Games and Automata for Verification

Christof Löding

RWTH Aachen, Germany

GAMES 2009

September 14, Udine, Italy

In this tutorial

Three fundamental models of automata and games and their use in verification:

- Automata on infinite words (ω-automata)
- Parity games
- Automata on infinite trees (tree automata)

In this tutorial

Three fundamental models of automata and games and their use in verification:

- Automata on infinite words (ω-automata)
- Parity games
- Automata on infinite trees (tree automata)

The rough picture:

model checking:
system $\models$ specification?

In this tutorial

Three fundamental models of automata and games and their use in verification:

- Automata on infinite words (ω-automata)
- Parity games
- Automata on infinite trees (tree automata)

The rough picture:

model checking:

system $\models$ specification?

LTL

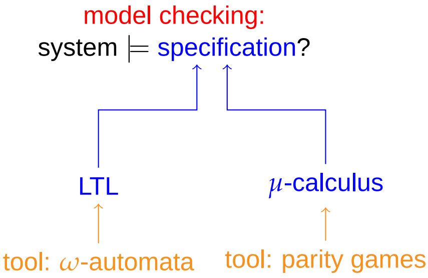
tool: ω-automata

In this tutorial

Three fundamental models of automata and games and their use in verification:

- Automata on infinite words (ω-automata)
- Parity games
- Automata on infinite trees (tree automata)

The rough picture:



In this tutorial

Three fundamental models of automata and games and their use in verification:

- Automata on infinite words (ω-automata)
- Parity games
- Automata on infinite trees (tree automata)

The rough picture:

model checking:

system $\models$ specification?

LTL

tool: ω-automata

μ-calculus

tool: parity games

satisfiability / synthesis:

$\exists$ system $\models$ specification?

In this tutorial

Three fundamental models of automata and games and their use in verification:

- Automata on infinite words (ω-automata)
- Parity games
- Automata on infinite trees (tree automata)

The rough picture:

model checking:

system $\models$ specification?

LTL

tool: ω-automata

μ-calculus

tool: parity games

satisfiability / synthesis:

$\exists$ system $\models$ specification?

LTL, S1S, μ-calculus,...

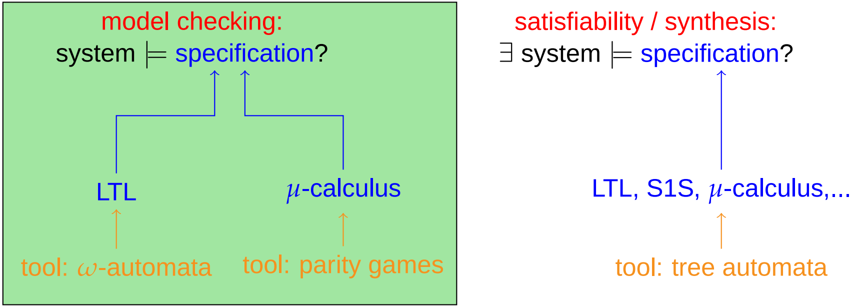
tool: tree automata

In this tutorial

Three fundamental models of automata and games and their use in verification:

- Automata on infinite words (ω-automata)
- Parity games
- Automata on infinite trees (tree automata)

The rough picture:



1 Introduction: model checking

2 Sequential specifications

- Linear temporal logic
- Automata on infinite words

3 Branching specifications

- Modal μ-calculus
- Parity games

4 Satisfiability and synthesis

- Synthesis problem
- Automata on infinite trees

Example – simple MUX protocol

Process 0: Repeat	Process 1: Repeat
0 non-critical section	0 non-critical section
1 wait unless turn = 0	1 wait unless turn = 1
2 critical section	2 critical section
3 turn := 1	3 turn := 0

Example – simple MUX protocol

Process 0: Repeat	Process 1: Repeat
0 non-critical section	0 non-critical section
1 wait unless turn = 0	1 wait unless turn = 1
2 critical section	2 critical section
3 turn := 1	3 turn := 0

Typical questions:

- Can the two processes reach their critical section at the same time?
- Does a process that wants to enter the critical section eventually succeed?

Example – simple MUX protocol

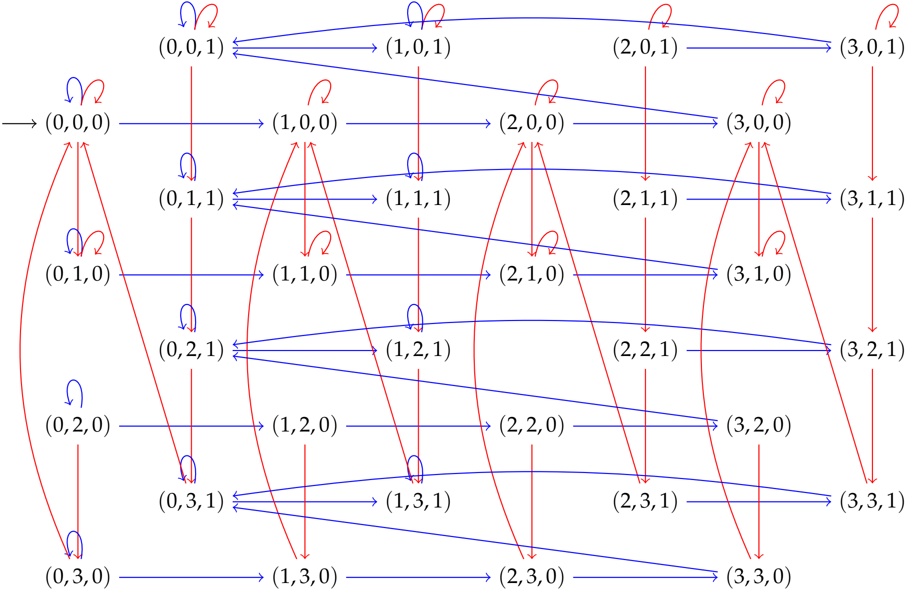
Process 0: Repeat	Process 1: Repeat
0 non-critical section	0 non-critical section
1 wait unless turn = 0	1 wait unless turn = 1
2 critical section	2 critical section
3 turn := 1	3 turn := 0

Typical questions:

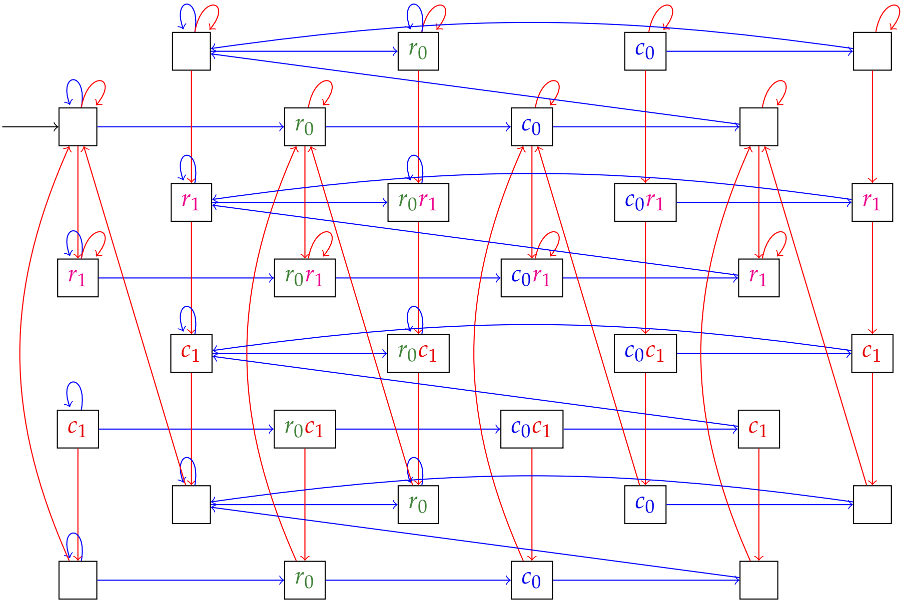
- Can the two processes reach their critical section at the same time?
- Does a process that wants to enter the critical section eventually succeed?

To answer such questions we can analyze the transition graph generated by the protocol.

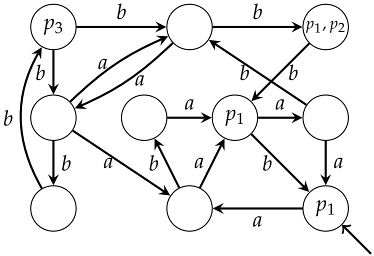
MUX protocol as transitions graph



Abstract transitions system



Transition systems – definition



propositions — labeling $\lambda : S \to 2^P$

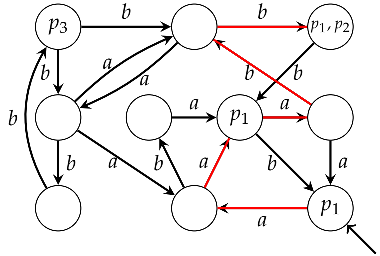
initial state

$$T = (S, P, A, s_{in}, R, \lambda)$$

actions

states — edges $R \subseteq S \times A \times S$

Transition systems – definition



propositions labeling $\lambda : S \to 2^P$

initial state

$$T = (S, P, A, s_{in}, R, \lambda)$$

actions

states edges $R \subseteq S \times A \times S$

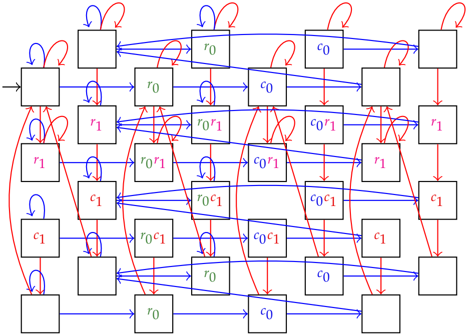
An **execution** (or **computation** or **trace**) of T is an infinite sequence from $(2^P \times A)^\omega$:

$$\{p_1\} \xrightarrow{a} \emptyset \xrightarrow{a} \{p_1\} \xrightarrow{a} \emptyset \xrightarrow{\quad b \quad} \emptyset \xrightarrow{b} \{p_1, p_2\} \quad \cdots$$

Model checking problem

Given: Transition system T, specification φ

Question: $T \models \varphi$?



$\models$ never both processes in critical section?

1. Introduction: model checking

2. Sequential specifications
   - Linear temporal logic
   - Automata on infinite words

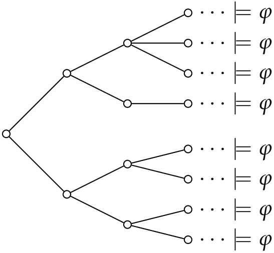
3. Branching specifications
   - Modal μ-calculus
   - Parity games

4. Satisfiability and synthesis
   - Synthesis problem
   - Automata on infinite trees

Sequential specifications

- Properties of single executions of the system, i.e., of infinite sequences of propositions and actions
- A sequential specification defines a set of such infinite sequences
- A system satisfies the specification if all possible executions do

Tree of executions



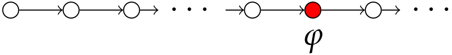
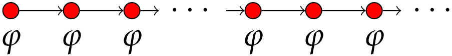
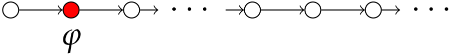
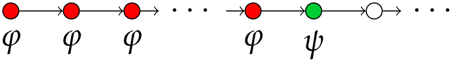
Linear temporal logic

Linear temporal logic – LTL

LTL formulas: Build over a set $P = \{p_1, \ldots, p_n\}$ of atomic propositions (we assume that actions are encoded by propositions in the computations)

Models: Infinite sequences of vectors of size n. Entry i of a vector codes truth value of p_i ($1 =$ true, $0 =$ false).

- Atomic formulas: p_i (p_i is true in the first position)
- Boolean combinations
- Temporal operators:

$\mathbf{F}\varphi$		"eventually φ"
$\mathbf{G}\varphi$		"always φ"
$\mathbf{X}\varphi$		"next φ"
$\varphi\mathbf{U}\psi$		"φ until ψ"

Semantics by example

$p_1 \wedge X \neg p_2$

$$\begin{pmatrix} \color{red}{1} \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ \color{red}{0} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} \cdots$$

Semantics by example

$p_1 \wedge X\neg p_2$

$$\begin{pmatrix}1\\1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}\begin{pmatrix}0\\0\end{pmatrix}\cdots$$

$\mathsf{G}p_2 \wedge \mathsf{F}p_1$

$$\begin{pmatrix}0\\1\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}\cdots\begin{pmatrix}0\\1\end{pmatrix}\begin{pmatrix}1\\1\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}\begin{pmatrix}1\\1\end{pmatrix}\cdots$$

Semantics by example

$p_1 \wedge X \neg p_2$

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} \cdots$$

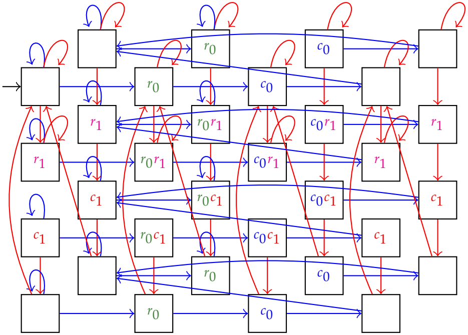
$\mathsf{G} p_2 \wedge \mathsf{F} p_1$

$$\begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \cdots \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \cdots$$

$\mathsf{F}(p_3 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1))$

$$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \cdots \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \cdots$$

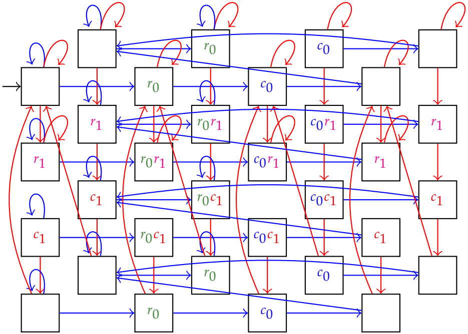
Examples for the MUX protocol



$P = \{c_0, c_1, r_0, r_1\}$, $A = \{blue, red\}$

- Safety: $\neg F(c_0 \wedge c_1)$

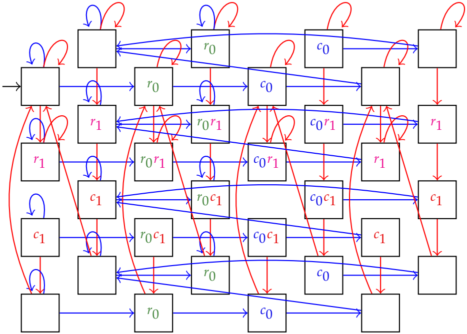
Examples for the MUX protocol



$P = \{c_0, c_1, r_0, r_1\}$, $A = \{blue, red\}$

- Safety: $\neg F(c_0 \wedge c_1)$
- Request-Response: $G(r_0 \longrightarrow F c_0)$

Examples for the MUX protocol



$P = \{c_0, c_1, r_0, r_1\}$, $A = \{blue, red\}$

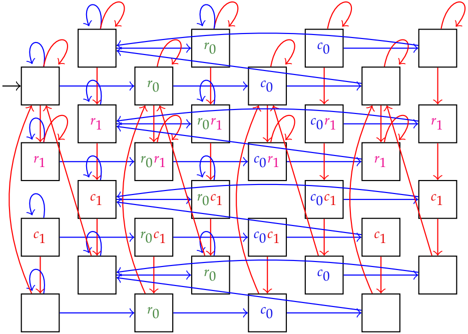
- Safety: $\neg F(c_0 \wedge c_1)$
- Request-Response: $G(r_0 \longrightarrow Fc_0)$
- Restriction to fair paths: $(GFblue) \wedge (GFred) \longrightarrow G(r_0 \longrightarrow Fc_0)$

Model checking problem for LTL

Given: Transition system T, LTL formula φ

Question: $T \models \varphi$?

Does φ hold on every path through T?



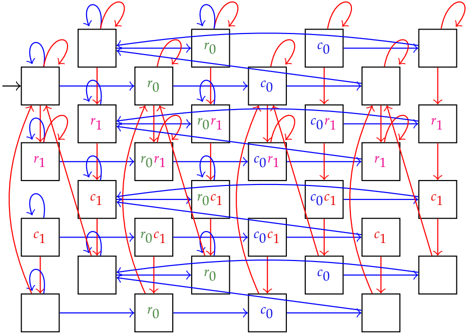
$\models (GF\mathit{blue}) \wedge (GF\mathit{red}) \rightarrow G(r_0 \rightarrow Fc_0)$?

Model checking problem for LTL

Given: Transition system T, LTL formula φ

Question: $T \models \varphi$?

Does φ hold on every path through T?



$\models (GF\textit{blue}) \wedge (GF\textit{red}) \rightarrow G(r_0 \rightarrow Fc_0)$?

transform the formula into an object closer to transition systems

Automata on infinite words

Büchi automata

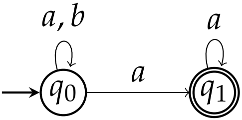
An ω-**automaton** is of the form $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, Acc)$, where $Q, \Sigma, q_{in}, \Delta$ are as for standard finite automata, and Acc defines the acceptance condition.

Büchi automata: Acc given as set $F \subseteq Q$ of final states.

A run is accepting if it contains infinitely often a state from F

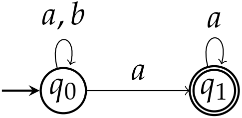
Examples

- $\Sigma = \{a, b\}$
  A nondeterministic Büchi automaton for "finitely many b":

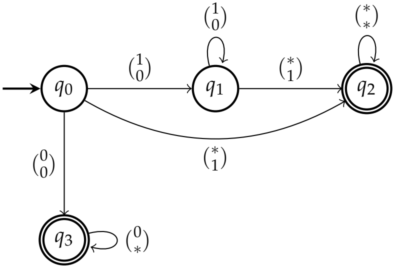


Examples

- $\Sigma = \{a, b\}$
  A nondeterministic Büchi automaton for "finitely many b":



- $\Sigma = \{0, 1\}^2$, LTL formula $(p_1 \mathsf{U} p_2) \vee \mathsf{G} \neg p_1$



From LTL to automata

Theorem (Vardi/Wolper'86)

For each LTL formula φ one can construct an equivalent Büchi automaton $\mathcal{A}_\varphi$ with $\mathcal{O}(2^{|\varphi|})$ states.

From LTL to automata – construction

Büchi automaton “guesses” valuations of the subformulas and verifies its guesses:

- Atomic formulas, Boolean combinations: verified directly
- Operators X, G: verified using the transitions
- Operators F, U: verified by acceptance condition

From LTL to automata – construction

Büchi automaton "guesses" valuations of the subformulas and verifies its guesses:

- Atomic formulas, Boolean combinations: verified directly
- Operators X, G: verified using the transitions
- Operators F, U: verified by acceptance condition

$\alpha =$	$\binom{1}{0}$	$\binom{0}{1}$	$\binom{1}{1}$	$\binom{0}{0}$	$\binom{1}{0}$	$\binom{0}{1}$	$\cdots$
$\neg p_1$							
$\neg p_2$							
$\neg p_2 \mathsf{U} p_1$							
$\mathsf{X}(\neg p_2 \mathsf{U} p_1)$							
$\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1)$							
$\mathsf{F}(\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1))$							

From LTL to automata – construction

Büchi automaton “guesses” valuations of the subformulas and verifies its guesses:

- Atomic formulas, Boolean combinations: verified directly
- Operators X, G: verified using the transitions
- Operators F, U: verified by acceptance condition

$\alpha =$	$\binom{1}{0}$	$\binom{0}{1}$	$\binom{1}{1}$	$\binom{0}{0}$	$\binom{1}{0}$	$\binom{0}{1}$	$\cdots$
$\neg p_1$	0						
$\neg p_2$	1						
$\neg p_2 \mathsf{U} p_1$	1						
$\mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0						
$\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0						
$\mathsf{F}(\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1))$	1						

From LTL to automata – construction

Büchi automaton "guesses" valuations of the subformulas and verifies its guesses:

- Atomic formulas, Boolean combinations: verified directly
- Operators X, G: verified using the transitions
- Operators F, U: verified by acceptance condition

$\alpha =$	$\binom{1}{0}$	$\binom{0}{1}$	$\binom{1}{1}$	$\binom{0}{0}$	$\binom{1}{0}$	$\binom{0}{1}$	$\cdots$
$\neg p_1$	0	1					
$\neg p_2$	1	0					
$\neg p_2 \mathsf{U} p_1$	1	0					
$\mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1					
$\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1					
$\mathsf{F}(\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1))$	1	1					

From LTL to automata – construction

Büchi automaton “guesses” valuations of the subformulas and verifies its guesses:

- Atomic formulas, Boolean combinations: verified directly
- Operators X, G: verified using the transitions
- Operators F, U: verified by acceptance condition

$\alpha =$	$\binom{1}{0}$	$\binom{0}{1}$	$\binom{1}{1}$	$\binom{0}{0}$	$\binom{1}{0}$	$\binom{0}{1}$	$\cdots$
$\neg p_1$	0	1	0				
$\neg p_2$	1	0	0				
$\neg p_2 \mathsf{U} p_1$	1	0	1				
$\mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1	1				
$\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1	0				
$\mathsf{F}(\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1))$	1	1	1				

From LTL to automata – construction

Büchi automaton “guesses” valuations of the subformulas and verifies its guesses:

- Atomic formulas, Boolean combinations: verified directly
- Operators X, G: verified using the transitions
- Operators F, U: verified by acceptance condition

$\alpha =$	$\binom{1}{0}$	$\binom{0}{1}$	$\binom{1}{1}$	$\binom{0}{0}$	$\binom{1}{0}$	$\binom{0}{1}$	$\cdots$
$\neg p_1$	0	1	0	1			
$\neg p_2$	1	0	0	1			
$\neg p_2 \mathsf{U} p_1$	1	0	1	1			
$\mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1	1	1			
$\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1	0	1			
$\mathsf{F}(\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1))$	1	1	1	1			

From LTL to automata – construction

Büchi automaton “guesses” valuations of the subformulas and verifies its guesses:

- Atomic formulas, Boolean combinations: verified directly
- Operators X, G: verified using the transitions
- Operators F, U: verified by acceptance condition

$\alpha =$	$\binom{1}{0}$	$\binom{0}{1}$	$\binom{1}{1}$	$\binom{0}{0}$	$\binom{1}{0}$	$\binom{0}{1}$	$\cdots$
$\neg p_1$	0	1	0	1	0		
$\neg p_2$	1	0	0	1	1		
$\neg p_2 \mathsf{U} p_1$	1	0	1	1	1		
$\mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1	1	1	0		
$\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1	0	1	0		
$\mathsf{F}(\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1))$	1	1	1	1	$\cdot$		

From LTL to automata – construction

Büchi automaton "guesses" valuations of the subformulas and verifies its guesses:

- Atomic formulas, Boolean combinations: verified directly
- Operators X, G: verified using the transitions
- Operators F, U: verified by acceptance condition

$\alpha =$	$\binom{1}{0}$	$\binom{0}{1}$	$\binom{1}{1}$	$\binom{0}{0}$	$\binom{1}{0}$	$\binom{0}{1}$	$\cdots$
$\neg p_1$	0	1	0	1	0	1	$\cdots$
$\neg p_2$	1	0	0	1	1	0	$\cdots$
$\neg p_2 \mathsf{U} p_1$	1	0	1	1	1	0	$\cdots$
$\mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1	1	1	0	$\cdot$	$\cdots$
$\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1)$	0	1	0	1	0	$\cdot$	$\cdots$
$\mathsf{F}(\neg p_1 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1))$	1	1	1	1	$\cdot$	$\cdot$	$\cdots$

Model checking for LTL – solution

Given: Transition system T, LTL formula φ

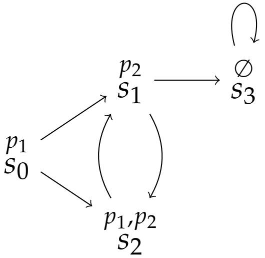
Question: $T \models \varphi$?

Does φ hold on every path through T?

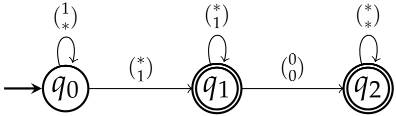
1. Transform $\neg\varphi$ into $\mathcal{A}_{\neg\varphi}$
2. Take the product of T and $\mathcal{A}_{\neg\varphi}$
   $\leadsto$ results in an "input free" Büchi automaton
3. Check if this "input free" Büchi automaton has an accepting path (a reachable loop through a final state)
4. Complexity: exponential in $|\varphi|$ and linear in $|T|$

Illustration

T:

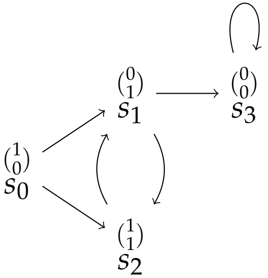


$\mathcal{A}_{\neg\varphi}$:

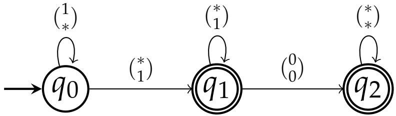


Illustration

T:

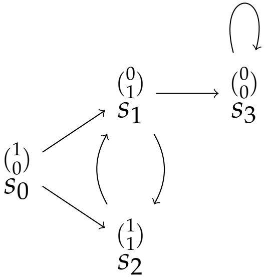


$\mathcal{A}_{\neg\varphi}$:

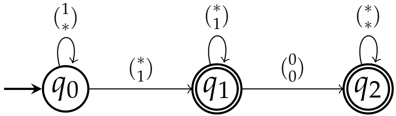


Illustration

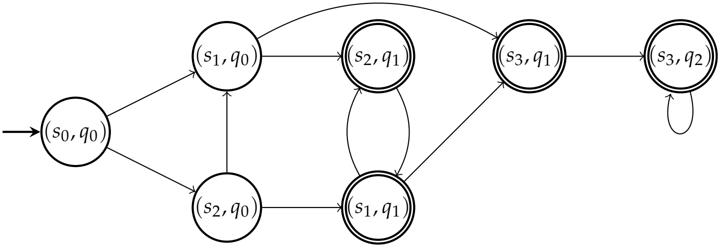
T:



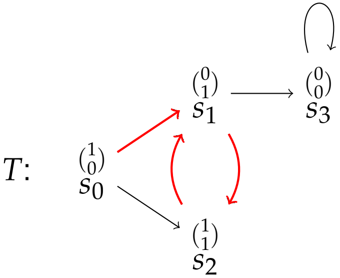
$\mathcal{A}_{\neg\varphi}$:

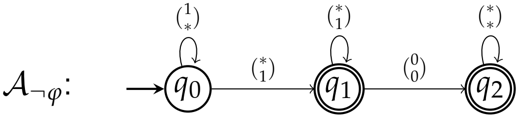


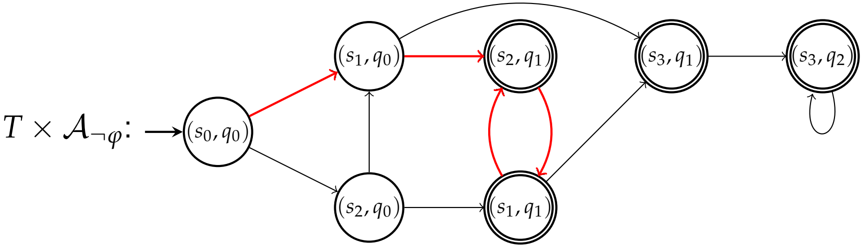
$T \times \mathcal{A}_{\neg\varphi}$:



Illustration







Summary of this section

- Model checking problem for LTL
- LTL formulas can be transformed into Büchi automata
- Using Büchi automata the model checking problem for LTL can be reduced to a graph theoretic problem about the existence of certain loops

Summary of this section

- Model checking problem for LTL
- LTL formulas can be transformed into Büchi automata
- Using Büchi automata the model checking problem for LTL can be reduced to a graph theoretic problem about the existence of certain loops

Further properties of Büchi automata

- Good closure properties: union, intersection, complement, projection
- Same expressive power as S1S (extension of first-order logic over $(\mathbb{N}, +1)$ with quantification over sets)
- Determinization into deterministic Muller or parity automata (but not into deterministic Büchi automata)

Branching specifications

- Some properties of a system cannot be expressed as properties of single executions:
  - Consider a system where the actions correspond to user inputs (e.g. a vending machine for train tickets).
  - A typical non-sequential property would be: As long as the ticket has not been paid, it should be possible to reset the session.
- For such specifications we need logics that consider the system as a whole.

Modal logic

A basic formalism to talk about branching in transition systems is **modal logic**.

Formulas are evaluated in a state s of the transition system:

- Atomic propositions and boolean combinations as usual
- $\Diamond_a \varphi$: there is an a-successor of s where φ holds
  $\Diamond \varphi := \bigvee_{a \in A} \varphi$
- $\Box_a \varphi$: φ holds at all a-successors of s
  $\Box \varphi := \bigwedge_{a \in A} \varphi$

A formula defines the set of states in which it evaluates to true.

Modal logic

A basic formalism to talk about branching in transition systems is **modal logic**.

Formulas are evaluated in a state s of the transition system:

- Atomic propositions and boolean combinations as usual
- $\Diamond_a \varphi$: there is an a-successor of s where φ holds
  $\Diamond\varphi := \bigvee_{a \in A} \varphi$
- $\Box_a \varphi$: φ holds at all a-successors of s
  $\Box\varphi := \bigwedge_{a \in A} \varphi$

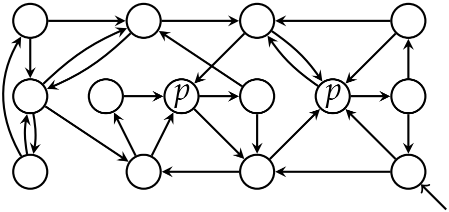
A formula defines the set of states in which it evaluates to true.

Modal logic is not very powerful: each formula can navigate only a bounded number of steps in the transition system.

Modal μ-calculus

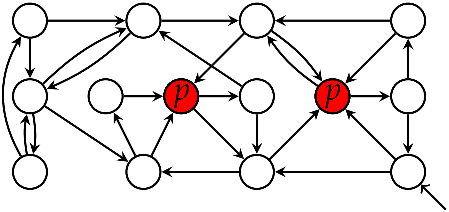
Modal μ-calculus – example

Example property: Each path (from the initial state) eventually reaches a state where p holds



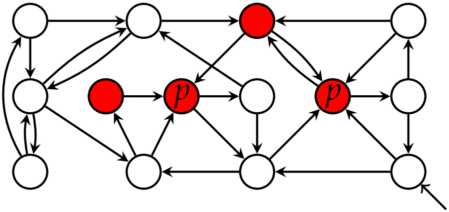
Modal μ-calculus – example

Example property: Each path (from the initial state) eventually reaches a state where p holds



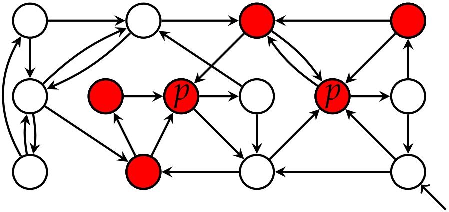
Modal μ-calculus – example

Example property: Each path (from the initial state) eventually reaches a state where p holds



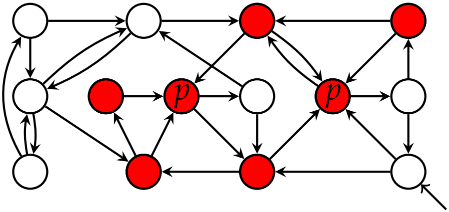
Modal μ-calculus – example

Example property: Each path (from the initial state) eventually reaches a state where p holds



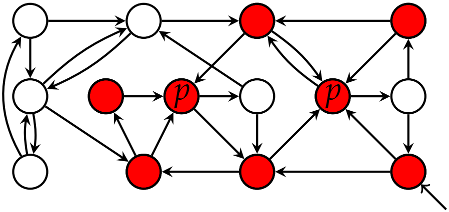
Modal μ-calculus – example

Example property: Each path (from the initial state) eventually reaches a state where p holds



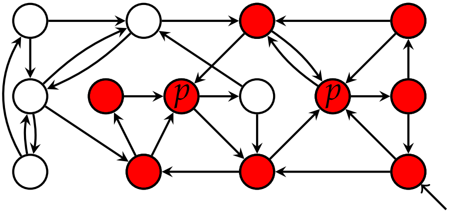
Modal μ-calculus – example

Example property: Each path (from the initial state) eventually reaches a state where p holds



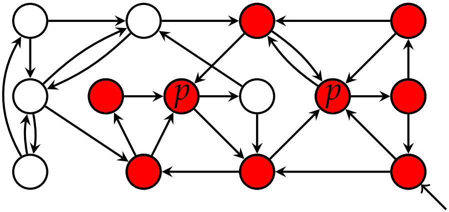
Modal μ-calculus – example

Example property: Each path (from the initial state) eventually reaches a state where p holds



Modal μ-calculus – example

Example property: Each path (from the initial state) eventually reaches a state where p holds

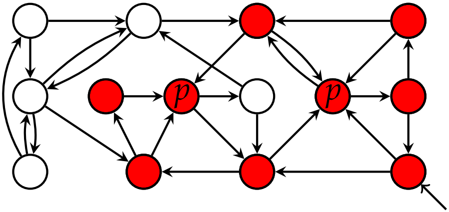


Is the initial state in the smallest set X that

- contains all p-states and
- contains t if all successors of t are in X?

Modal μ-calculus – example

Example property: Each path (from the initial state) eventually reaches a state where p holds



Is the initial state in the smallest set X that

- contains all p-states and
- contains t if all successors of t are in X?

In the modal μ-calculus: $T \models \mu X.(p \vee \Box X)$

Fixpoint formulas

The **modal μ-calculus** L_μ extends modal logic by fixpoint formulas:

- We use variables $X, Y, \ldots$ denoting sets of states.
- For each formula $\varphi(X)$ where X occurs only positively, L_μ contains the formulas $\mu X.\varphi(X)$ and $\nu X.\varphi(X)$.

Fixpoint formulas

The **modal μ-calculus** L_μ extends modal logic by fixpoint formulas:

- We use variables $X, Y, \ldots$ denoting sets of states.
- For each formula $\varphi(X)$ where X occurs only positively, L_μ contains the formulas $\mu X.\varphi(X)$ and $\nu X.\varphi(X)$.

Semantics. Each formula φ together with an interpretation of the free variables denotes a set $[\![\varphi]\!]$ of states:

- atomic formulas, boolean combinations, $\Diamond_a \varphi$, and $\Box_a \varphi$ as for modal logic
- $[\![\mu X.\varphi(X)]\!] :=$ least set X such that $[\![\varphi(X)]\!] = X$
- $[\![\nu X.\varphi(X)]\!] :=$ greatest set X such that $[\![\varphi(X)]\!] = X$

Examples

- $\mu X.(p \vee \Box X)$: All states from which all path finally reach p.

Examples

- $\mu X.(p \vee \Box X)$: All states from which all path finally reach p.
- $\nu X.(\Diamond_a X)$: The biggest set X such that all states have an a-successor in X.

Examples

- $\mu X.(p \vee \Box X)$: All states from which all path finally reach p.
- $\nu X.(\Diamond_a X)$: The biggest set X such that all states have an a-successor in X.
  - States without a-successor cannot be in X.

Examples

- $\mu X.(p \vee \Box X)$: All states from which all path finally reach p.
- $\nu X.(\Diamond_a X)$: The biggest set X such that all states have an a-successor in X.
  - States without a-successor cannot be in X.
  - States without an a-successor that has an a-successor cannot be in X.
  - ...

Examples

- $\mu X.(p \vee \Box X)$: All states from which all path finally reach p.
- $\nu X.(\Diamond_a X)$: The biggest set X such that all states have an a-successor in X.
  - States without a-successor cannot be in X.
  - States without an a-successor that has an a-successor cannot be in X.
  - ...

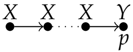
$\leadsto$ All states from which there is an infinite a-path.

Nested fixpoints

$\nu Y.\mu X.\Diamond((p \wedge Y) \vee X)$

Nested fixpoints

$\nu Y.\mu X.\Diamond((p \wedge Y) \vee X)$



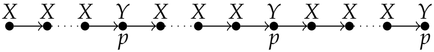
Nested fixpoints

$\nu Y.\mu X.\Diamond((p \wedge Y) \vee X)$

$$\begin{tikzcd}
\overset{X}{\bullet} \arrow[r] & \overset{X}{\bullet} \arrow[r, dotted, no head] & \overset{X}{\bullet} \arrow[r] & \underset{p}{\overset{Y}{\bullet}} \arrow[r] & \overset{X}{\bullet} \arrow[r, dotted, no head] & \overset{X}{\bullet} \arrow[r] & \overset{X}{\bullet} \arrow[r] & \underset{p}{\overset{Y}{\bullet}}
\end{tikzcd}$$

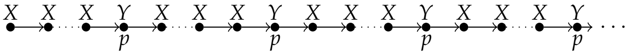
Nested fixpoints

$\nu Y.\mu X.\Diamond((p \wedge Y) \vee X)$



Nested fixpoints

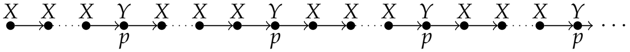
$\nu Y.\mu X.\Diamond((p \wedge Y) \vee X)$



$\rightsquigarrow$ a path with infinitely many p-states

Nested fixpoints

$\nu Y.\mu X.\Diamond((p \wedge Y) \vee X)$



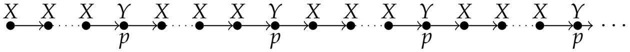
$\rightsquigarrow$ a path with infinitely many p-states

What happens when swapping the fixpoints?

$\mu X.\nu Y.\Diamond((p \wedge Y) \vee X)$

Nested fixpoints

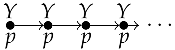
$\nu Y.\mu X.\Diamond((p \wedge Y) \vee X)$



$\rightsquigarrow$ a path with infinitely many p-states

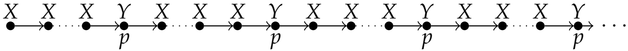
What happens when swapping the fixpoints?

$\mu X.\nu Y.\Diamond((p \wedge Y) \vee X)$



Nested fixpoints

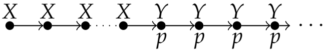
$\nu Y.\mu X.\Diamond((p \wedge Y) \vee X)$



$\rightsquigarrow$ a path with infinitely many p-states

What happens when swapping the fixpoints?

$\mu X.\nu Y.\Diamond((p \wedge Y) \vee X)$



$\rightsquigarrow$ a path which finally only has p-states

Significance of L_μ

- L_μ is rather powerful: it subsumes many popular logics like LTL, CTL, CTL*, PDL, ...
- Complexities of decision problems are reasonable compared to other logics
- Efficient algorithms for interesting fragments
- Disadvantage: Fixpoint formulas are hard to read

Model checking for L_μ

Let T be a transition system with initial state s_{in}, and φ an L_μ-formula.

We write $T \models \varphi$ if $s_{in} \in [\![\varphi]\!]$

The model checking problem is:

Given:	Transition system T, L_μ-formula φ
Question:	$T \models \varphi$?

We translate this problem to the problem of solving specific games.

Model checking game – idea

- The model checking game is played between VERIFIER (trying to prove $T \models \varphi$) and FALSIFIER (trying to prove $T \not\models \varphi$).
- A game position is a pair (s, ψ) consisting of a state of T and a subformula of φ.
- The player who moves and the options for the next move are determined by the structure of ψ.

Model checking game – idea

- The model checking game is played between VERIFIER (trying to prove $T \models \varphi$) and FALSIFIER (trying to prove $T \not\models \varphi$).
- A game position is a pair (s, ψ) consisting of a state of T and a subformula of φ.
- The player who moves and the options for the next move are determined by the structure of ψ. Examples:
  - For a conjunction $\psi = \psi_1 \wedge \psi_2$ FALSIFIER can choose one of the ψ_i and move to (s, ψ_i)

$$
\begin{array}{ccc}
 & \nearrow & (s, \psi_1) \\
(s, \psi_1 \wedge \psi_2) & & \\
 & \searrow & (s, \psi_2)
\end{array}
$$

Model checking game – idea

- The model checking game is played between VERIFIER (trying to prove $T \models \varphi$) and FALSIFIER (trying to prove $T \not\models \varphi$).
- A game position is a pair (s, ψ) consisting of a state of T and a subformula of φ.
- The player who moves and the options for the next move are determined by the structure of ψ. Examples:
  - For a conjunction $\psi = \psi_1 \wedge \psi_2$ FALSIFIER can choose one of the ψ_i and move to (s, ψ_i)

$$\begin{tikzcd}
 & (s,\psi_1) \\
(s,\psi_1 \wedge \psi_2) \arrow[ur] \arrow[dr] & \\
 & (s,\psi_2)
\end{tikzcd}$$

  - For $\psi = \Diamond_a \psi'$ VERIFIER can choose an a-successor s' of s and move to (s', ψ')

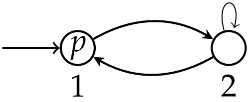
$$\begin{tikzcd}
 & (s_1,\psi') \\
(s,\Diamond_a \psi') \arrow[ur] \arrow[dr] & \vdots \\
 & (s_n,\psi')
\end{tikzcd}
\quad \text{for} \quad
\begin{tikzcd}
 & s_1 \\
s \arrow[ur, "a"] \arrow[dr, "a"'] & \vdots \\
 & s_n
\end{tikzcd}
\quad \text{in } T$$

- The fixpoints are captured by the winning condition.

Model checking game – example

$$\varphi = \nu Y.\, \underbrace{\mu X. \Diamond(\overbrace{(p \wedge Y) \vee X}^{\theta})}_{\psi}$$

$T :$

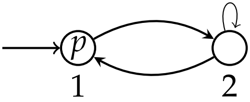


VERIFIER vs. FALSIFIER:

Model checking game – example

$$\varphi = \nu Y.\, \underbrace{\mu X.\Diamond(\overbrace{(p \wedge Y) \vee X}^{\theta})}_{\psi}$$

T :



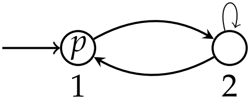
Verifier vs. Falsifier:

$(1, \varphi)$

Model checking game – example

$$\varphi = \nu Y.\, \underbrace{\mu X.\Diamond(\overbrace{(p \wedge Y) \vee X}^{\theta})}_{\psi}$$

$T:$



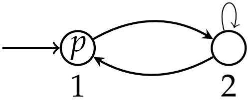
VERIFIER vs. FALSIFIER:

$$(1, \varphi) \longrightarrow (1, \psi)$$

Model checking game – example

$$\varphi = \nu Y.\,\underbrace{\mu X.\Diamond\big(\overbrace{(p \wedge Y) \vee X}^{\theta}\big)}_{\psi}$$

$T:$



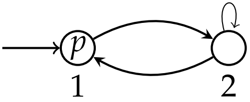
VERIFIER vs. FALSIFIER:

$$(1,\varphi) \longrightarrow (1,\psi) \longrightarrow (1,\Diamond\theta)$$

Model checking game – example

$$\varphi = \nu Y.\, \underbrace{\mu X.\Diamond(\overbrace{(p \wedge Y) \vee X}^{\theta})}_{\psi}$$

$T:$

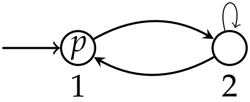


VERIFIER vs. FALSIFIER:

$$(1, \varphi) \longrightarrow (1, \psi) \longrightarrow (1, \Diamond\theta) \longrightarrow (2, \theta)$$

Model checking game – example

$$\varphi = \nu Y.\, \underbrace{\mu X. \Diamond(\overbrace{(p \wedge Y) \vee X}^{\theta})}_{\psi}$$

$T :$ 

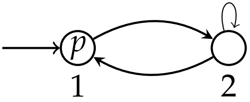
VERIFIER vs. FALSIFIER:

$$\begin{tikzcd}
(1,\varphi) \arrow[r] & (1,\psi) \arrow[r] & (1,\Diamond\theta) \arrow[r] & (2,\theta) \arrow[r] \arrow[d] & (2, p \wedge Y) \\
 & & & (2, X) &
\end{tikzcd}$$

Model checking game – example

$$\varphi = \nu Y.\, \underbrace{\mu X.\Diamond(\overbrace{(p \wedge Y) \vee X}^{\theta})}_{\psi}$$

$T:$



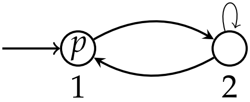
Verifier vs. Falsifier:

$$\begin{tikzcd}
(1,\varphi) \arrow[r] & (1,\psi) \arrow[r] & (1,\Diamond\theta) \arrow[r] & (2,\theta) \arrow[r] \arrow[d] & (2, p \wedge Y) \arrow[r] \arrow[d] & (2,p) \\
 & & & (2,X) & (2,Y) &
\end{tikzcd}$$

Model checking game – example

$$\varphi = \nu Y.\,\underbrace{\mu X.\Diamond(\overbrace{(p \wedge Y) \vee X}^{\theta})}_{\psi}$$

$T :$



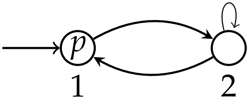
VERIFIER vs. FALSIFIER:

$$\begin{tikzcd}
(1,\varphi) \arrow[r] & (1,\psi) \arrow[r] & (1,\Diamond\theta) \arrow[r] & (2,\theta) \arrow[r] \arrow[d] & (2, p \wedge Y) \arrow[r] \arrow[d] & (2,p) \\
 & & & (2,X) & (2,Y) \arrow[d] & \\
 & & & & (2,\varphi) &
\end{tikzcd}$$

Model checking game – example

$$\varphi = \nu Y.\, \underbrace{\mu X. \Diamond\overbrace{((p \wedge Y) \vee X)}^{\theta}}_{\psi}$$

$T:$



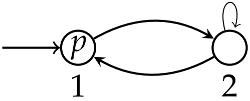
VERIFIER vs. FALSIFIER:

$$\begin{tikzcd}
(1,\varphi) \arrow[r] & (1,\psi) \arrow[r] & (1,\Diamond\theta) \arrow[r] & (2,\theta) \arrow[r] \arrow[d] & (2, p \wedge Y) \arrow[r] \arrow[d] & (2,p) \\
 & & & (2,X) \arrow[d] & (2,Y) \arrow[d] & \\
 & & & (2,\psi) & (2,\varphi) \arrow[l] &
\end{tikzcd}$$

Model checking game – example

$$\varphi = \nu Y.\, \underbrace{\mu X. \Diamond\big(\overbrace{(p \wedge Y) \vee X}^{\theta}\big)}_{\psi}$$

T :

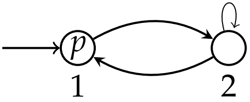


Verifier vs. Falsifier:

$$\begin{tikzcd}
(1,\varphi) \arrow[r] & (1,\psi) \arrow[r] & (1,\Diamond\theta) \arrow[r] & (2,\theta) \arrow[r] \arrow[d] & (2,p\wedge Y) \arrow[r] \arrow[d] & (2,p) \\
 & & & (2,X) \arrow[d] & (2,Y) \arrow[d] & \\
 & & (2,\Diamond\theta) & (2,\psi) \arrow[l] & (2,\varphi) \arrow[l] &
\end{tikzcd}$$

Model checking game – example

$$\varphi = \nu Y.\, \underbrace{\mu X.\Diamond(\overbrace{(p \wedge Y) \vee X}^{\theta})}_{\psi}$$

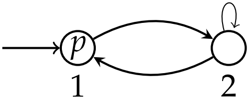
$T :$ 

VERIFIER vs. FALSIFIER:

$$\begin{tikzcd}
(1,\varphi) \arrow[r] & (1,\psi) \arrow[r] & (1,\Diamond\theta) \arrow[r] & (2,\theta) \arrow[r] \arrow[d] & (2,p \wedge Y) \arrow[r] \arrow[d] & (2,p) \\
 & & & (2,X) \arrow[d] & (2,Y) \arrow[d] & \\
 & (1,\theta) & (2,\Diamond\theta) \arrow[l] \arrow[uur] & (2,\psi) \arrow[l] & (2,\varphi) \arrow[l] &
\end{tikzcd}$$

Model checking game – example

$$\varphi = \nu Y.\,\underbrace{\mu X.\Diamond(\overbrace{(p \wedge Y) \vee X}^{\theta})}_{\psi}$$

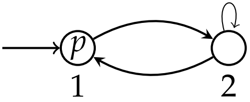
$T :$ 

Verifier vs. Falsifier:

$$\begin{tikzcd}
 & (1,\varphi) \arrow[r] & (1,\psi) \arrow[r] & (1,\Diamond\theta) \arrow[r] & (2,\theta) \arrow[r] \arrow[d] & (2,p\wedge Y) \arrow[r] \arrow[d] & (2,p) \\
 & (1,Y) \arrow[u] & (1,X) \arrow[u] & & (2,X) \arrow[d] & (2,Y) \arrow[d] & \\
(1,p) & (1,p\wedge Y) \arrow[l] \arrow[u] & (1,\theta) \arrow[l] \arrow[u] & (2,\Diamond\theta) \arrow[l] \arrow[uur] & (2,\psi) \arrow[l] & (2,\varphi) \arrow[l] &
\end{tikzcd}$$

Model checking game – example

$$\varphi = \nu Y.\, \underbrace{\mu X.\Diamond(\overbrace{(p \wedge Y) \vee X}^{\theta})}_{\psi}$$

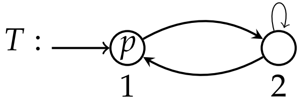
$T:$ 

VERIFIER vs. FALSIFIER:

$$\begin{tikzcd}
 & (1,\varphi) \arrow[r] & (1,\psi) \arrow[r] & {\color{orange}(1,\Diamond\theta)} \arrow[r] & {\color{orange}(2,\theta)} \arrow[r] \arrow[d] & {\color{purple}(2,p\wedge Y)} \arrow[r] \arrow[d] & (2,p) \\
\text{{\color{orange}VERIFIER} wins} \arrow[d, dashed] & (1,Y) \arrow[u] & (1,X) \arrow[u] & & (2,X) \arrow[d] & (2,Y) \arrow[d] & \text{{\color{purple}FALSIFIER} wins} \arrow[u, dashed] \\
(1,p) & {\color{purple}(1,p\wedge Y)} \arrow[l] \arrow[u] & {\color{orange}(1,\theta)} \arrow[l] \arrow[u] & {\color{orange}(2,\Diamond\theta)} \arrow[l] \arrow[uur] & (2,\psi) \arrow[l] & (2,\varphi) \arrow[l] &
\end{tikzcd}$$

Model checking game – example

$$\varphi = \nu Y.\underbrace{\mu X.\Diamond(\overbrace{(p \wedge Y) \vee X}^{\theta})}_{\psi}$$



VERIFIER vs. FALSIFIER:

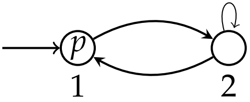
$$\begin{tikzcd}
 & (1,\varphi) \arrow[r] & (1,\psi) \arrow[r] & (1,\Diamond\theta) \arrow[r] & (2,\theta) \arrow[r] \arrow[d] & (2,p\wedge Y) \arrow[r] \arrow[d] & (2,p) \\
\text{VERIFIER wins} \arrow[d, dashed] & (1,Y)^{++} \arrow[u] & (1,X)^{-} \arrow[u] & & (2,X)^{-} \arrow[d] & (2,Y)^{++} \arrow[d] & \text{FALSIFIER wins} \arrow[u, dashed] \\
(1,p) & (1,p\wedge Y) \arrow[l] \arrow[u] & (1,\theta) \arrow[l] \arrow[u] & (2,\Diamond\theta) \arrow[l] \arrow[uur] & (2,\psi) \arrow[l] & (2,\varphi) \arrow[l] &
\end{tikzcd}$$

VERIFIER wins an infinite play if the outermost fixpoint variable that is visited infinitely often is a greatest fixpoint.

Model checking game – example

$$\varphi = \nu Y.\, \underbrace{\mu X. \Diamond(\overbrace{(p \wedge Y) \vee X}^{\theta})}_{\psi}$$

$T :$



VERIFIER vs. FALSIFIER:

$$\begin{tikzcd}
 & (1,\varphi) \arrow[r] & (1,\psi) \arrow[r] & (1,\Diamond\theta) \arrow[r] & (2,\theta) \arrow[r] \arrow[d] & (2,p\wedge Y) \arrow[r] \arrow[d] & (2,p) \\
\text{VERIFIER wins} \arrow[d, dashed] & (1,Y)^{++} \arrow[u] & (1,X)^{-} \arrow[u] & & (2,X)^{-} \arrow[d] & (2,Y)^{++} \arrow[d] & \text{FALSIFIER wins} \arrow[u, dashed] \\
(1,p) & (1,p\wedge Y) \arrow[l] \arrow[u] & (1,\theta) \arrow[l] \arrow[u] & (2,\Diamond\theta) \arrow[l] \arrow[uur] & (2,\psi) \arrow[l] & (2,\varphi) \arrow[l] &
\end{tikzcd}$$

VERIFIER wins an infinite play if the outermost fixpoint variable that is visited infinitely often is a greatest fixpoint.

Model checking game – summary

- Two players
- Game graph: each position belong to one of the players (if there is no choice the position can belong to any player)
- Terminal vertices: the winner is directly specified
- Infinite plays:
  - There are “good” events (greatest fixpoints) and “bad” events (least fixpoints) which are ordered hierarchically
  - VERIFIER wins a play if the most important event that appears infinitely often is good

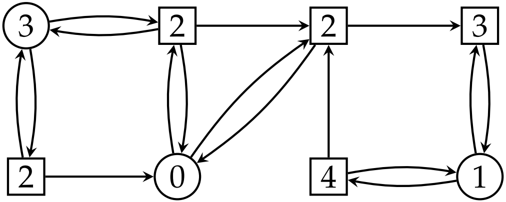
Theorem (Emerson/Jutla/Sistla’93, Stirling’95)

VERIFIER *has a winning strategy in the model checking game for* T *and* φ *iff* $T \models \varphi$.

Parity games

Parity games

- Call players Eva (VERIFIER) and Adam (FALSIFIER).
- Vertices of Eva are circles, those of Adam squares (we assume that there are no terminal vertices)
- Good and bad hierarchical events: vertices are assigned numbers
  - Even number = good event
  - Odd number = bad event
  - Hierarchy: natural order on numbers
- Eva wins an infinite play if the highest number that appears infinitely often is even.



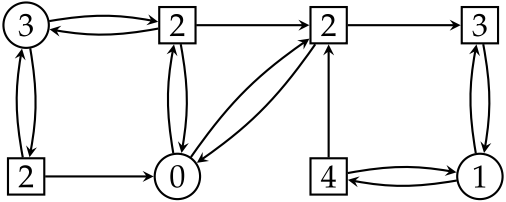
Parity games – notations

Game graph (or arena) $G = (V, V_{\mathsf{E}}, V_{\mathsf{A}}, E, c)$ with $c : V \rightarrow \{0, \ldots, k\}$ for some k.

A **strategy** σ for Eva is a mapping $\sigma : V^* V_{\mathsf{E}} \rightarrow V$ assigning a next move to finite plays ending in a vertex of Eva.

σ is a **winning strategy** for Eva if she wins all plays that she plays according to σ.

As we will see, for parity games special strategies that do not take the history of the play into account are sufficient: **Positional strategies** are of the form $\sigma : V_{\mathsf{E}} \rightarrow V$.



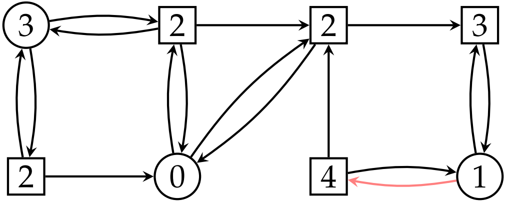
Parity games – notations

Game graph (or arena) $G = (V, V_{\mathsf{E}}, V_{\mathsf{A}}, E, c)$ with $c : V \to \{0, \ldots, k\}$ for some k.

A **strategy** σ for Eva is a mapping $\sigma : V^* V_{\mathsf{E}} \to V$ assigning a next move to finite plays ending in a vertex of Eva.

σ is a **winning strategy** for Eva if she wins all plays that she plays according to σ.

As we will see, for parity games special strategies that do not take the history of the play into account are sufficient: **Positional strategies** are of the form $\sigma : V_{\mathsf{E}} \to V$.



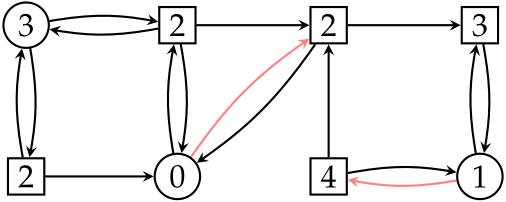
Parity games – notations

Game graph (or arena) $G = (V, V_\mathsf{E}, V_\mathsf{A}, E, c)$ with $c : V \to \{0, \ldots, k\}$ for some k.

A **strategy** σ for Eva is a mapping $\sigma : V^* V_\mathsf{E} \to V$ assigning a next move to finite plays ending in a vertex of Eva.

σ is a **winning strategy** for Eva if she wins all plays that she plays according to σ.

As we will see, for parity games special strategies that do not take the history of the play into account are sufficient: **Positional strategies** are of the form $\sigma : V_\mathsf{E} \to V$.



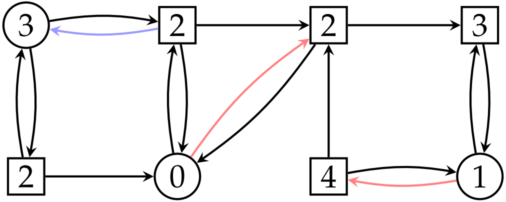
Parity games – notations

Game graph (or arena) $G = (V, V_\mathsf{E}, V_\mathsf{A}, E, c)$ with $c : V \to \{0, \dots, k\}$ for some k.

A **strategy** σ for Eva is a mapping $\sigma : V^* V_\mathsf{E} \to V$ assigning a next move to finite plays ending in a vertex of Eva.

σ is a **winning strategy** for Eva if she wins all plays that she plays according to σ.

As we will see, for parity games special strategies that do not take the history of the play into account are sufficient: **Positional strategies** are of the form $\sigma : V_\mathsf{E} \to V$.



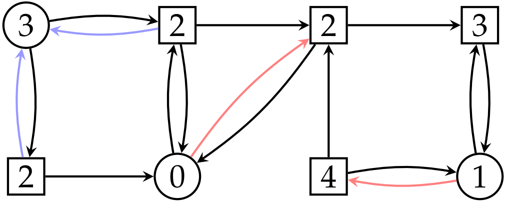
Parity games – notations

Game graph (or arena) $G = (V, V_E, V_A, E, c)$ with $c : V \to \{0, \ldots, k\}$ for some k.

A **strategy** σ for Eva is a mapping $\sigma : V^* V_E \to V$ assigning a next move to finite plays ending in a vertex of Eva.

σ is a **winning strategy** for Eva if she wins all plays that she plays according to σ.

As we will see, for parity games special strategies that do not take the history of the play into account are sufficient: **Positional strategies** are of the form $\sigma : V_E \to V$.



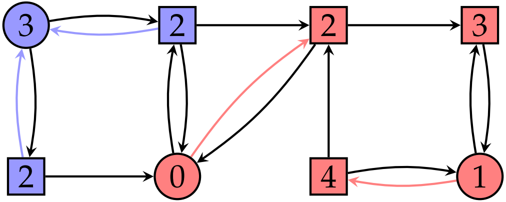
Parity games – notations

Game graph (or arena) $G = (V, V_{\mathsf{E}}, V_{\mathsf{A}}, E, c)$ with $c : V \to \{0, \ldots, k\}$ for some k.

A **strategy** σ for Eva is a mapping $\sigma : V^* V_{\mathsf{E}} \to V$ assigning a next move to finite plays ending in a vertex of Eva.

σ is a **winning strategy** for Eva if she wins all plays that she plays according to σ.

As we will see, for parity games special strategies that do not take the history of the play into account are sufficient: **Positional strategies** are of the form $\sigma : V_{\mathsf{E}} \to V$.



Solving parity games

The problem of solving parity games is

Given:	Parity game G and an initial vertex v_0
Question:	Does Eva have a winning strategy from v_0?

Solving parity games

The problem of solving parity games is

Given: Parity game G and an initial vertex v_0

Question: Does Eva have a winning strategy from v_0?

We are going to show that

1. Parity games are determined: From each vertex one of the players has a winning strategy.
2. Positional strategies are sufficient.

Solving parity games

The problem of solving parity games is

Given:	Parity game G and an initial vertex v_0
Question:	Does Eva have a winning strategy from v_0?

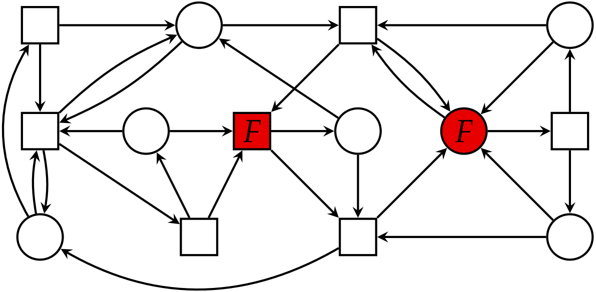
We are going to show that

1. Parity games are determined: From each vertex one of the players has a winning strategy.
2. Positional strategies are sufficient.

Before solving parity games we analyze how to solve simpler games with a reachability condition.

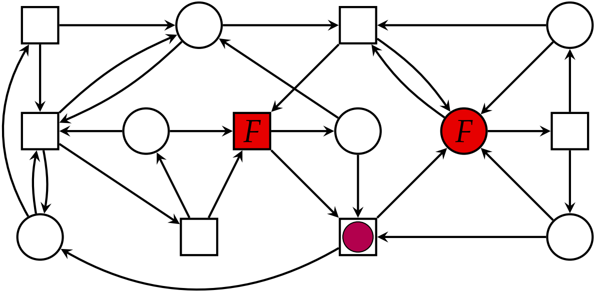
Reachability games

A **reachability game** is of the form $\mathcal{G} = (G, F)$ where F is the set of vertices that Eva wants to reach.



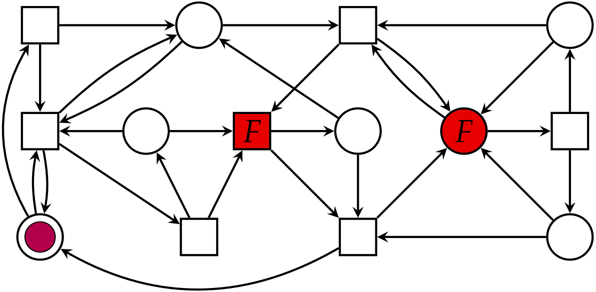
Reachability games

A **reachability game** is of the form $\mathcal{G} = (G, F)$ where F is the set of vertices that Eva wants to reach.



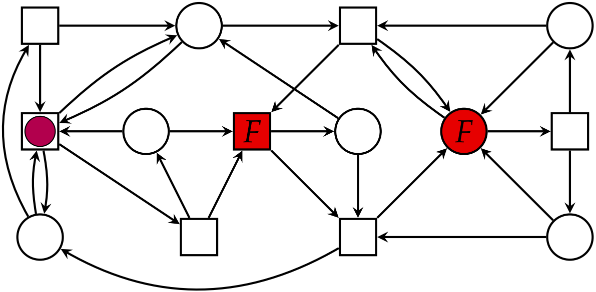
Reachability games

A **reachability game** is of the form $\mathcal{G} = (G, F)$ where F is the set of vertices that Eva wants to reach.



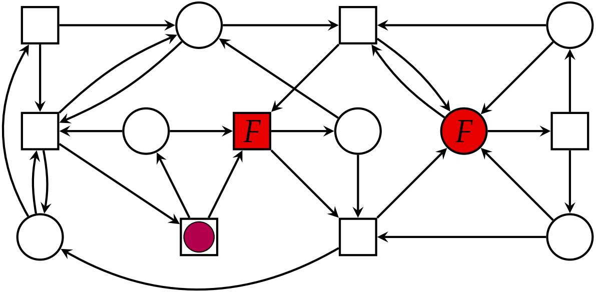
Reachability games

A **reachability game** is of the form $\mathcal{G} = (G, F)$ where F is the set of vertices that Eva wants to reach.



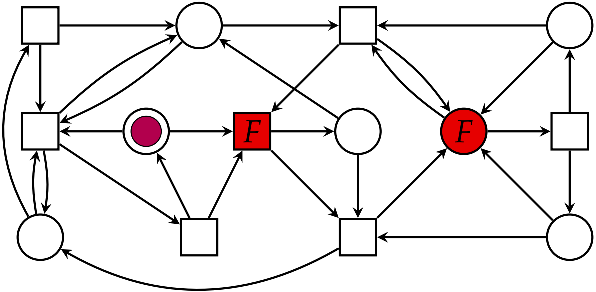
Reachability games

A **reachability game** is of the form $\mathcal{G} = (G, F)$ where F is the set of vertices that Eva wants to reach.



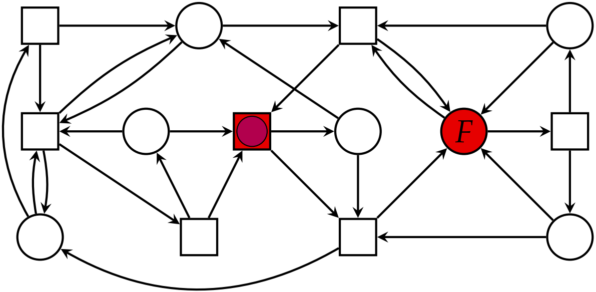
Reachability games

A **reachability game** is of the form $\mathcal{G} = (G, F)$ where F is the set of vertices that Eva wants to reach.



Reachability games

A **reachability game** is of the form $\mathcal{G} = (G, F)$ where F is the set of vertices that Eva wants to reach.



Operator Pre

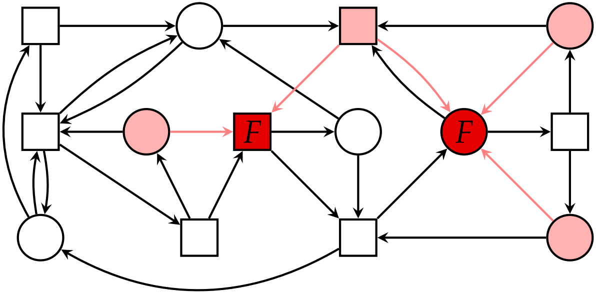
On $Pre_{\mathsf{E}}(F)$ Eva can force to reach F in one step:

$$\begin{array}{rcl} Pre_{\mathsf{E}}(F) & = & \{v \in V_{\mathsf{E}} \mid \exists v' \in E(v) \cap F\} \\ & & \cup \{v \in V_{\mathsf{A}} \mid \neg \exists v' \in E(v) \setminus F\} \end{array}$$

Operator *Pre*

On $Pre_{\mathsf{E}}(F)$ Eva can force to reach F in one step:

$$\begin{aligned} Pre_{\mathsf{E}}(F) &= \{v \in V_{\mathsf{E}} \mid \exists v' \in E(v) \cap F\} \\ &\cup \{v \in V_{\mathsf{A}} \mid \neg\exists v' \in E(v) \setminus F\} \end{aligned}$$

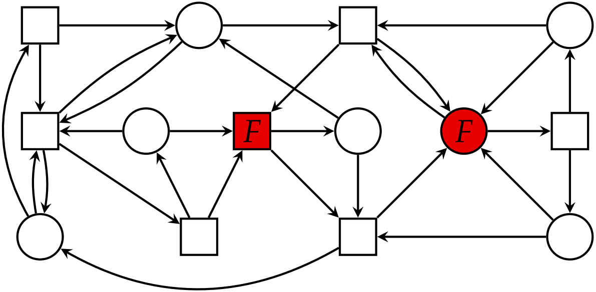


Attractor

The **attractor for Eva of a set** F is the set of all nodes from which Eva can force to reach F in finitely many moves:

$$Attr_{\mathsf{E}}(F) = \bigcup_{i \geq 0} Attr^i_{\mathsf{E}}(F)$$

- $Attr^0_{\mathsf{E}}(F) = F$
- $Attr^{i+1}_{\mathsf{E}}(F) = Attr^i_{\mathsf{E}}(F) \cup Pre_{\mathsf{E}}(Attr^i_{\mathsf{E}}(F))$

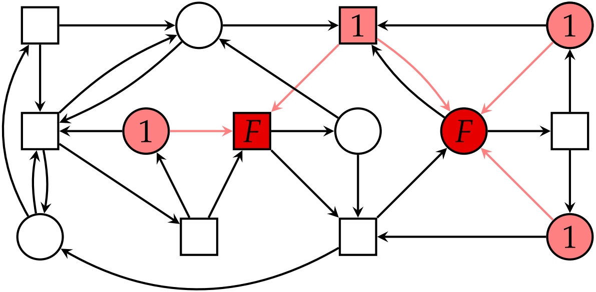


Attractor

The **attractor for Eva of a set** F is the set of all nodes from which Eva can force to reach F in finitely many moves:

$$Attr_{\mathsf{E}}(F) = \bigcup_{i \geq 0} Attr^{i}_{\mathsf{E}}(F)$$

- $Attr^{0}_{\mathsf{E}}(F) = F$
- $Attr^{i+1}_{\mathsf{E}}(F) = Attr^{i}_{\mathsf{E}}(F) \cup Pre_{\mathsf{E}}(Attr^{i}_{\mathsf{E}}(F))$

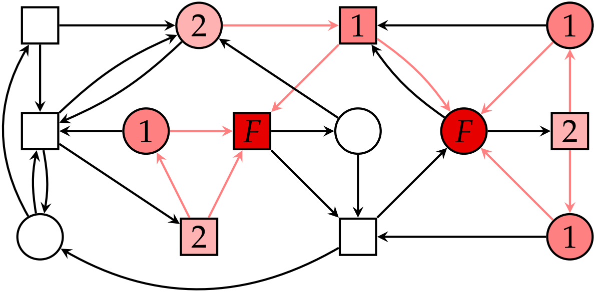


Attractor

The **attractor for Eva of a set** F is the set of all nodes from which Eva can force to reach F in finitely many moves:

$$Attr_{\mathsf{E}}(F) = \bigcup_{i \geq 0} Attr^{i}_{\mathsf{E}}(F)$$

- $Attr^{0}_{\mathsf{E}}(F) = F$
- $Attr^{i+1}_{\mathsf{E}}(F) = Attr^{i}_{\mathsf{E}}(F) \cup Pre_{\mathsf{E}}(Attr^{i}_{\mathsf{E}}(F))$

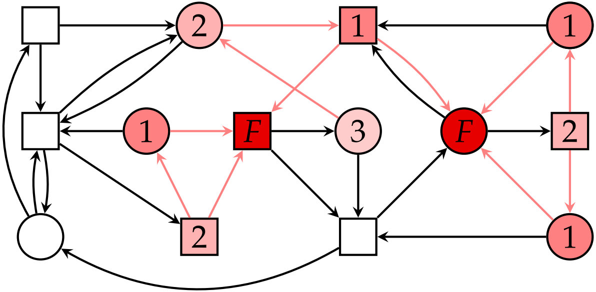


Attractor

The **attractor for Eva of a set** F is the set of all nodes from which Eva can force to reach F in finitely many moves:

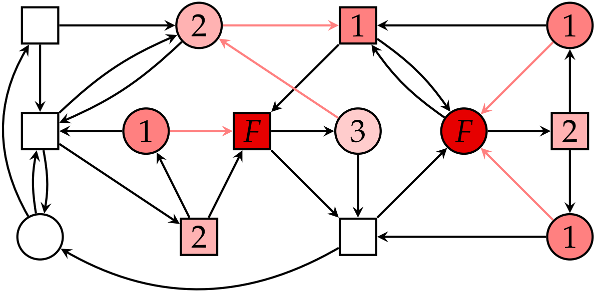
$$Attr_{\mathsf{E}}(F) = \bigcup_{i \geq 0} Attr^{i}_{\mathsf{E}}(F)$$

- $Attr^{0}_{\mathsf{E}}(F) = F$
- $Attr^{i+1}_{\mathsf{E}}(F) = Attr^{i}_{\mathsf{E}}(F) \cup Pre_{\mathsf{E}}(Attr^{i}_{\mathsf{E}}(F))$



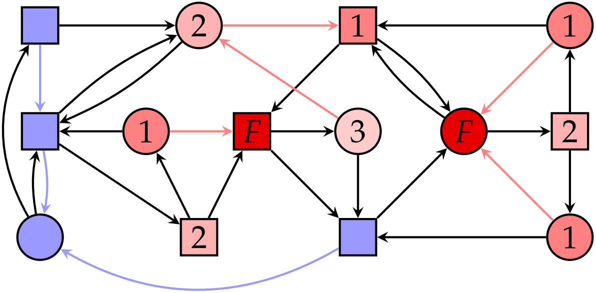
Attractor and trap strategy

- On $Attr^i_{\mathsf{E}}(F)$ Eva can ensure that the next move leads to $Attr^j_{\mathsf{E}}(F)$ for $j < i$. Call this an **attractor strategy**.



Attractor and trap strategy

- On $Attr^{i}_{\mathsf{E}}(F)$ Eva can ensure that the next move leads to $Attr^{j}_{\mathsf{E}}(F)$ for $j < i$. Call this an **attractor strategy**.
- The complement $V \setminus Attr_{\mathsf{E}}(F)$ is a **trap** for Eva: Adam can ensure that the play remains in this set. Call this a **trap strategy** for Adam.



Determinacy of reachability games

Theorem

Reachability games $\mathcal{G} = (G, F)$ are determined with positional strategies. The winning region of Eva is $Attr_E(F)$.

Determinacy of reachability games

Theorem

Reachability games $\mathcal{G} = (G, F)$ are determined with positional strategies. The winning region of Eva is $Attr_E(F)$.

This theorem was already shown in 1913 by E. Zermelo:

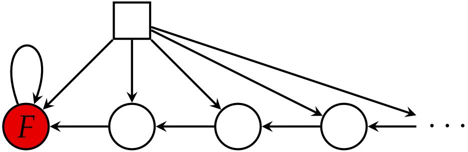
“Über eine Anwendung der Mengenlehre auf die Theorie des Schachspiels”
(On an application of set theory to the theory of chess)

Final remarks

- An attractor can be computed in linear time (in the size of the graph)

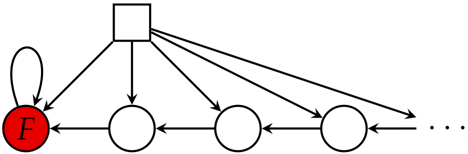
Final remarks

- An attractor can be computed in linear time (in the size of the graph)
- For graphs with infinite degree it is not enough to stop the attractor computation after ω iterations.
  Illustration:



Final remarks

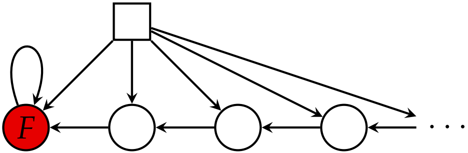
- An attractor can be computed in linear time (in the size of the graph)
- For graphs with infinite degree it is not enough to stop the attractor computation after ω iterations.
  Illustration:



- A trap defines a subgame (each vertex has at least one edge staying inside the trap). This is important when inductively solving games.

Final remarks

- An attractor can be computed in linear time (in the size of the graph)
- For graphs with infinite degree it is not enough to stop the attractor computation after ω iterations.
  Illustration:

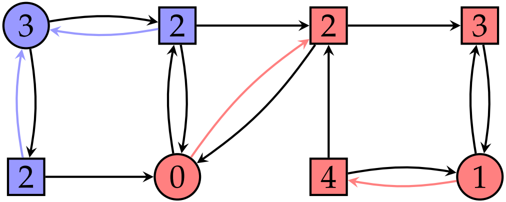


- A trap defines a subgame (each vertex has at least one edge staying inside the trap). This is important when inductively solving games.
- The dual condition of avoiding a set of vertices is called **safety condition**. A reachability condition is a safety condition for the opponent.

Summary

- Reachability games: goal of Eva is to reach a set F of vertices
- Attractor $Attr_{\mathsf{E}}(F)$ (based on Pre_{E})
- Complement of an attractor is a trap for the other player
- Attractor and trap strategies are positional
- Reachability games are positionally determined

Back to parity games



Eva wins iff the maximal priority appearing infinitely often is even.

Goal: Compute winning areas and winning strategies.

Positional determinacy of parity games

Theorem (Emerson/Jutla'88, Mostowski'91)

Parity games are determined with positional winning strategies for both players on their winning areas.

Positional determinacy of parity games

Theorem (Emerson/Jutla'88, Mostowski'91)

Parity games are determined with positional winning strategies for both players on their winning areas.

A non-constructive proof:

- Induction on the number of priorities
- Let k be the maximal priority and assume that it is even
- Let U be the set of vertices on which Adam has a positional winning strategy

Illustration of proof idea

Theorem (Emerson/Jutla'88, Mostowski'91)

Parity games are determined with positional winning strategies for both players on their winning areas.

Illustration of proof idea

Theorem (Emerson/Jutla'88, Mostowski'91)

Parity games are determined with positional winning strategies for both players on their winning areas.

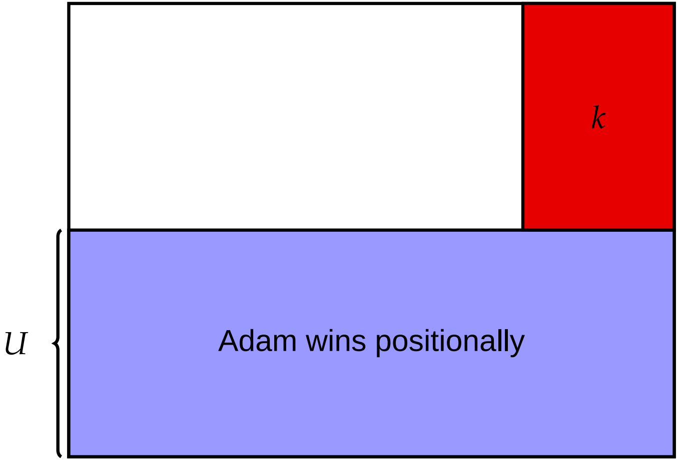


Illustration of proof idea

Theorem (Emerson/Jutla'88, Mostowski'91)

Parity games are determined with positional winning strategies for both players on their winning areas.

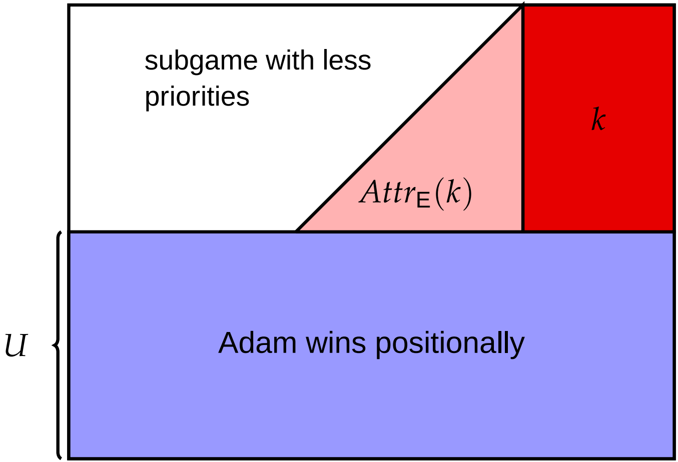


Illustration of proof idea

Theorem (Emerson/Jutla'88, Mostowski'91)

Parity games are determined with positional winning strategies for both players on their winning areas.

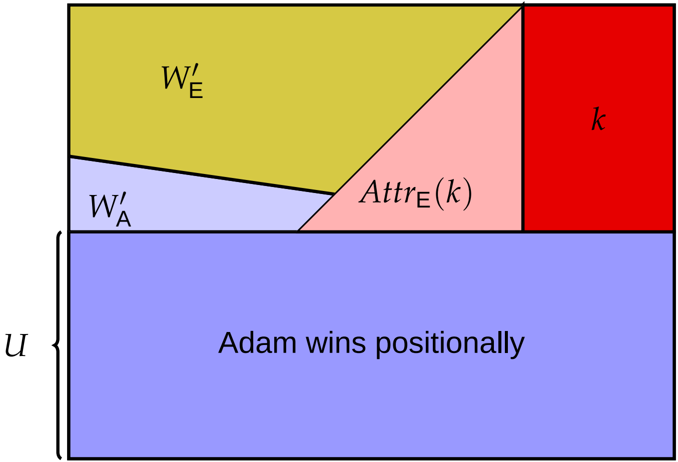


Illustration of proof idea

Theorem (Emerson/Jutla'88, Mostowski'91)

Parity games are determined with positional winning strategies for both players on their winning areas.

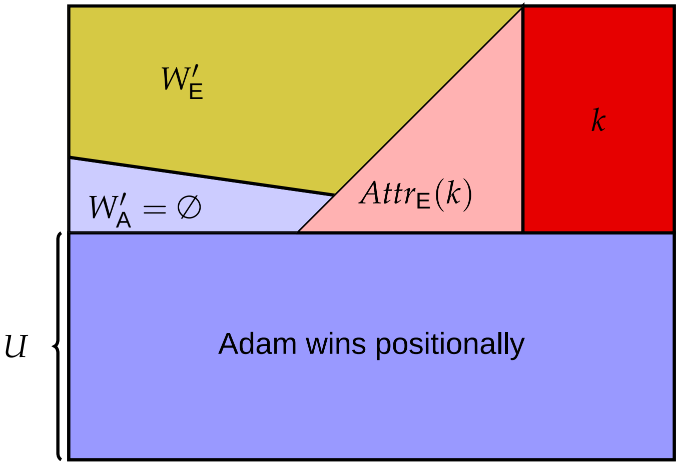


Illustration of proof idea

Theorem (Emerson/Jutla'88, Mostowski'91)

Parity games are determined with positional winning strategies for both players on their winning areas.

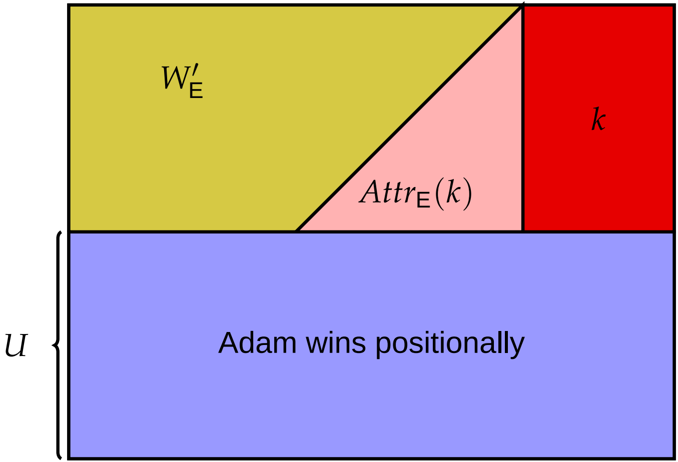


Illustration of proof idea

Theorem (Emerson/Jutla'88, Mostowski'91)

Parity games are determined with positional winning strategies for both players on their winning areas.

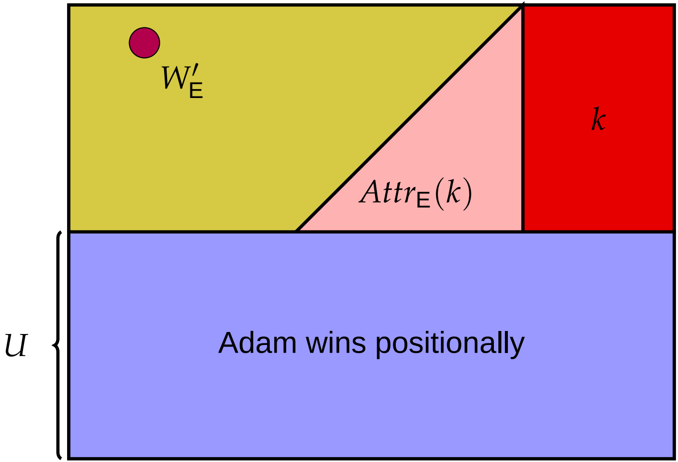


Illustration of proof idea

Theorem (Emerson/Jutla'88, Mostowski'91)

Parity games are determined with positional winning strategies for both players on their winning areas.

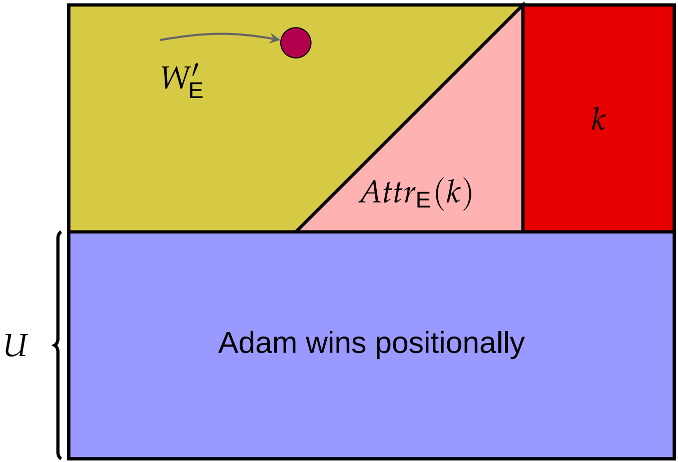


Illustration of proof idea

Theorem (Emerson/Jutla'88, Mostowski'91)

Parity games are determined with positional winning strategies for both players on their winning areas.

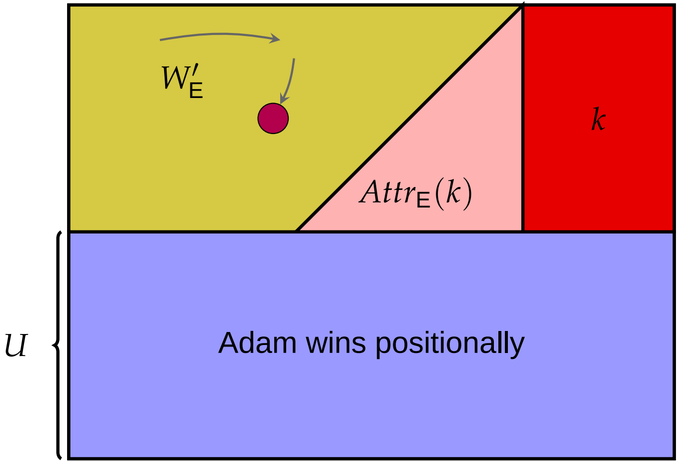


Illustration of proof idea

Theorem (Emerson/Jutla'88, Mostowski'91)

Parity games are determined with positional winning strategies for both players on their winning areas.

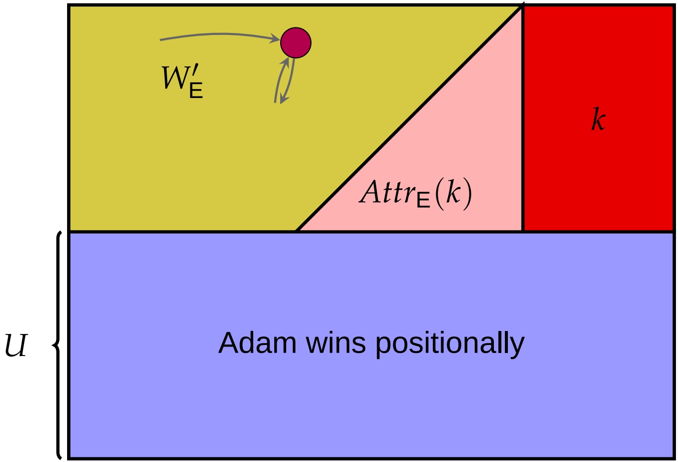


Illustration of proof idea

Theorem (Emerson/Jutla'88, Mostowski'91)

Parity games are determined with positional winning strategies for both players on their winning areas.

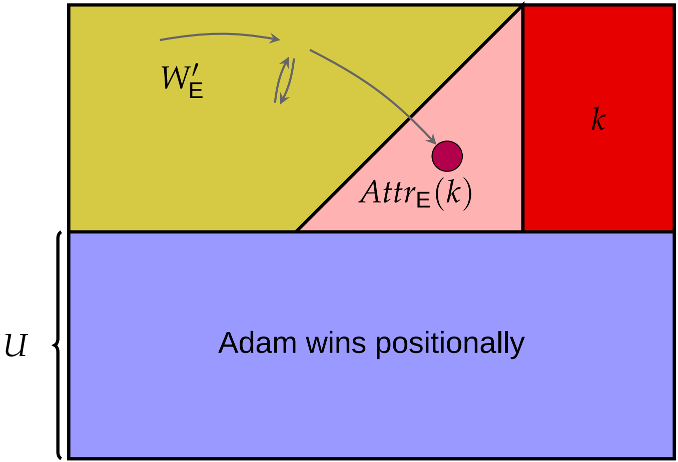


Illustration of proof idea

Theorem (Emerson/Jutla'88, Mostowski'91)

Parity games are determined with positional winning strategies for both players on their winning areas.

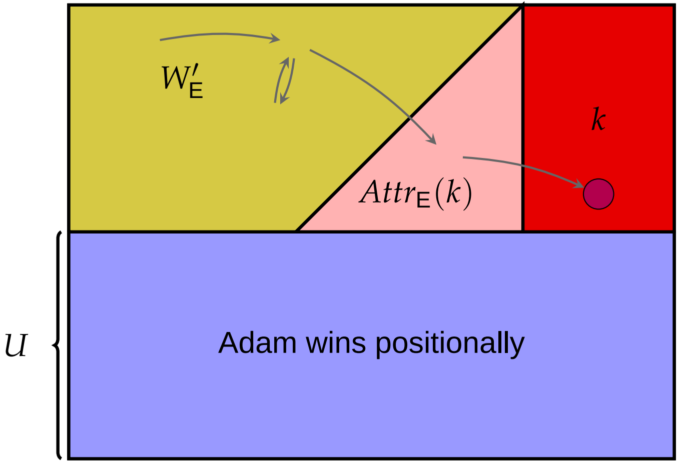
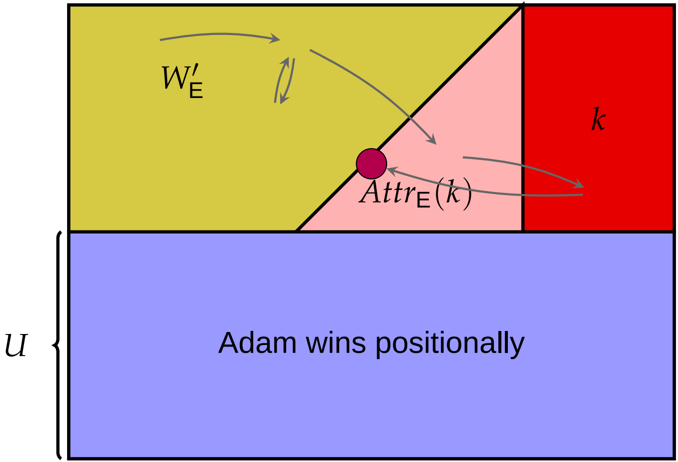


Illustration of proof idea

Theorem (Emerson/Jutla'88, Mostowski'91)

Parity games are determined with positional winning strategies for both players on their winning areas.



Remarks

- The proof is non-constructive because we pick the set U of vertices from which Adam has a positional winning strategy.
- Now we see how to algorithmically solve parity games.

An NP algorithm

Theorem

The decision problem "Given a parity game $\mathcal{G}$ and an initial vertex v_0, does Eva have a winning strategy from v_0?" is in NP $\cap$ co-NP.

An NP algorithm

Theorem

The decision problem "Given a parity game $\mathcal{G}$ and an initial vertex v_0, does Eva have a winning strategy from v_0?" is in NP $\cap$ co-NP.

Proof

Membership in NP:

1. Guess a positional strategy for Eva
2. Verify that this strategy is winning from v_0
   (this is possible in polynomial time; not obvious but not too difficult)

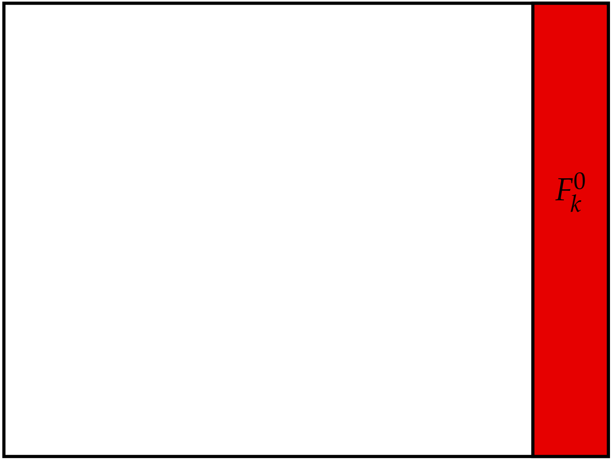
Membership in co-NP follows from positional determinacy.

Remarks on the complexity

- A deterministic algorithm based on the previous idea can test all positional strategies for Eva. The complexity is exponential in the size of the graph.
- By an inductive construction we can show that parity games can be solved in time that is only exponential in the number of priorities.

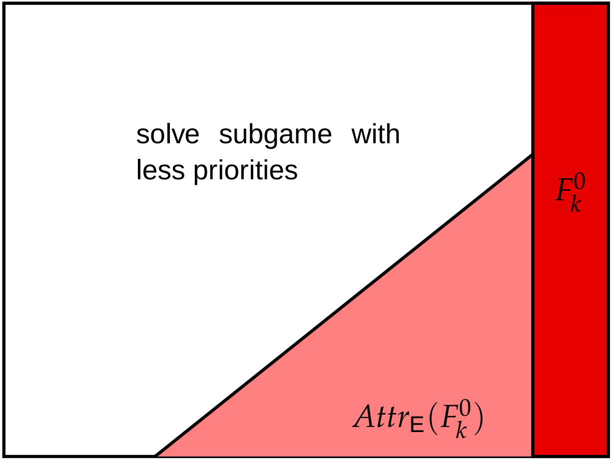
Inductive construction

Highest priority k (assumed to be even)



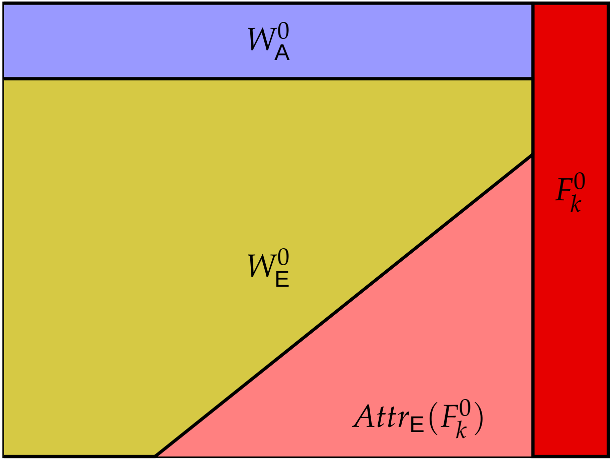
Inductive construction

Highest priority k (assumed to be even)



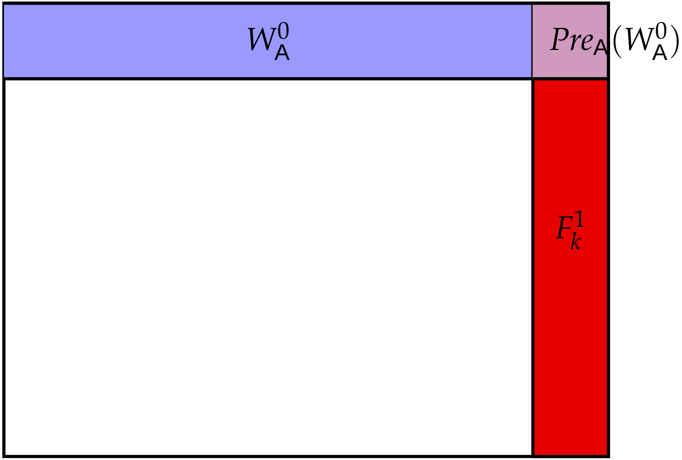
Inductive construction

Highest priority k (assumed to be even)



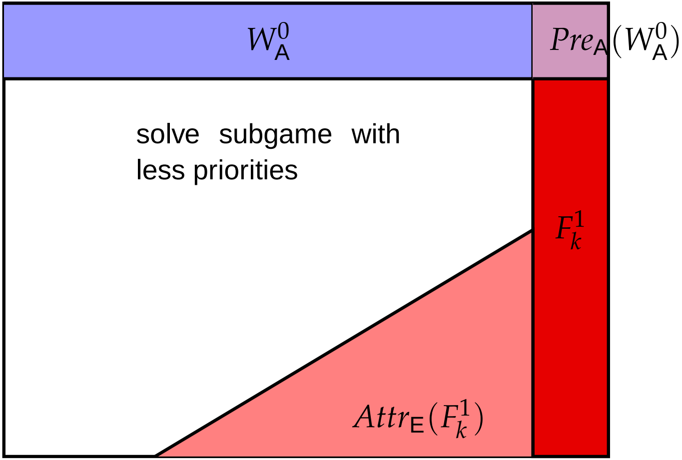
Inductive construction

Highest priority k (assumed to be even)



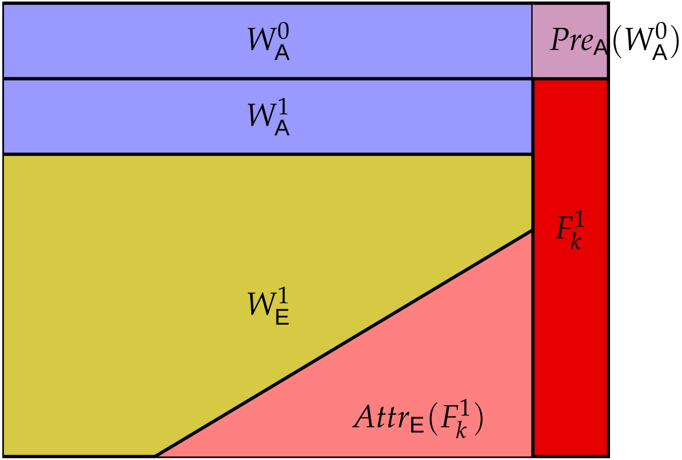
Inductive construction

Highest priority k (assumed to be even)



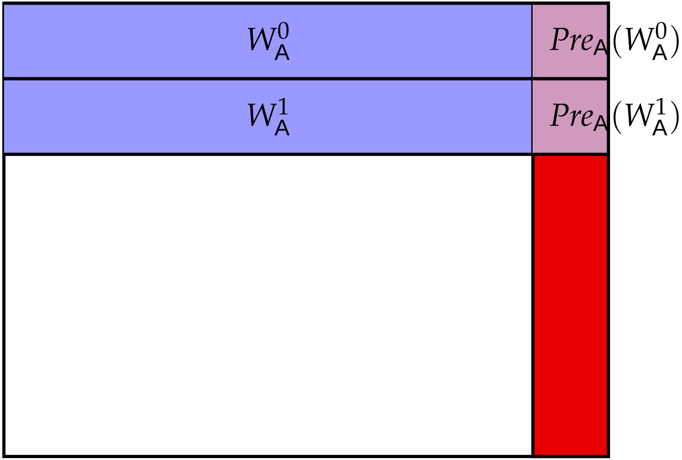
Inductive construction

Highest priority k (assumed to be even)



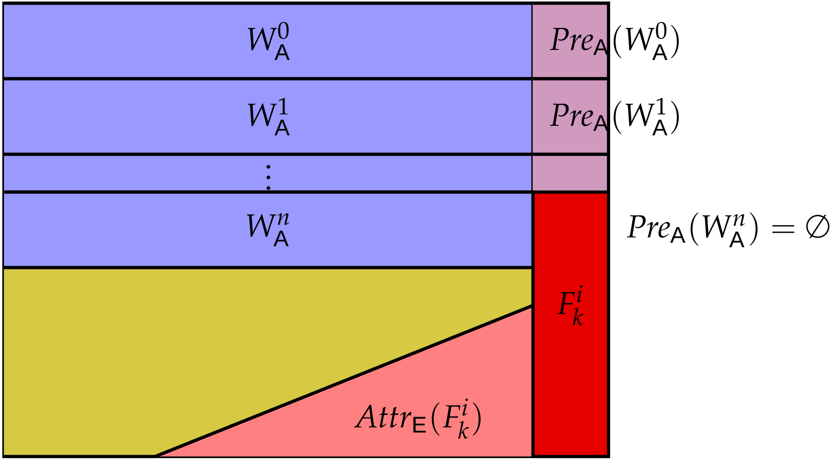
Inductive construction

Highest priority k (assumed to be even)



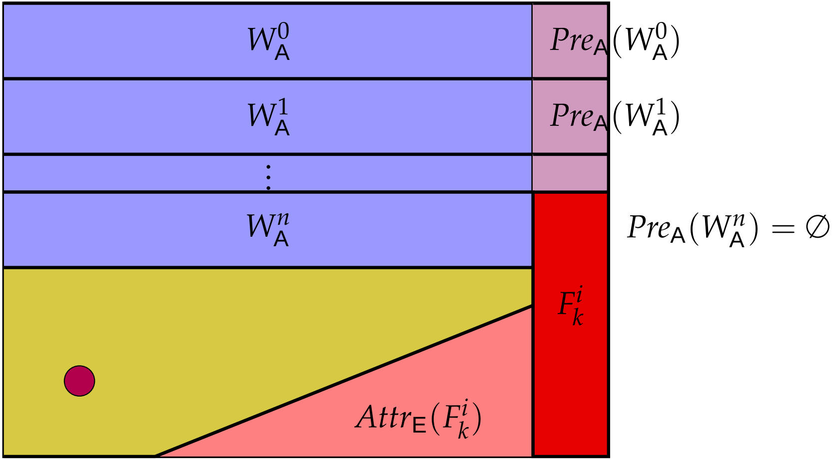
Inductive construction

Highest priority k (assumed to be even)



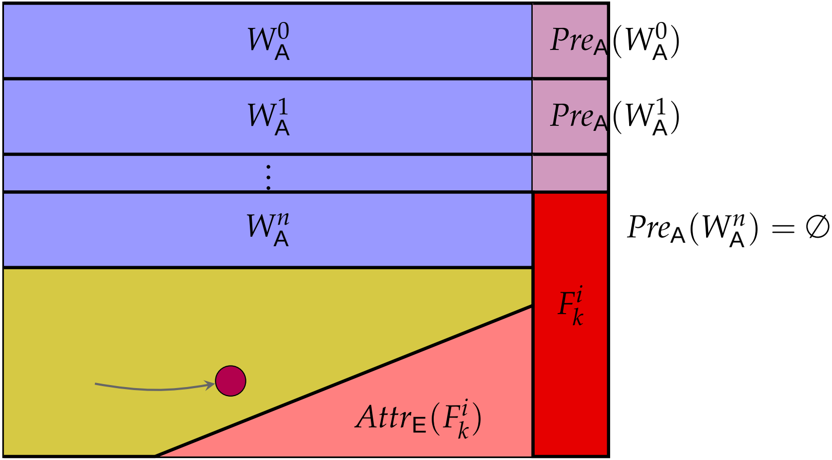
Inductive construction

Highest priority k (assumed to be even)



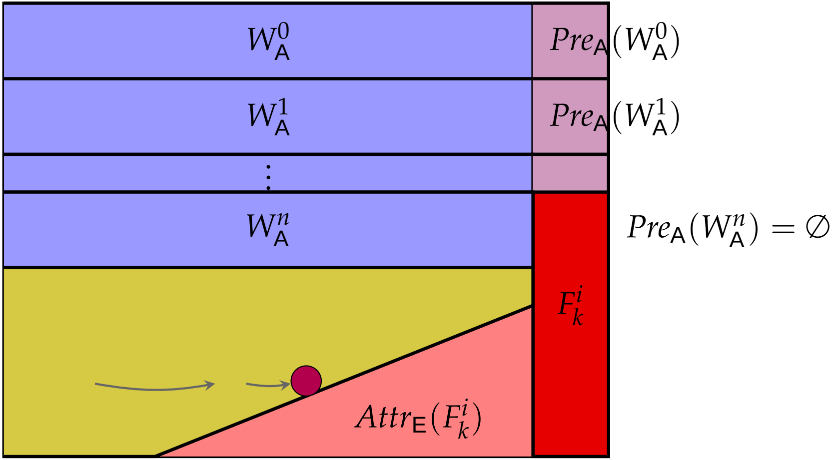
Inductive construction

Highest priority k (assumed to be even)



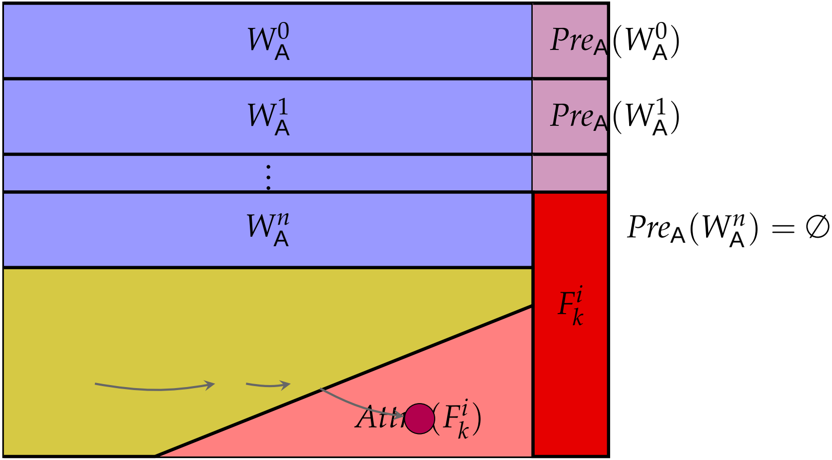
Inductive construction

Highest priority k (assumed to be even)



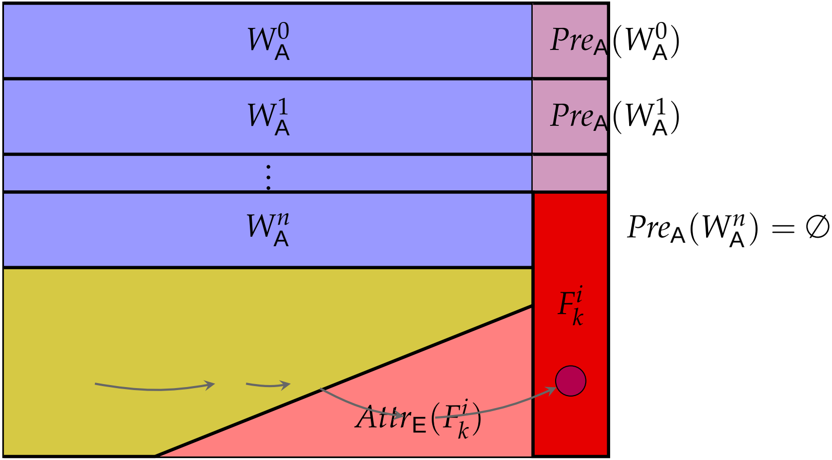
Inductive construction

Highest priority k (assumed to be even)



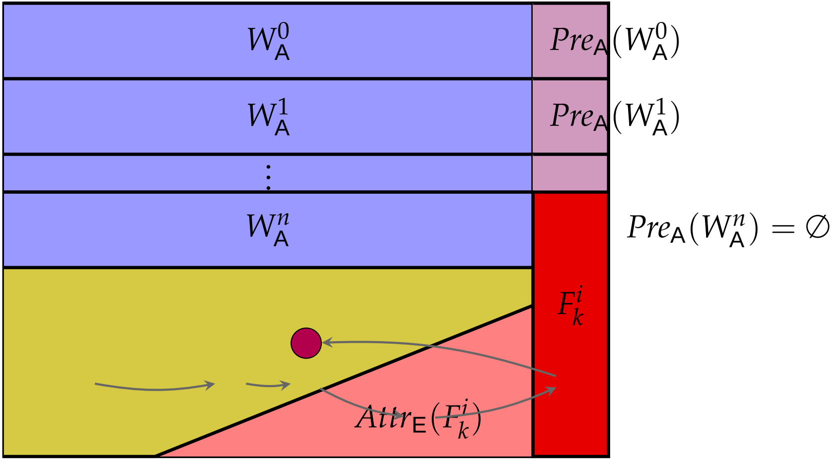
Inductive construction

Highest priority k (assumed to be even)



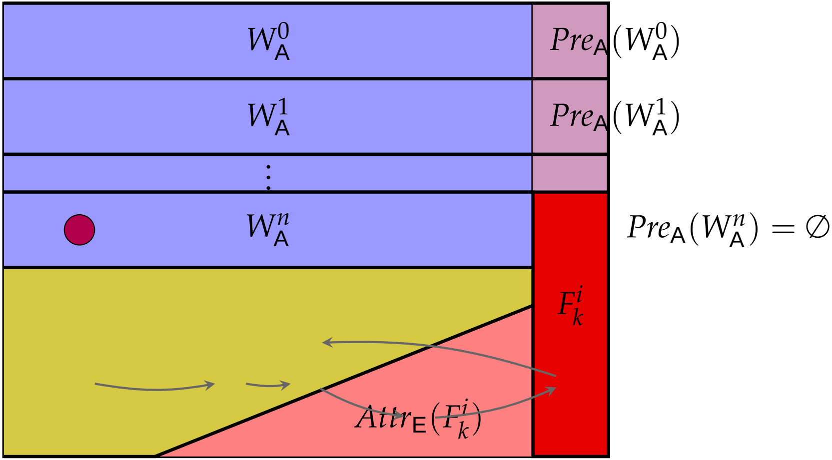
Inductive construction

Highest priority k (assumed to be even)



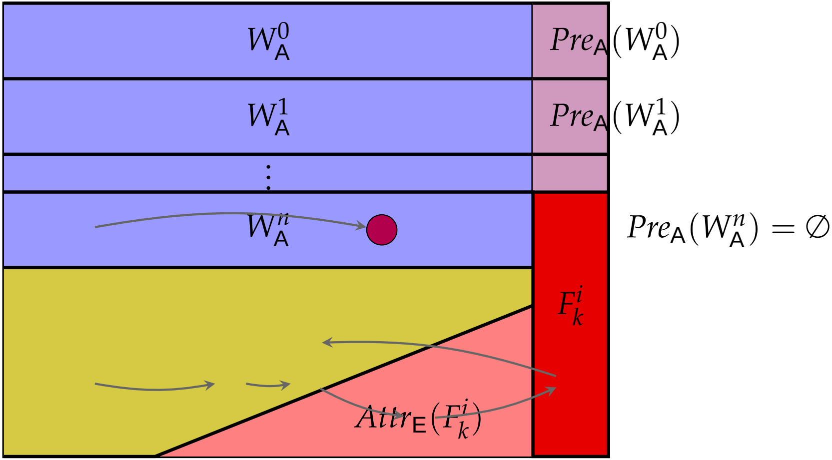
Inductive construction

Highest priority k (assumed to be even)



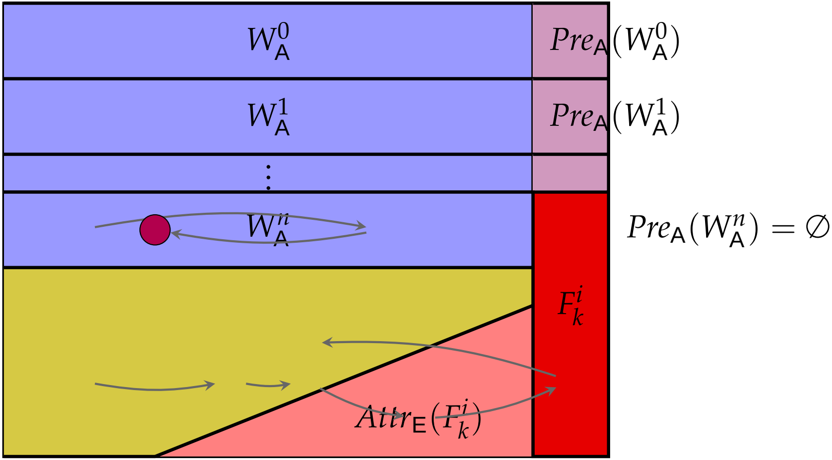
Inductive construction

Highest priority k (assumed to be even)



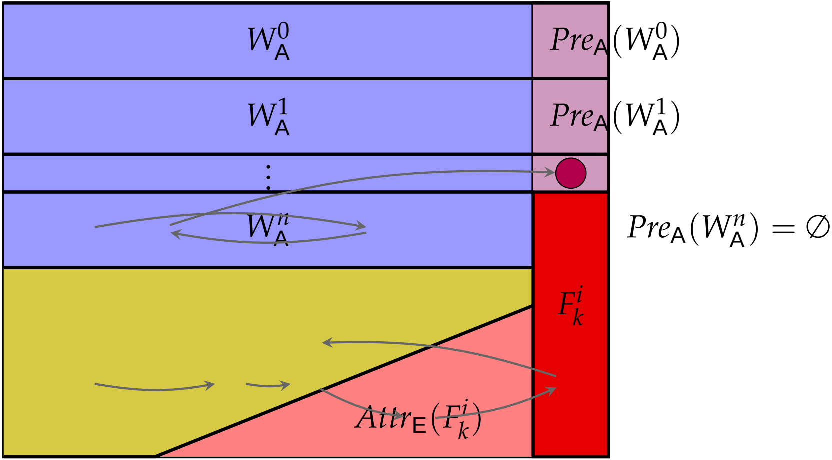
Inductive construction

Highest priority k (assumed to be even)



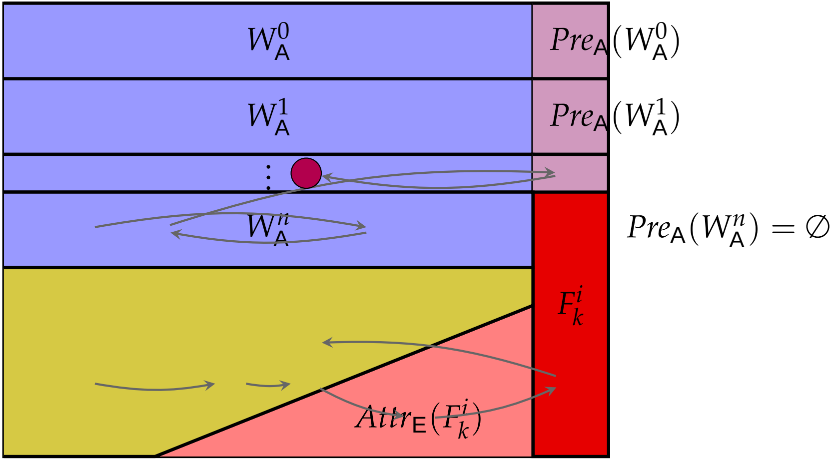
Inductive construction

Highest priority k (assumed to be even)



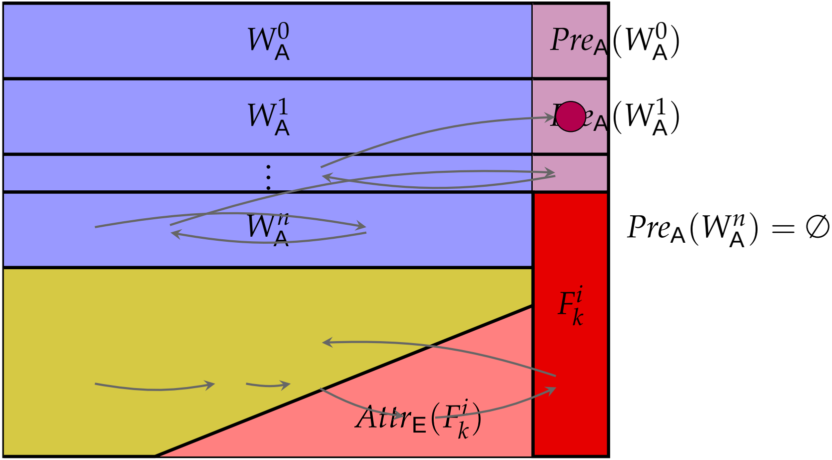
Inductive construction

Highest priority k (assumed to be even)



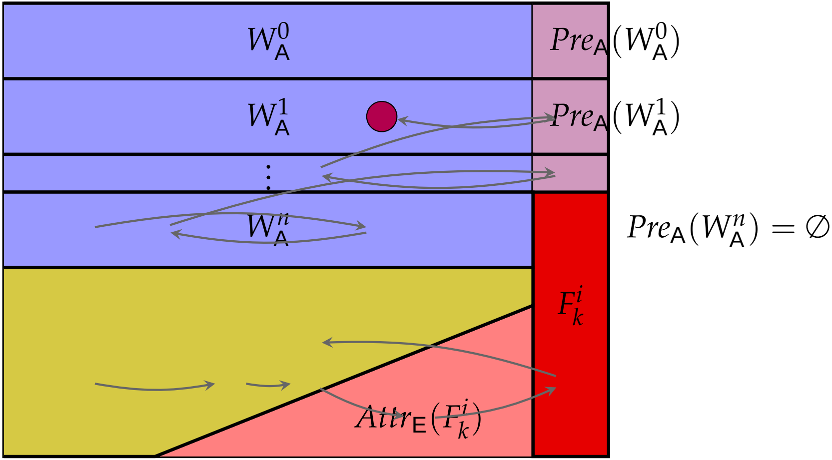
Inductive construction

Highest priority k (assumed to be even)



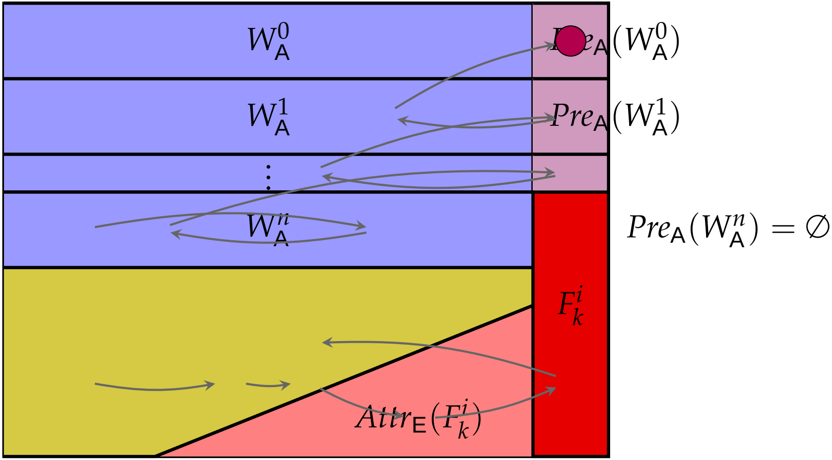
Inductive construction

Highest priority k (assumed to be even)



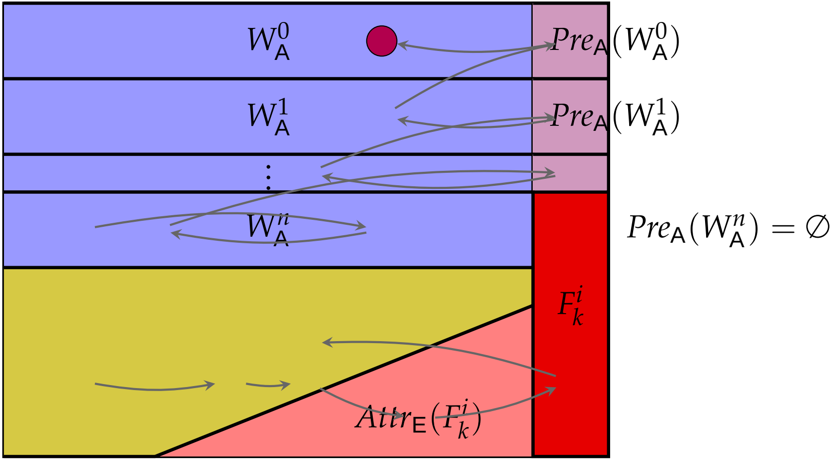
Inductive construction

Highest priority k (assumed to be even)



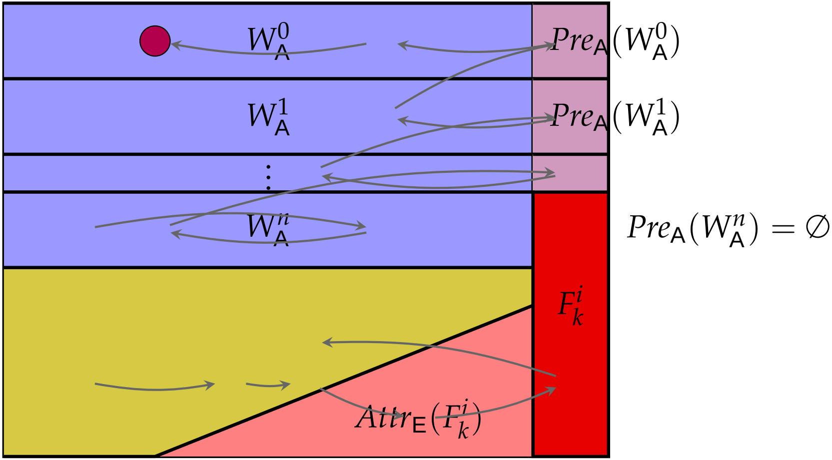
Inductive construction

Highest priority k (assumed to be even)



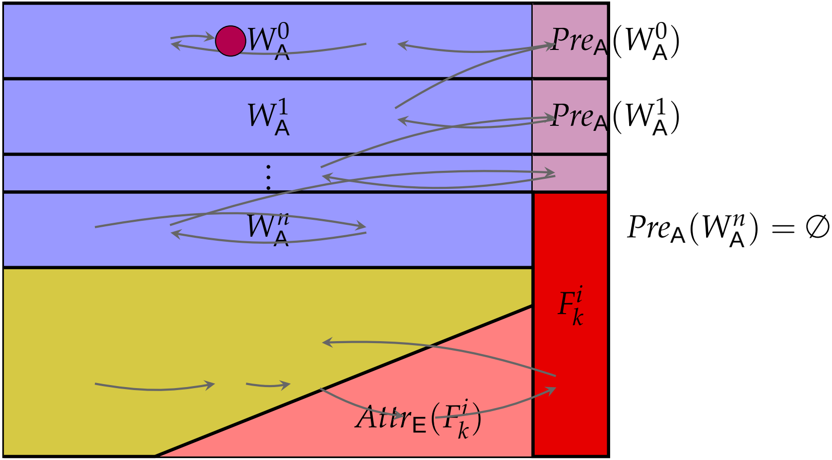
Inductive construction

Highest priority k (assumed to be even)



Inductive construction

Highest priority k (assumed to be even)



Complexity

Inductive construction for Graph of size n with k priorities:

- Compute attractor: linear in n
- Solve subgame with $k-1$ priorities
- Number of iterations bounded by n

$\leadsto \mathcal{O}(n^k)$

Other Algorithms

- Small progress measures (Jurdziński 2000): $\mathcal{O}(n^{\frac{k}{2}})$
- Inductive construction with preprocessing (Jurdziński, Paterson, Zwick 2006): $\mathcal{O}(n^{\sqrt{n}})$
- Inductive construction with refined preprocessing (Schewe 2007): $\mathcal{O}(n^{\frac{k}{3}})$
- Strategy improvement (Vöge, Jurdziński 2000)
  - Recently superpolynomial behavior has been shown (Friedmann 2009)

Other Algorithms

- Small progress measures (Jurdziński 2000): $\mathcal{O}(n^{\frac{k}{2}})$
- Inductive construction with preprocessing (Jurdziński, Paterson, Zwick 2006): $\mathcal{O}(n^{\sqrt{n}})$
- Inductive construction with refined preprocessing (Schewe 2007): $\mathcal{O}(n^{\frac{k}{3}})$
- Strategy improvement (Vöge, Jurdziński 2000)
  - Recently superpolynomial behavior has been shown (Friedmann 2009)

Open problem:
Can parity games be solved in polynomial time?

Consequences for L_μ

- The problem $T \models \varphi$ for an L_μ formula φ can be solved in time

$$\mathcal{O}((|T| \cdot |\varphi|)^k)$$

  where k is the alternation depth of the fixpoints in φ.
- For small alternation depth the model checking problem for L_μ can be solved efficiently.

1. Introduction: model checking
2. Sequential specifications
   - Linear temporal logic
   - Automata on infinite words
3. Branching specifications
   - Modal μ-calculus
   - Parity games
4. Satisfiability and synthesis
   - Synthesis problem
   - Automata on infinite trees

Synthesis problem

Origin

Circuit synthesis and Church's problem (1957)

Setting:

- Sequence of input signals arrives
- Circuit produces a sequence of output signals (depending on the inputs it has seen)
- Result is a non-terminating sequence of input and output signals
- A logical specification describes the desired properties of these sequences

$$\text{input} \longrightarrow \boxed{\text{circuit } f} \longrightarrow \text{output}$$

Task: Automatically synthesize a circuit from the specification

More formally

$$\text{input} \in \Sigma_1 \longrightarrow \boxed{\text{circuit } f} \longrightarrow \text{output} \in \Sigma_2$$

- Input sequence $\alpha \in \Sigma_1^\omega$ and output sequence $\beta \in \Sigma_2^\omega$
- Specification $\varphi(\alpha, \beta)$

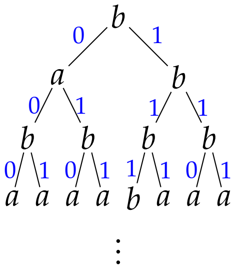
Problem:

- Decide if the there is a sequential transformation $f : \Sigma_1^* \to \Sigma_2$ realizing φ, and construct one if possible.

Sequential transformations as infinite trees

- For simplicity we assume that $\Sigma_1 = \{0, 1\}$, which results in binary trees.
- The nodes of the tree are labeled from Σ_2.
- Formally, a tree is a mapping $t : \{0, 1\}^* \to \Sigma$.

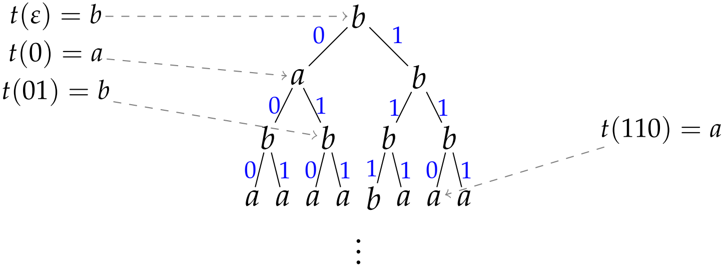
Example: $\Sigma_1 = \{0, 1\}$ and $\Sigma_2 = \{a, b\}$



Sequential transformations as infinite trees

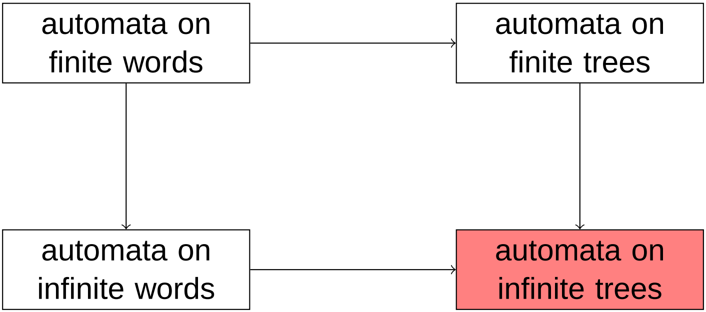
- For simplicity we assume that $\Sigma_1 = \{0,1\}$, which results in binary trees.
- The nodes of the tree are labeled from Σ_2.
- Formally, a tree is a mapping $t : \{0,1\}^* \to \Sigma$.

Example: $\Sigma_1 = \{0,1\}$ and $\Sigma_2 = \{a,b\}$



Automata on infinite trees

Automata on infinite trees

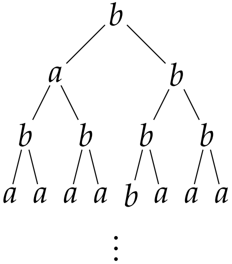


- Robust model (good closure and algorithmic properties)
- Captures many known specification logics

Parity tree automata

- $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, pri)$
- Transitions of the form (q, a, q', q'')

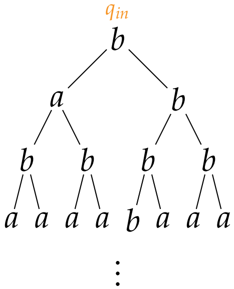
Run of $\mathcal{A}$:



Parity tree automata

- $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, pri)$
- Transitions of the form (q, a, q', q'')

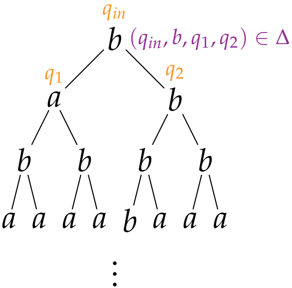
Run of $\mathcal{A}$:



Parity tree automata

- $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, pri)$
- Transitions of the form (q, a, q', q'')

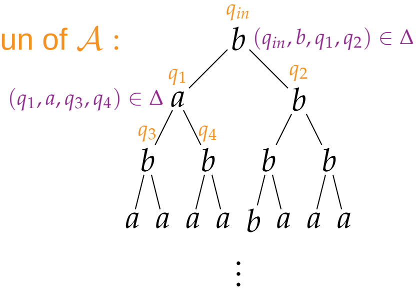
Run of $\mathcal{A}$:



Parity tree automata

- $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, pri)$
- Transitions of the form (q, a, q', q'')

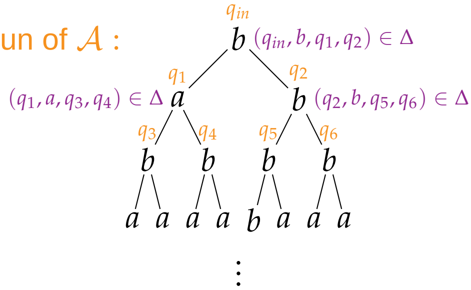
Run of $\mathcal{A}$:



Parity tree automata

- $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, pri)$
- Transitions of the form (q, a, q', q'')

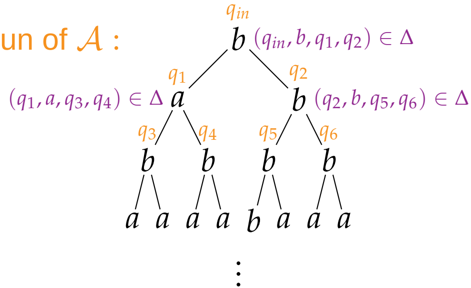
Run of $\mathcal{A}$:



Parity tree automata

- $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, pri)$
- Transitions of the form (q, a, q', q'')

Run of $\mathcal{A}$:



- Priority function $pri : Q \to \mathbb{N}$
- Run accepting if on each path the maximal priority appearing infinitely often is even.
- Tree accepted if there is an accepting run on this tree. $T(\mathcal{A})$ denotes the language of accepted trees.

Solving the synthesis problem

$$\text{input} \in \Sigma_1 \longrightarrow \boxed{\text{circuit } f} \longrightarrow \text{output} \in \Sigma_2$$

- Input sequence $\alpha \in \Sigma_1^\omega$ and output sequence $\beta \in \Sigma_2^\omega$
- Specification $\varphi(\alpha, \beta)$ in LTL

Solution:

- Construct a PTA $\mathcal{A}_\varphi$ that accepts those trees $t : \Sigma_1 \to \Sigma_2$ such that each path satisfies φ (doubly exponential).
- Check $\mathcal{A}_\varphi$ for emptiness (equivalent to solving parity games).
- If $\mathcal{A}_\varphi$ accepts some tree then a finite representation can be constructed. This is a sequential transformation as required.

Summary

model checking:

system $\models$ specification?

LTL

$\uparrow$

tool: ω-automata

μ-calculus

$\uparrow$

tool: parity games

satisfiability / synthesis:

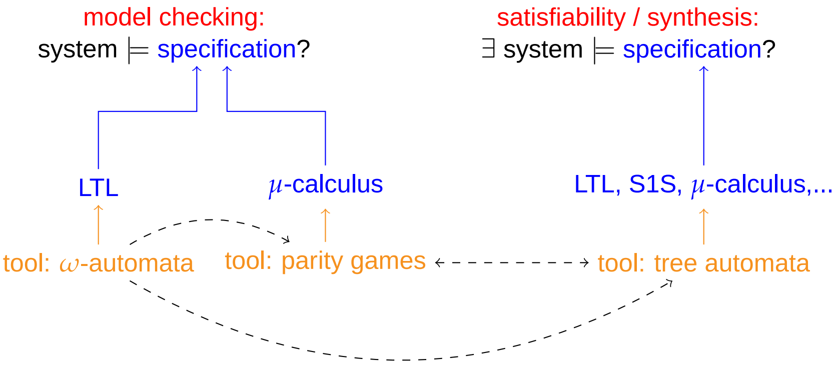
$\exists$ system $\models$ specification?

LTL, S1S, μ-calculus,...

$\uparrow$

tool: tree automata

Summary


model checking:
system $\models$ specification?
satisfiability / synthesis:
$\exists$ system $\models$ specification?
LTL
μ-calculus
LTL, S1S, μ-calculus,...
tool: ω-automata
tool: parity games
tool: tree automata