).

Example

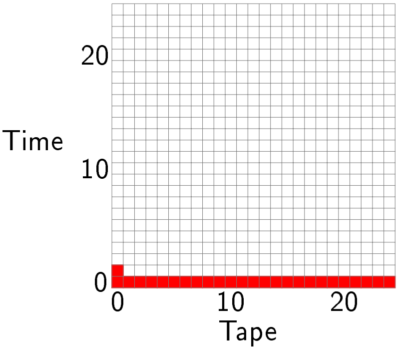


Figure: The CA given by $f(x, y) = x \oplus y$ (Wolfram's rule 90).

Example

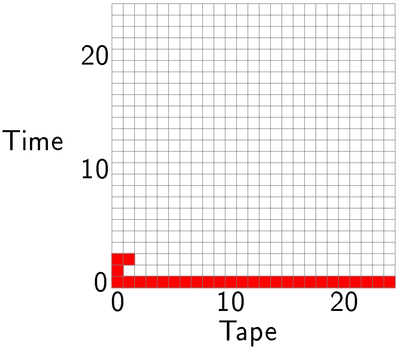


Figure: The CA given by $f(x, y) = x \oplus y$ (Wolfram's rule 90).

Example

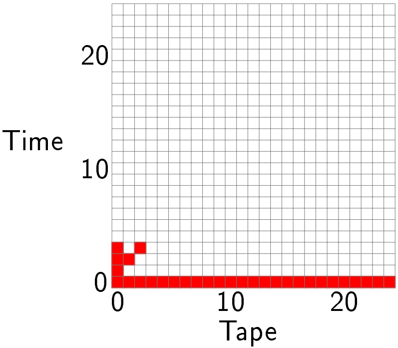


Figure: The CA given by $f(x, y) = x \oplus y$ (Wolfram's rule 90).

Example

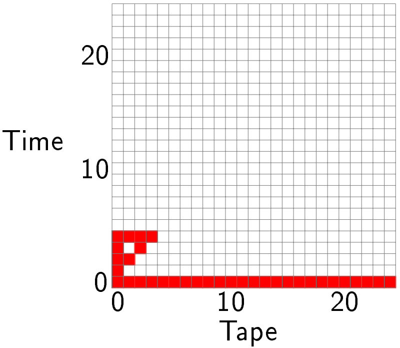


Figure: The CA given by $f(x, y) = x \oplus y$ (Wolfram's rule 90).

Example

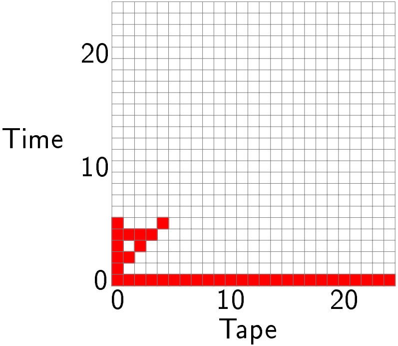


Figure: The CA given by $f(x, y) = x \oplus y$ (Wolfram's rule 90).

Example

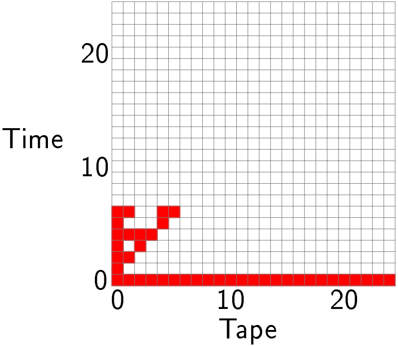


Figure: The CA given by $f(x, y) = x \oplus y$ (Wolfram's rule 90).

Example

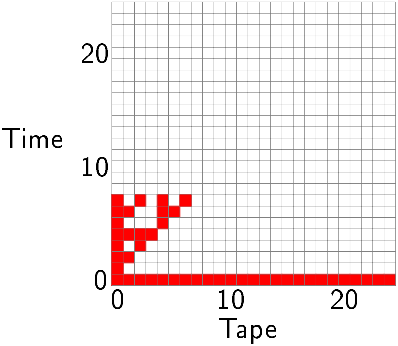


Figure: The CA given by $f(x, y) = x \oplus y$ (Wolfram's rule 90).

Example

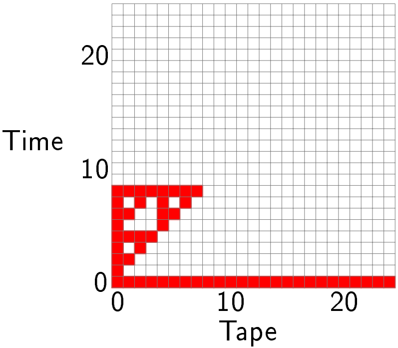


Figure: The CA given by $f(x, y) = x \oplus y$ (Wolfram's rule 90).

The pattern is Pascal's triangle modulo 2

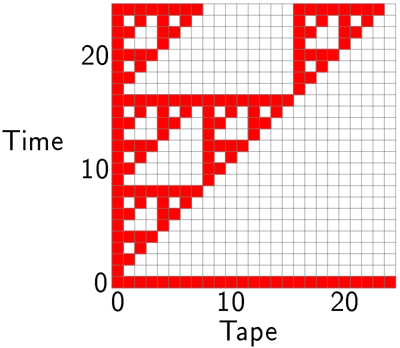


Figure: The CA given by $f(x, y) = x \oplus y$ (Wolfram's rule 90).

A well-known undecidability result

Pascal's triangle modulo 2 is non-periodic, but well understood. There are Boolean functions that give rise to more difficult behavior (e.g. Wolfram's Rule 110). The following theorem is well-known.

A well-known undecidability result

Pascal's triangle modulo 2 is non-periodic, but well understood. There are Boolean functions that give rise to more difficult behavior (e.g. Wolfram's Rule 110). The following theorem is well-known.

Theorem

There is no algorithm that takes input n, f, and a bit-string s of length n, and answers whether or not s ever occurs in the CA given by f.

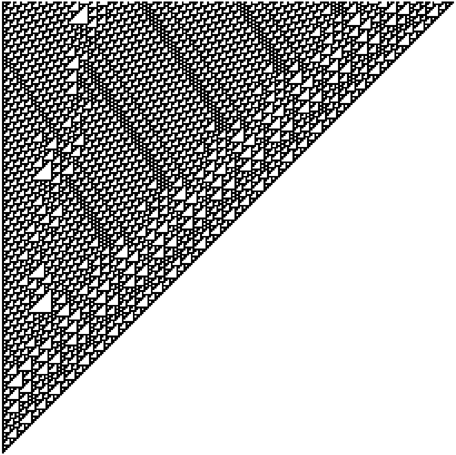
CA: Rule 110

- The CA given by Stephen Wolfram's rule 110 and a (doubly) periodic initial pattern was proved undecidable by Matthew Cook around year 2000.

CA: Rule 110

- The CA given by Stephen Wolfram's rule 110 and a (doubly) periodic initial pattern was proved undecidable by Matthew Cook around year 2000.
- Update rule: "0s are changed to 1s at all positions where the value to the right is a 1, while 1s are changed to 0s at all positions where the values to the left and right are both 1".

CA, rule 110, time moves upwards



Outline: The CA's equivalence with games

- Define a class of non-invariant games and show that arbitrary CA's can be reduced to computing their P-positions.

Outline: The CA's equivalence with games

- Define a class of non-invariant games and show that arbitrary CA's can be reduced to computing their P-positions.
- Two heaps, the tape-heap and the time-heap:

Outline: The CA's equivalence with games

- Define a class of non-invariant games and show that arbitrary CA's can be reduced to computing their P-positions.
- Two heaps, the tape-heap and the time-heap:
- The tape-heap: its number of tokens corresponds to a position on the tape

Outline: The CA's equivalence with games

- Define a class of non-invariant games and show that arbitrary CA's can be reduced to computing their P-positions.
- Two heaps, the tape-heap and the time-heap:
- The tape-heap: its number of tokens corresponds to a position on the tape
- The Time heap: we insert some space, k, to allow for the rules of the game to "compute" an arbitrary Boolean function f.

Outline: The CA's equivalence with games

- Define a class of non-invariant games and show that arbitrary CA's can be reduced to computing their P-positions.
- Two heaps, the tape-heap and the time-heap:
- The tape-heap: its number of tokens corresponds to a position on the tape
- The Time heap: we insert some space, k, to allow for the rules of the game to "compute" an arbitrary Boolean function f.
- The positions with km tokens in the time-heap will correspond to the state of the CA at time m.

The permitted moves depend on the congruence class of the time heap

- Definition of a class of modular games.

The permitted moves depend on the congruence class of the time heap

- Definition of a class of modular games.
- For some positive integer k, depending on our move function f, there are sets of integer vectors $\mathcal{M}_0, \ldots, \mathcal{M}_{k-1}$ that specify the available move options.

The permitted moves depend on the congruence class of the time heap

- Definition of a class of modular games.
- For some positive integer k, depending on our move function f, there are sets of integer vectors $\mathcal{M}_0, \ldots, \mathcal{M}_{k-1}$ that specify the available move options.
- Let (a_1, a_2) denote a given position, where a_1 is the number of tokens in the tape-heap and a_2 is the number of tokens in the time-heap,

The permitted moves depend on the congruence class of the time heap

- Definition of a class of modular games.
- For some positive integer k, depending on our move function f, there are sets of integer vectors $\mathcal{M}_0, \ldots, \mathcal{M}_{k-1}$ that specify the available move options.
- Let (a_1, a_2) denote a given position, where a_1 is the number of tokens in the tape-heap and a_2 is the number of tokens in the time-heap,
- Then the set of available moves is given by $\mathcal{M}_i$, where $i \equiv a_2 \pmod{k}$.

The construction of a modular game computer

- Let square brackets $[\cdot]$ denote $1 - \max(\cdot)$.

The construction of a modular game computer

- Let square brackets $[\cdot]$ denote $1 - \max(\cdot)$.
- For instance

$$[xyz] = 1 - \max(x, y, z),$$

and by convention $[\,] = 1$.

The construction of a modular game computer

- Let square brackets $[\cdot]$ denote $1 - \max(\cdot)$.
- For instance
$$[xyz] = 1 - \max(x, y, z),$$
and by convention $[\,] = 1$.
- Every Boolean function can be expressed in terms of nested brackets.

The construction of a modular game computer

- Let square brackets $[\cdot]$ denote $1 - \max(\cdot)$.
- For instance
  $$[xyz] = 1 - \max(x, y, z),$$
  and by convention $[\,] = 1$.
- Every Boolean function can be expressed in terms of nested brackets.
- The set consisting of & and $\sim$ (negation) is a complete set of connectives in propositional logic.

The construction of a modular game computer

- Let square brackets $[\cdot]$ denote $1 - \max(\cdot)$.
- For instance
$$[xyz] = 1 - \max(x, y, z),$$
and by convention $[\,] = 1$.
- Every Boolean function can be expressed in terms of nested brackets.
- The set consisting of & and $\sim$ (negation) is a complete set of connectives in propositional logic.
- For instance the function $f(x, y) = x \oplus y$ can be expressed as
$$x \oplus y = [[xy][[x][y]]].$$

Computing the function f

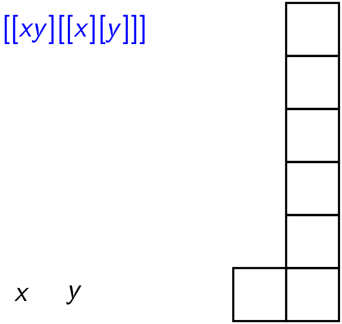


Figure: A modular game computing $f(x, y) = x \oplus y$ in five steps. The arrows indicate move options. The value of each cell is the bracket of the values of all its options.

Computing the function f

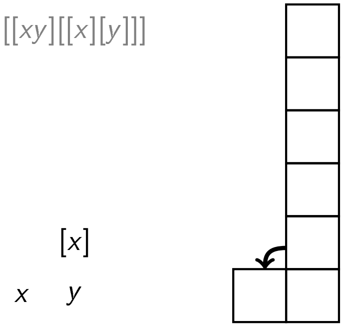


Figure: A modular game computing $f(x,y) = x \oplus y$ in five steps. The arrows indicate move options. The value of each cell is the bracket of the values of all its options.

Computing the function f

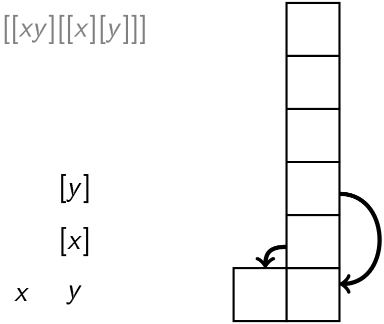


Figure: A modular game computing $f(x, y) = x \oplus y$ in five steps. The arrows indicate move options. The value of each cell is the bracket of the values of all its options.

Computing the function f

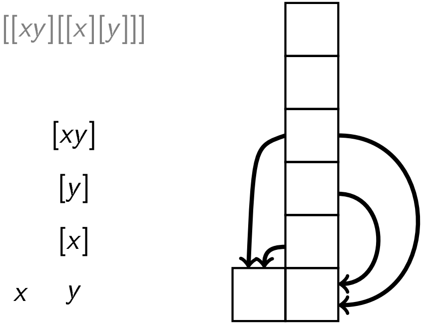


Figure: A modular game computing $f(x, y) = x \oplus y$ in five steps. The arrows indicate move options. The value of each cell is the bracket of the values of all its options.

Computing the function f

$[[xy][[x][y]]]$

$[[x][y]]$

$[xy]$

$[y]$

$[x]$

$x \quad y$

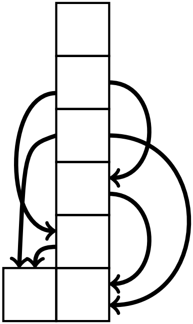


Figure: A modular game computing $f(x, y) = x \oplus y$ in five steps. The arrows indicate move options. The value of each cell is the bracket of the values of all its options.

Computing the function f

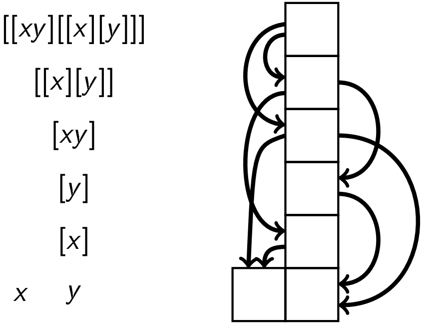


Figure: A modular game computing $f(x, y) = x \oplus y$ in five steps. The arrows indicate move options. The value of each cell is the bracket of the values of all its options.

and its moves

The construction in the last figure corresponds to the modular game

- $\mathcal{M}_1 = \{(-1,-1)\}$

and its moves

The construction in the last figure corresponds to the modular game

- $\mathcal{M}_2 = \{(0, -2)\}$
- $\mathcal{M}_1 = \{(-1, -1)\}$

and its moves

The construction in the last figure corresponds to the modular game

- $\mathcal{M}_3 = \{(0, -3), (-1, -3)\}$
- $\mathcal{M}_2 = \{(0, -2)\}$
- $\mathcal{M}_1 = \{(-1, -1)\}$

and its moves

The construction in the last figure corresponds to the modular game

- $\mathcal{M}_4 = \{(0, -2), (0, -3)\}$
- $\mathcal{M}_3 = \{(0, -3), (-1, -3)\}$
- $\mathcal{M}_2 = \{(0, -2)\}$
- $\mathcal{M}_1 = \{(-1, -1)\}$

and its moves

The construction in the last figure corresponds to the modular game

- $\mathcal{M}_0 = \{(0, -1), (0, -2)\}$
- $\mathcal{M}_4 = \{(0, -2), (0, -3)\}$
- $\mathcal{M}_3 = \{(0, -3), (-1, -3)\}$
- $\mathcal{M}_2 = \{(0, -2)\}$
- $\mathcal{M}_1 = \{(-1, -1)\}$

The P-positions

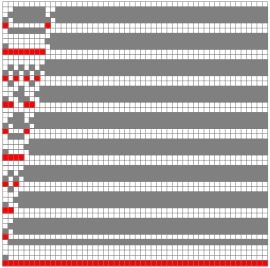


Figure: A modular game emulating $f(x,y) = x \oplus y$. Here the P-positions with fewer than 50 tokens in each heap are represented by filled squares. Rows corresponding to $a_2 \equiv 0 \pmod 5$ are highlighted by drawing the P-positions in red.

A gadget keeps track of the congruence class of the time-heap

- Our invariant game has a time-heap, a tape-heap and a gadget with k more heaps.

A gadget keeps track of the congruence class of the time-heap

- Our invariant game has a time-heap, a tape-heap and a gadget with k more heaps.
- Assume that one of the heaps in the gadget contains a single match and the remaining $k-1$ heaps are empty

A gadget keeps track of the congruence class of the time-heap

- Our invariant game has a time-heap, a tape-heap and a gadget with k more heaps.
- Assume that one of the heaps in the gadget contains a single match and the remaining $k-1$ heaps are empty
- This need not be the case, but we want a single match in the gadget to follow the time-heap modulo k.

A gadget keeps track of the congruence class of the time-heap

- Our invariant game has a time-heap, a tape-heap and a gadget with k more heaps.
- Assume that one of the heaps in the gadget contains a single match and the remaining $k-1$ heaps are empty
- This need not be the case, but we want a single match in the gadget to follow the time-heap modulo k.
- The heaps of the gadget may be numbered $0, \ldots, k-1$

The gadget allows us to emulate a modular game by an invariant game

- For each of the move sets $\mathcal{M}_i$ of the modular game, we introduce corresponding moves in the invariant game:

The gadget allows us to emulate a modular game by an invariant game

- For each of the move sets $\mathcal{M}_i$ of the modular game, we introduce corresponding moves in the invariant game:
- The ith heap of the gadget is emptied,

The gadget allows us to emulate a modular game by an invariant game

- For each of the move sets $\mathcal{M}_i$ of the modular game, we introduce corresponding moves in the invariant game:
- The ith heap of the gadget is emptied,
- The tape- and time-heaps are affected as given by $\mathcal{M}_i$,

The gadget allows us to emulate a modular game by an invariant game

- For each of the move sets $\mathcal{M}_i$ of the modular game, we introduce corresponding moves in the invariant game:
- The ith heap of the gadget is emptied,
- The tape- and time-heaps are affected as given by $\mathcal{M}_i$,
- A match is added to the heap of the gadget corresponding to the new modulus of the time-heap.

Triviality ruled out

- In this way, the single token in the gadget follows the congruence class of the time heap.

Triviality ruled out

- In this way, the single token in the gadget follows the congruence class of the time heap.
- A subset of positions of the invariant game emulates the modular game, namely those where

Triviality ruled out

- In this way, the single token in the gadget follows the congruence class of the time heap.
- A subset of positions of the invariant game emulates the modular game, namely those where
- (i) there is exactly one match in the gadget, and

Triviality ruled out

- In this way, the single token in the gadget follows the congruence class of the time heap.
- A subset of positions of the invariant game emulates the modular game, namely those where
- (i) there is exactly one match in the gadget, and
- (ii) this match is in the heap corresponding to the number of tokens in the time-heap modulo k.

Triviality ruled out

- In this way, the single token in the gadget follows the congruence class of the time heap.
- A subset of positions of the invariant game emulates the modular game, namely those where
- (i) there is exactly one match in the gadget, and
- (ii) this match is in the heap corresponding to the number of tokens in the time-heap modulo k.
- We may conclude that the pattern-occurrence problem is computationally unsolvable for the invariant game

Triviality ruled out

- In this way, the single token in the gadget follows the congruence class of the time heap.
- A subset of positions of the invariant game emulates the modular game, namely those where
- (i) there is exactly one match in the gadget, and
- (ii) this match is in the heap corresponding to the number of tokens in the time-heap modulo k.
- We may conclude that the pattern-occurrence problem is computationally unsolvable for the invariant game
- if and only if the patterns we ask for do not "trivially" occur for positions violating (i) or (ii).

If the gadget contains more than one match

- All earlier moves will still be available,

If the gadget contains more than one match

- All earlier moves will still be available,
- Choose some large number N, and allow any move that transfers two tokens in the gadget to two other heaps and removes any number smaller than N from the two main heaps.

If the gadget contains more than one match

- All earlier moves will still be available,
- Choose some large number N, and allow any move that transfers two tokens in the gadget to two other heaps and removes any number smaller than N from the two main heaps.
- Since the number of tokens in the gadget will never change, this will give a trivial periodic pattern of P-positions in the main heaps.

If the gadget is out of phase

The gadget contains exactly one match, but this match does not correspond to the number of tokens in the time-heap modulo k.

If the gadget is out of phase

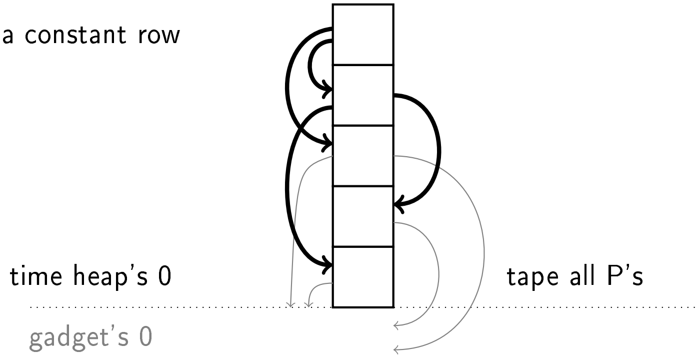


Figure: The gadget "thinks" the time heap is congruent to zero when it is not. The gray moves will not be possible.

If the gadget is out of phase

- The least row that corresponds to a finished computation congruent to 0 modulo k will be constant, since no information has been propagated sideways up to this point.

If the gadget is out of phase

- The least row that corresponds to a finished computation congruent to 0 modulo k will be constant, since no information has been propagated sideways up to this point.
- If constant 0 (N-positions), then the following pattern will be periodic, since the computation now restarts as if it had started on a tape of only zeros

If the gadget is out of phase

- The least row that corresponds to a finished computation congruent to 0 modulo k will be constant, since no information has been propagated sideways up to this point.
- If constant 0 (N-positions), then the following pattern will be periodic, since the computation now restarts as if it had started on a tape of only zeros
- If constant 1 (P-positions), then the following behavior will be the same as if the gadget was in phase, since the computation now starts from a row of $\cdots 000111 \cdots$

Conclusion

- We have shown that a game consisting of two main heaps and a "gadget" can emulate any one-dimensional cellular automaton, and thereby any Turing machine.

Conclusion

- We have shown that a game consisting of two main heaps and a "gadget" can emulate any one-dimensional cellular automaton, and thereby any Turing machine.
- We can program a Turing machine so that it halts if and only if a certain finite sequence of 0s and 1s appears on its tape.

Conclusion

- We have shown that a game consisting of two main heaps and a "gadget" can emulate any one-dimensional cellular automaton, and thereby any Turing machine.
- We can program a Turing machine so that it halts if and only if a certain finite sequence of 0s and 1s appears on its tape.
- The halting problem for a Turing machine is undecidable.

Conclusion

- We have shown that a game consisting of two main heaps and a "gadget" can emulate any one-dimensional cellular automaton, and thereby any Turing machine.
- We can program a Turing machine so that it halts if and only if a certain finite sequence of 0s and 1s appears on its tape.
- The halting problem for a Turing machine is undecidable.
- Hence it is undecidable whether a given finite pattern of P-position occurs in our invariant heap game.

Game equivalence?

- The difference of any two P-positions (equal Grundy values) in a game is impossible as a move in this game (in a disjunctive sum).

Game equivalence?

- The difference of any two P-positions (equal Grundy values) in a game is impossible as a move in this game (in a disjunctive sum).
- Suppose that we adjoin a certain move to given game. Is it decidable whether it connects any two P-positions (equal Grundy values) in the original game?

Game equivalence?

- The difference of any two P-positions (equal Grundy values) in a game is impossible as a move in this game (in a disjunctive sum).
- Suppose that we adjoin a certain move to given game. Is it decidable whether it connects any two P-positions (equal Grundy values) in the original game?
- Two distinct P-positions constitute a finite pattern...

Further questions?

- How many heaps are required for undecidability? (strictly speaking we didn't prove that *any* finite number of heaps leads to undecidability).

Further questions?

- How many heaps are required for undecidability? (strictly speaking we didn't prove that *any* finite number of heaps leads to undecidability).
- We guess that three heaps suffice, and perhaps even two.

Further questions?

- How many heaps are required for undecidability? (strictly speaking we didn't prove that *any* finite number of heaps leads to undecidability).
- We guess that three heaps suffice, and perhaps even two.
- Do we need to be able to add tokens to heaps in order to achieve undecidability?