Strategy Synthesis for Infinite Games

Christof Löding

RWTH Aachen

On the Posterity of Büchi

Lausanne, March 31 – April 1, 2011

Starting Point

SOLVING SEQUENTIAL CONDITIONS BY FINITE-STATE STRATEGIES(1)

BY

J. RICHARD BUCHI AND LAWRENCE H. LANDWEBER

Our main purpose is to present an algorithm which decides whether or not a condition $\mathfrak{C}(X, Y)$ stated in sequential calculus admits a finite automata solution, and produces one if it exists. This solves a problem stated in [4] and contains, as a very special case, the answer to Case 4 left open in [6]. In an equally appealing form the result can be restated in the terminology of [7], [10], [15]: Every ω-game definable in sequential calculus is determined. Moreover the player who has a winning strategy, in fact, has a winning finite-state strategy, that is one which can effectively be played in a strong sense. The main proof, that of the central Theorem 1, will be presented at the end. We begin with a discussion of its consequences.

Outline

1. Büchi-Landweber Theorem

2. Extensions
   - Pushdown Games
   - Delay Games

3. Implementations

Problem Setting

- Infinite input sequence α over I
- Infinite output sequence β over J
- Formula φ specifying the “good” sequences from $(I \times J)^\omega$

Problem Setting

- Infinite input sequence α over I
- Infinite output sequence β over J
- Formula φ specifying the "good" sequences from $(I \times J)^\omega$

Synthesis problem: Decide if the there is a program $P : I^* \longrightarrow J$ realizing φ, and construct one if possible.

Problem Setting

- Infinite input sequence α over I
- Infinite output sequence β over J
- Formula φ specifying the "good" sequences from $(I \times J)^\omega$

Synthesis problem: Decide if the there is a program $P : I^* \longrightarrow J$ realizing φ, and construct one if possible.

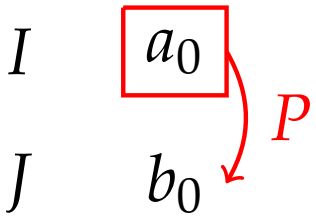
I a_0

J

Problem Setting

- Infinite input sequence α over I
- Infinite output sequence β over J
- Formula φ specifying the "good" sequences from $(I \times J)^\omega$

Synthesis problem: Decide if the there is a program $P : I^* \longrightarrow J$ realizing φ, and construct one if possible.



Problem Setting

- Infinite input sequence α over I
- Infinite output sequence β over J
- Formula φ specifying the "good" sequences from $(I \times J)^\omega$

Synthesis problem: Decide if the there is a program $P : I^* \longrightarrow J$ realizing φ, and construct one if possible.

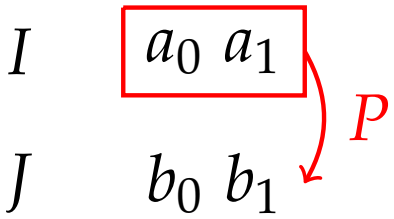
I $a_0\ a_1$

J b_0

Problem Setting

- Infinite input sequence α over I
- Infinite output sequence β over J
- Formula φ specifying the "good" sequences from $(I \times J)^\omega$

Synthesis problem: Decide if the there is a program $P : I^* \longrightarrow J$ realizing φ, and construct one if possible.



Problem Setting

- Infinite input sequence α over I
- Infinite output sequence β over J
- Formula φ specifying the "good" sequences from $(I \times J)^\omega$

Synthesis problem: Decide if the there is a program $P : I^* \longrightarrow J$ realizing φ, and construct one if possible.

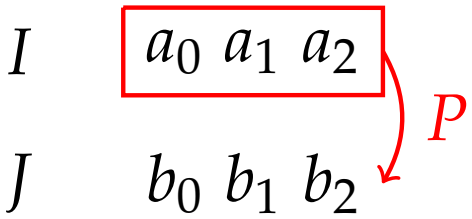
I $\quad a_0\ a_1\ a_2$

J $\quad b_0\ b_1$

Problem Setting

- Infinite input sequence α over I
- Infinite output sequence β over J
- Formula φ specifying the "good" sequences from $(I \times J)^\omega$

Synthesis problem: Decide if the there is a program $P : I^* \to J$ realizing φ, and construct one if possible.



Problem Setting

- Infinite input sequence α over I
- Infinite output sequence β over J
- Formula φ specifying the "good" sequences from $(I \times J)^\omega$

Synthesis problem: Decide if the there is a program $P : I^* \longrightarrow J$ realizing φ, and construct one if possible.

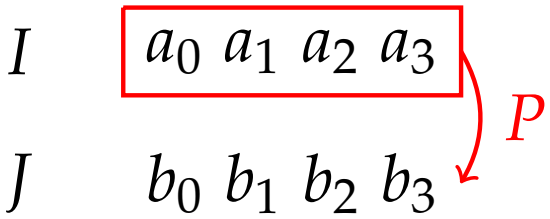
I $\quad a_0\ a_1\ a_2\ a_3$

J $\quad b_0\ b_1\ b_2$

Problem Setting

- Infinite input sequence α over I
- Infinite output sequence β over J
- Formula φ specifying the "good" sequences from $(I \times J)^\omega$

Synthesis problem: Decide if the there is a program $P : I^* \longrightarrow J$ realizing φ, and construct one if possible.



Problem Setting

- Infinite input sequence α over I
- Infinite output sequence β over J
- Formula φ specifying the "good" sequences from $(I \times J)^\omega$

Synthesis problem: Decide if the there is a program $P : I^* \longrightarrow J$ realizing φ, and construct one if possible.

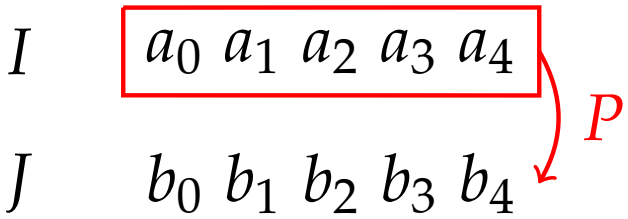
I $\quad a_0\ a_1\ a_2\ a_3\ a_4$

J $\quad b_0\ b_1\ b_2\ b_3$

Problem Setting

- Infinite input sequence α over I
- Infinite output sequence β over J
- Formula φ specifying the "good" sequences from $(I \times J)^\omega$

Synthesis problem: Decide if the there is a program $P : I^* \longrightarrow J$ realizing φ, and construct one if possible.



Problem Setting

- Infinite input sequence α over I
- Infinite output sequence β over J
- Formula φ specifying the "good" sequences from $(I \times J)^\omega$

Synthesis problem: Decide if the there is a program $P : I^* \to J$ realizing φ, and construct one if possible.

$$\begin{array}{ll} I & a_0\ a_1\ a_2\ a_3\ a_4 \cdots \\ J & b_0\ b_1\ b_2\ b_3\ b_4 \cdots \end{array} \models \varphi$$

Problem Setting

- Infinite input sequence α over I
- Infinite output sequence β over J
- Formula φ specifying the "good" sequences from $(I \times J)^\omega$

Synthesis problem: Decide if the there is a program $P : I^* \to J$ realizing φ, and construct one if possible.

$$\begin{array}{ll} I & a_0\ a_1\ a_2\ a_3\ a_4 \cdots \\ J & b_0\ b_1\ b_2\ b_3\ b_4 \cdots \end{array} \models \varphi$$

Finite automata solution: P is a finite state machine (S, I, s_0, δ, f) with output function $f : S \times I \to J$.

Sequential Calculus

Monadic second-order logic (MSO) over the structure $(\mathbb{N}, +1, <)$
(first-order logic plus quantification over sets of elements)

Example: $I = \{a, a'\}$ and $J = \{b, b'\}$

$$\underbrace{\forall x \Big(a(x) \rightarrow \exists y > x \, (b(y)) \Big)}_{\text{each input } a \text{ later followed by output } b} \wedge \underbrace{\forall x \exists y > x \, \big(b'(y)\big)}_{\text{infinitely often output } b'}$$

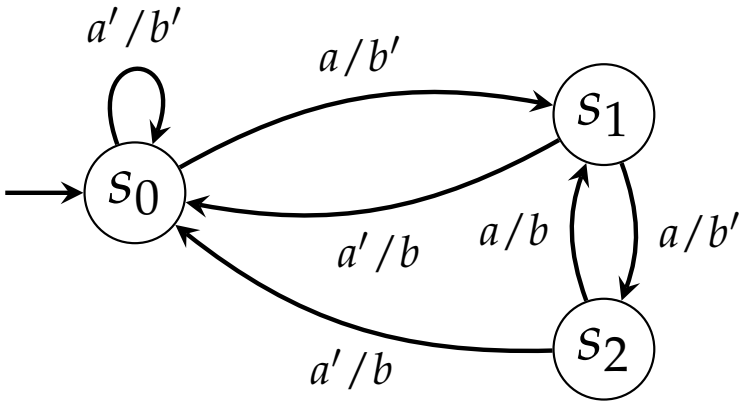
Sequential Calculus

Monadic second-order logic (MSO) over the structure $(\mathbb{N}, +1, <)$
(first-order logic plus quantification over sets of elements)

Example: $I = \{a, a'\}$ and $J = \{b, b'\}$

$$\underbrace{\forall x \Big(a(x) \rightarrow \exists y > x \, (b(y)) \Big)}_{\text{each input } a \text{ later followed by output } b} \wedge \underbrace{\forall x \exists y > x \, \big(b'(y)\big)}_{\text{infinitely often output } b'}$$

Finite automata solutions:



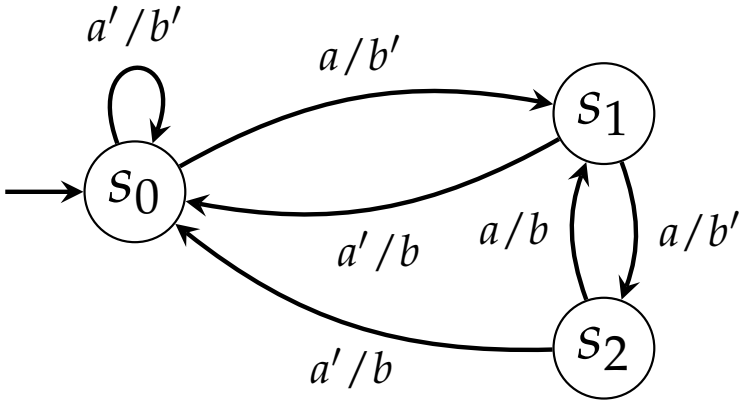
Sequential Calculus

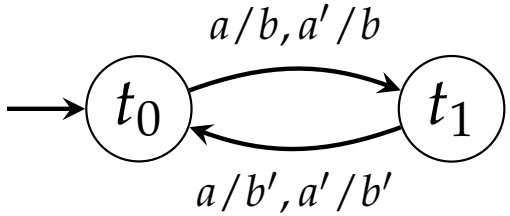
Monadic second-order logic (MSO) over the structure $(\mathbb{N}, +1, <)$
(first-order logic plus quantification over sets of elements)

Example: $I = \{a, a'\}$ and $J = \{b, b'\}$

$$\underbrace{\forall x \Big(a(x) \rightarrow \exists y > x \, (b(y)) \Big)}_{\text{each input } a \text{ later followed by output } b} \wedge \underbrace{\forall x \exists y > x \, \big(b'(y)\big)}_{\text{infinitely often output } b'}$$

Finite automata solutions:

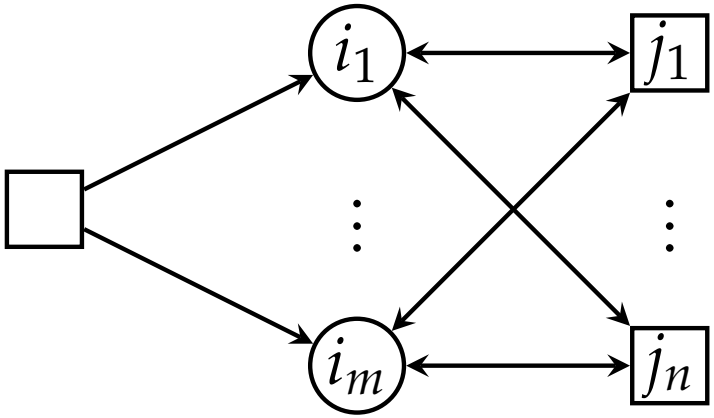




Game-Theoretic Formulation

$I = \{i_1, \ldots, i_m\}$ and $J = \{j_1, \ldots, j_n\}$

Two player game on a graph:



- Players Box and Circle move a token along the edges.
- Winning condition defines the vertex sequences that are winning for player Circle.

General Setting

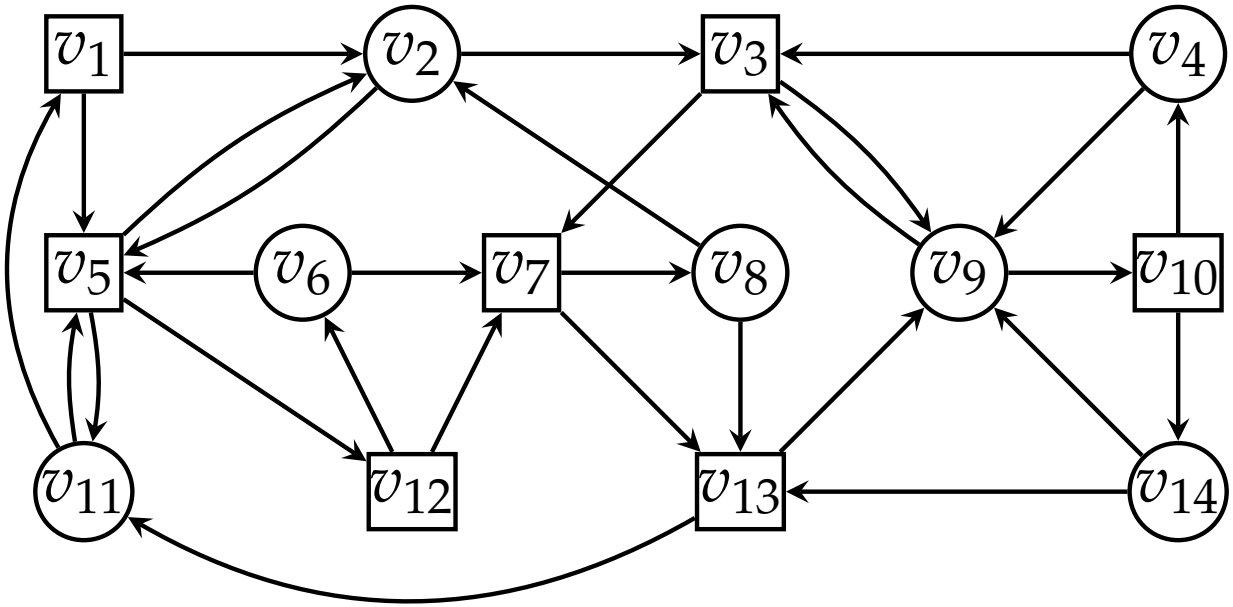
$G = (V_\circ, V_\square, E, c)$

- $V_\circ$: vertices of player Circle
- $V_\square$: vertices of player Box
- $E \subseteq V \times V$: edges with $V = V_\circ \cup V_\square$
- $Win \subseteq V^\omega$ winning condition for Circle

General Setting

$G = (V_\circ, V_\square, E, c)$

- $V_\circ$: vertices of player Circle
- $V_\square$: vertices of player Box
- $E \subseteq V \times V$: edges with $V = V_\circ \cup V_\square$
- $Win \subseteq V^\omega$ winning condition for Circle



Strategies

A *strategy for Circle* is a function

$$\sigma : V^* V_\circ \to V$$

with $\sigma(xv) = v'$ implies $(v, v') \in E$

Special strategies:

- *Computable*: The function $\sigma : V^* V_\circ \to V$ is computable.
- *Positional*: The strategy only depends on the current vertex (not on the past), i.e., $\sigma : V_\circ \to V$.
- *Finite memory*: The strategy is implemented by a deterministic finite automaton that reads the vertices of the play:
  $\mathcal{S} = (S, V, s_{in}, \delta, \sigma)$
  The strategy moves depend on the current vertex and the state of the automaton: $\sigma : S \times V_\circ \to V$.

Solving Infinite Games on Finite Graphs

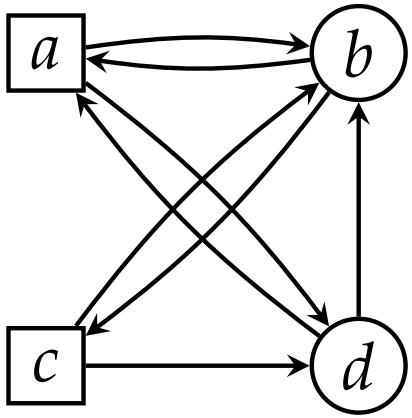
A Simpler Scenario

Goal: general mechanism for transforming winning conditions

Example:

Game graph with vertices $V = \{a, b, c, d\}$

Winning condition: Circle wins if the play never matches the regular expression r (over the alphabet V)



$$V^*c(a+b)^*cV^* + V^*d(a+b)^*dV^*$$

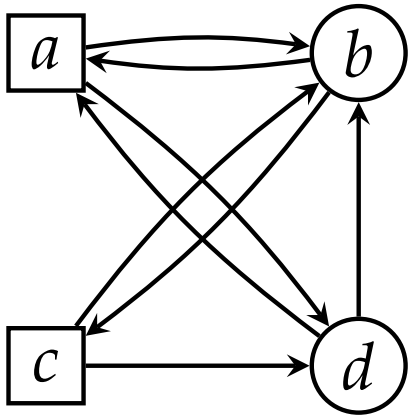
A Simpler Scenario

Goal: general mechanism for transforming winning conditions

Example:

Game graph with vertices $V = \{a, b, c, d\}$

Winning condition: Circle wins if the play never matches the regular expression r (over the alphabet V)



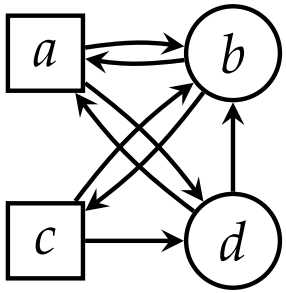
$$V^*c(a+b)^*cV^* + V^*d(a+b)^*dV^*$$

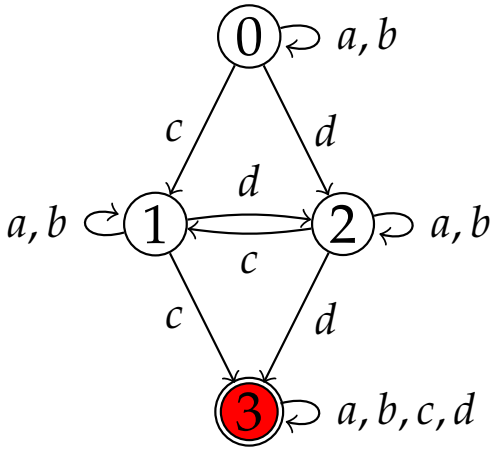
How to solve such games in general?

Safety Game

Product of game graph and DFA for r: the automaton reads the play

Eva wins if she can avoid the final states of the DFA.

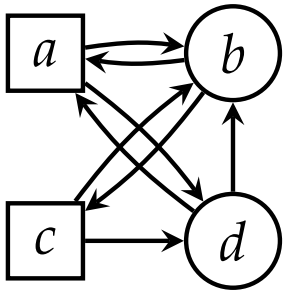


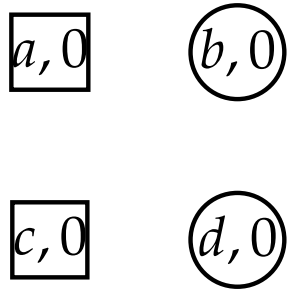


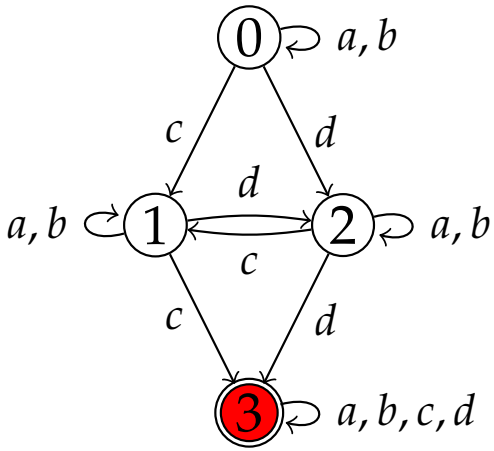
Safety Game

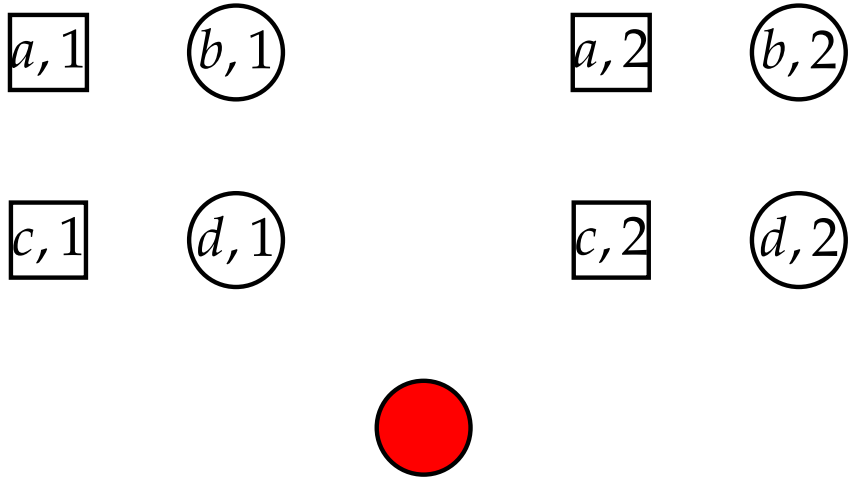
Product of game graph and DFA for r: the automaton reads the play

Eva wins if she can avoid the final states of the DFA.





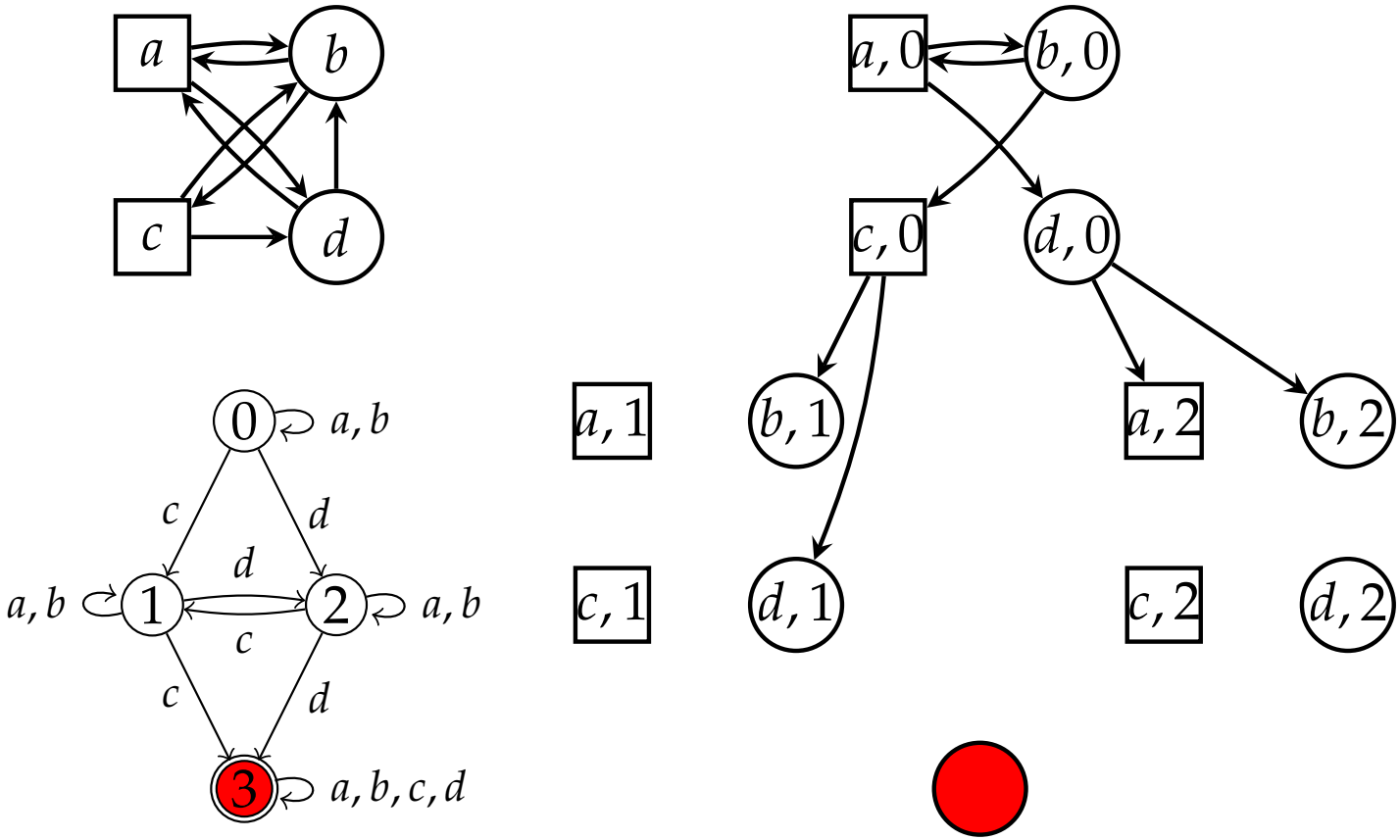




Safety Game

Product of game graph and DFA for r: the automaton reads the play

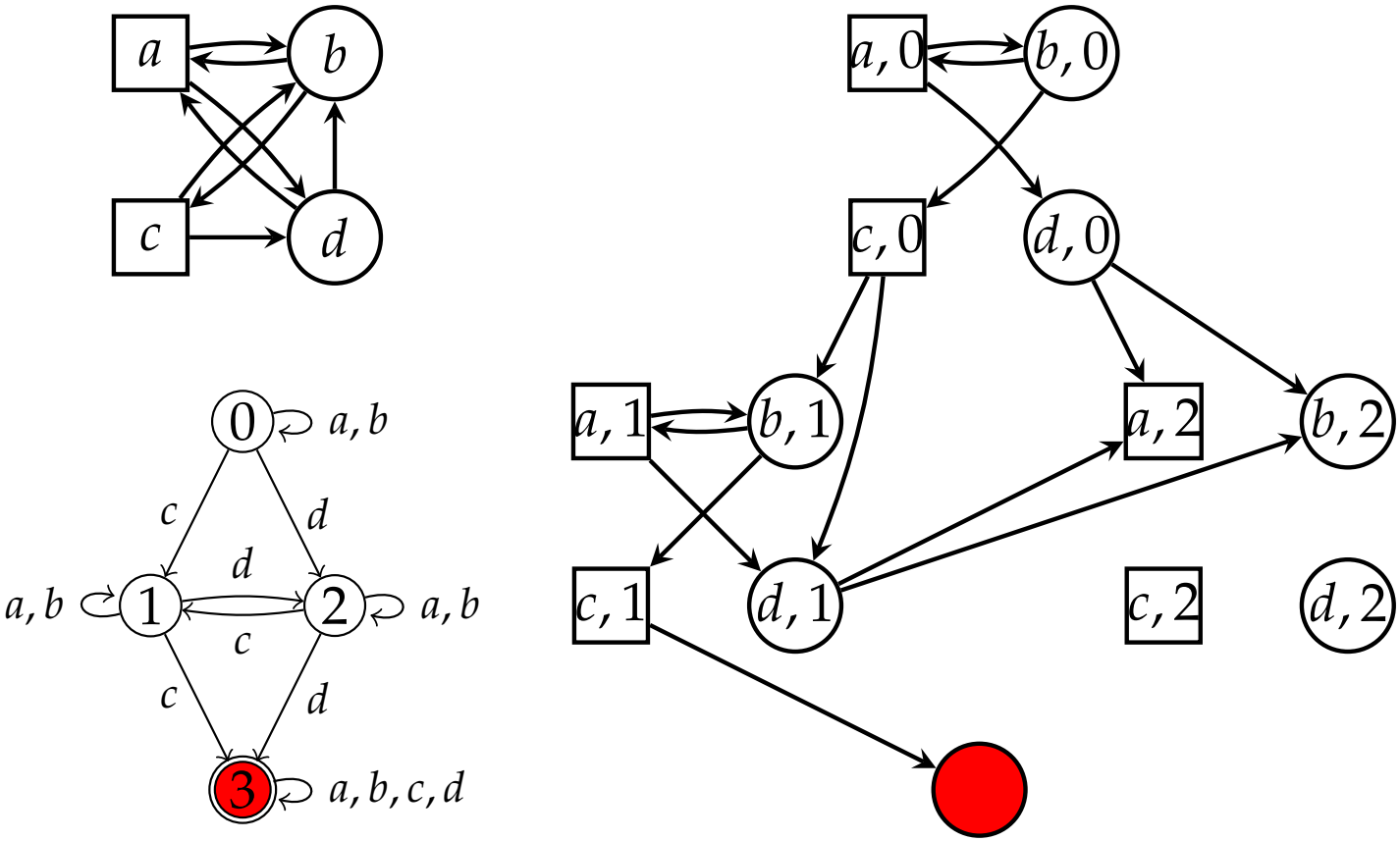
Eva wins if she can avoid the final states of the DFA.



Safety Game

Product of game graph and DFA for r: the automaton reads the play

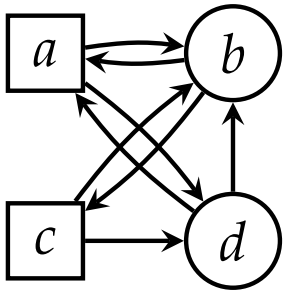
Eva wins if she can avoid the final states of the DFA.

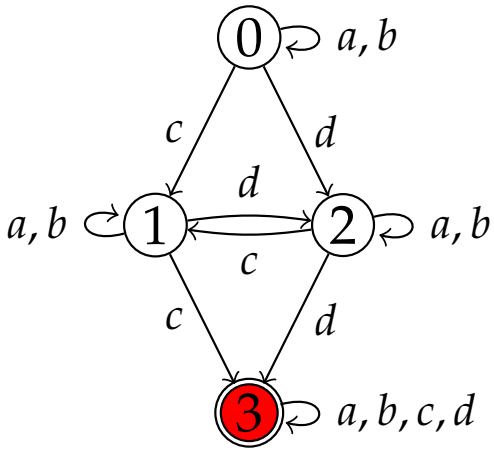


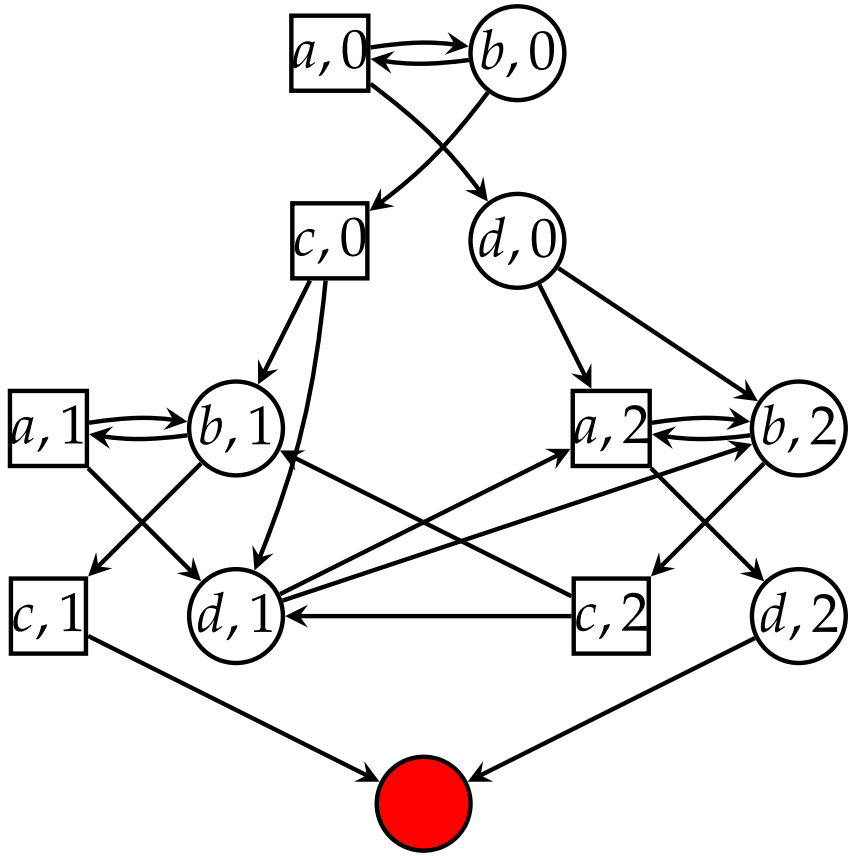
Safety Game

Product of game graph and DFA for r: the automaton reads the play

Eva wins if she can avoid the final states of the DFA.



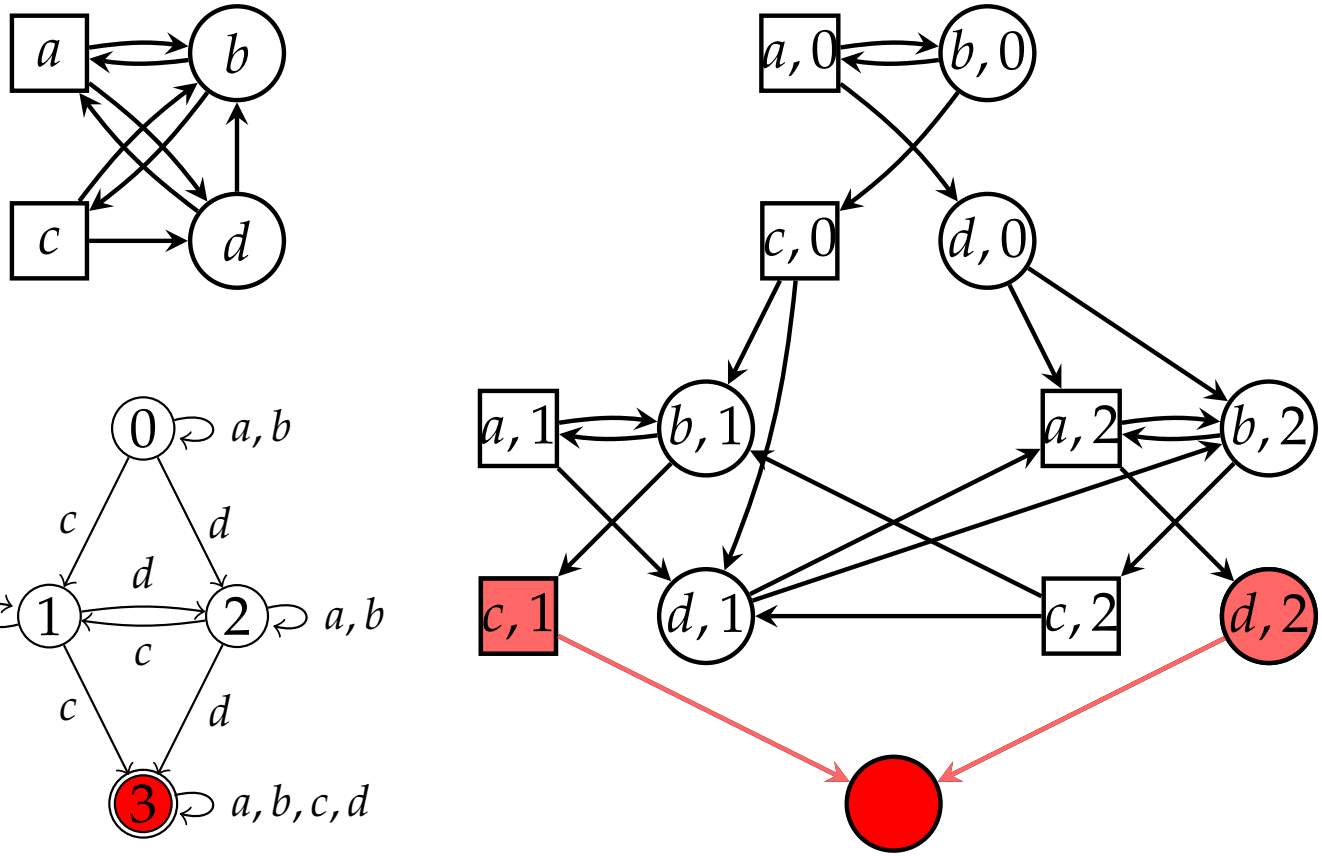




Safety Game

Product of game graph and DFA for r: the automaton reads the play

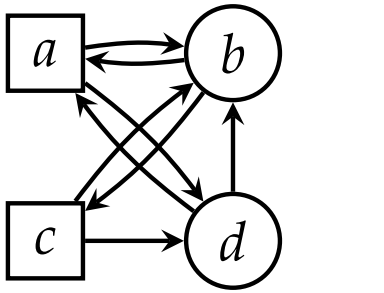
Eva wins if she can avoid the final states of the DFA.

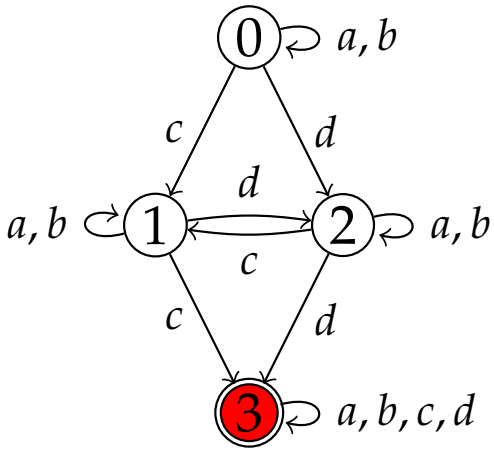


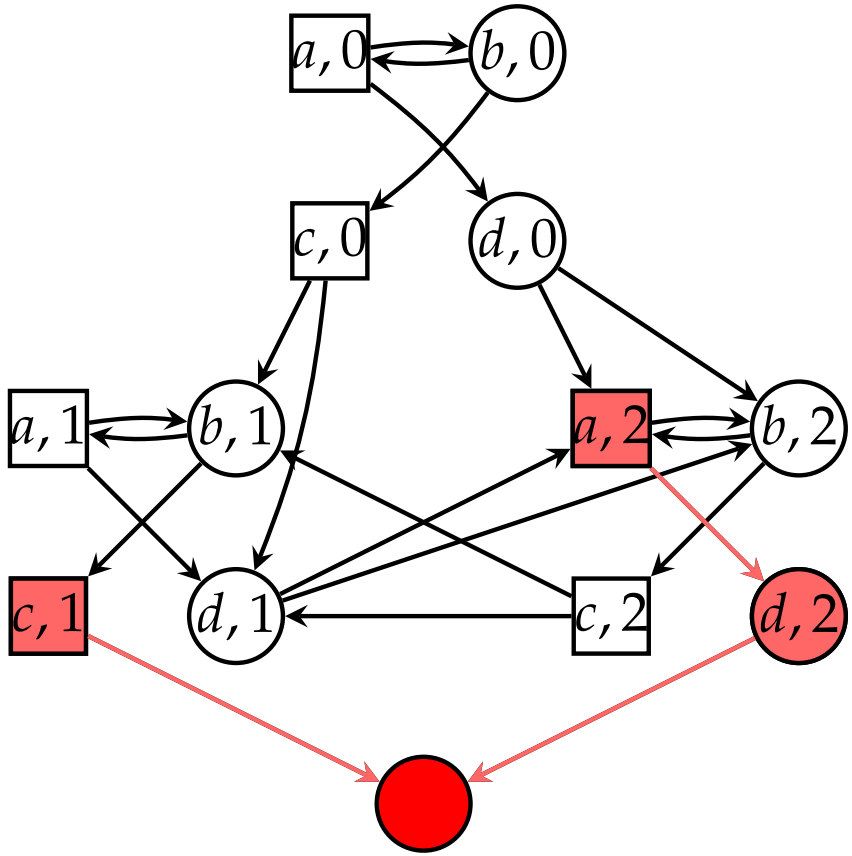
Safety Game

Product of game graph and DFA for r: the automaton reads the play

Eva wins if she can avoid the final states of the DFA.



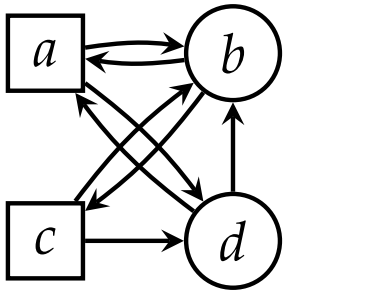


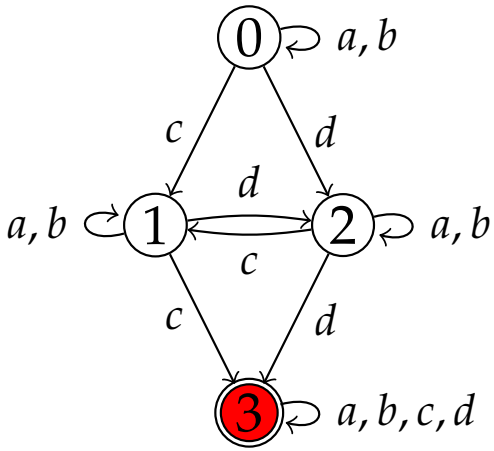


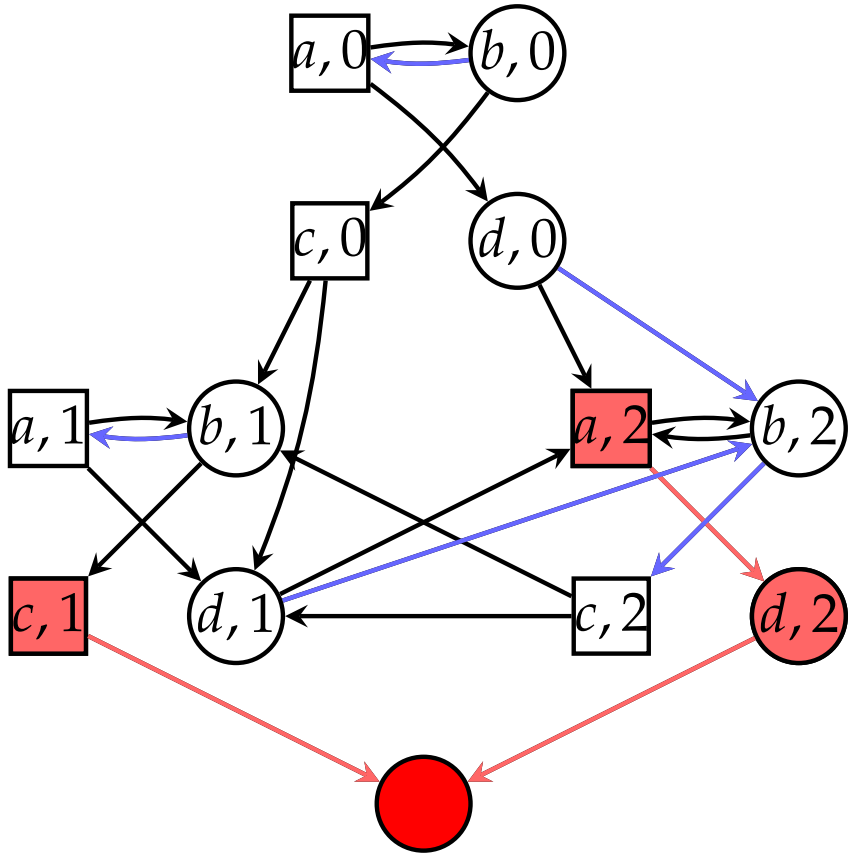
Safety Game

Product of game graph and DFA for r: the automaton reads the play

Eva wins if she can avoid the final states of the DFA.



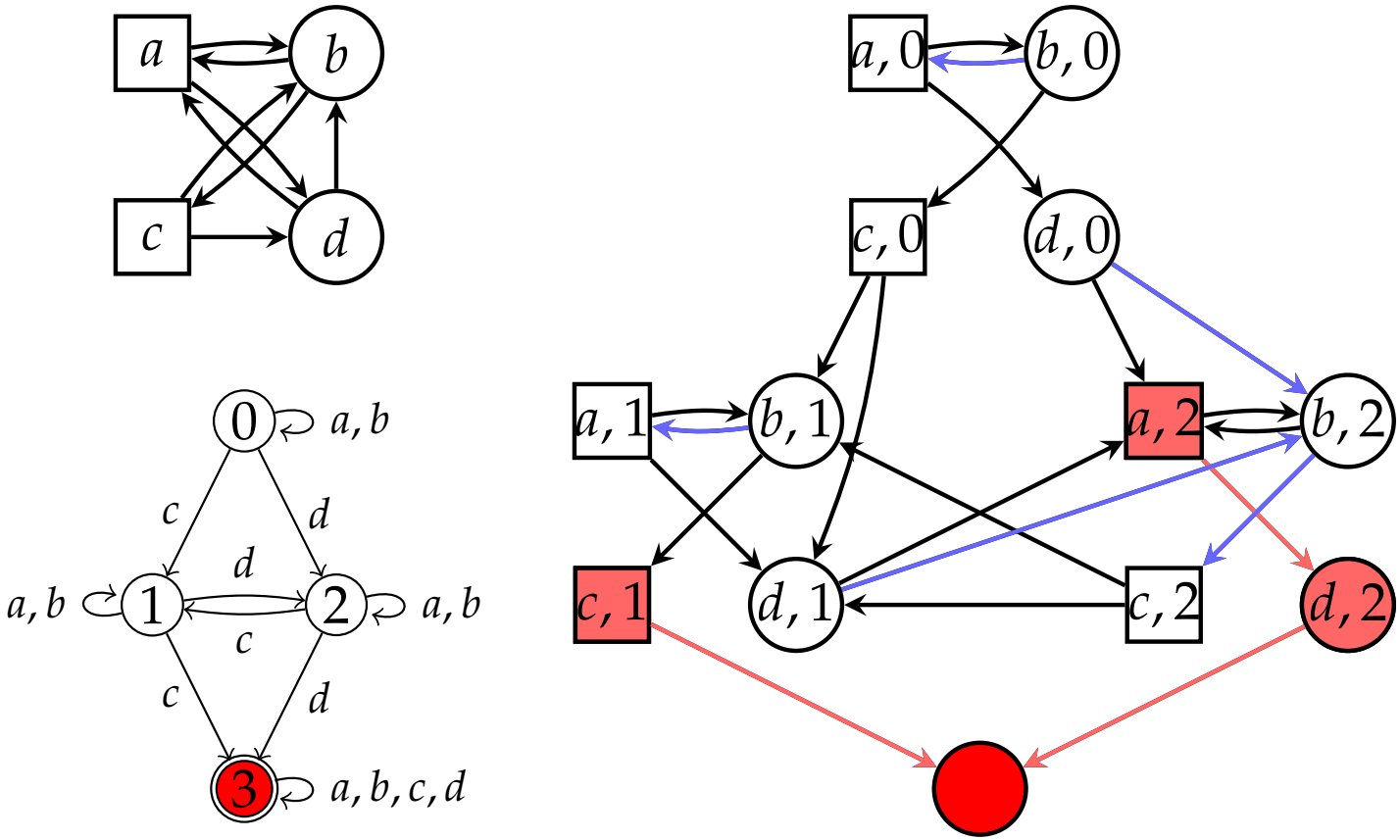




Safety Game

Product of game graph and DFA for r: the automaton reads the play

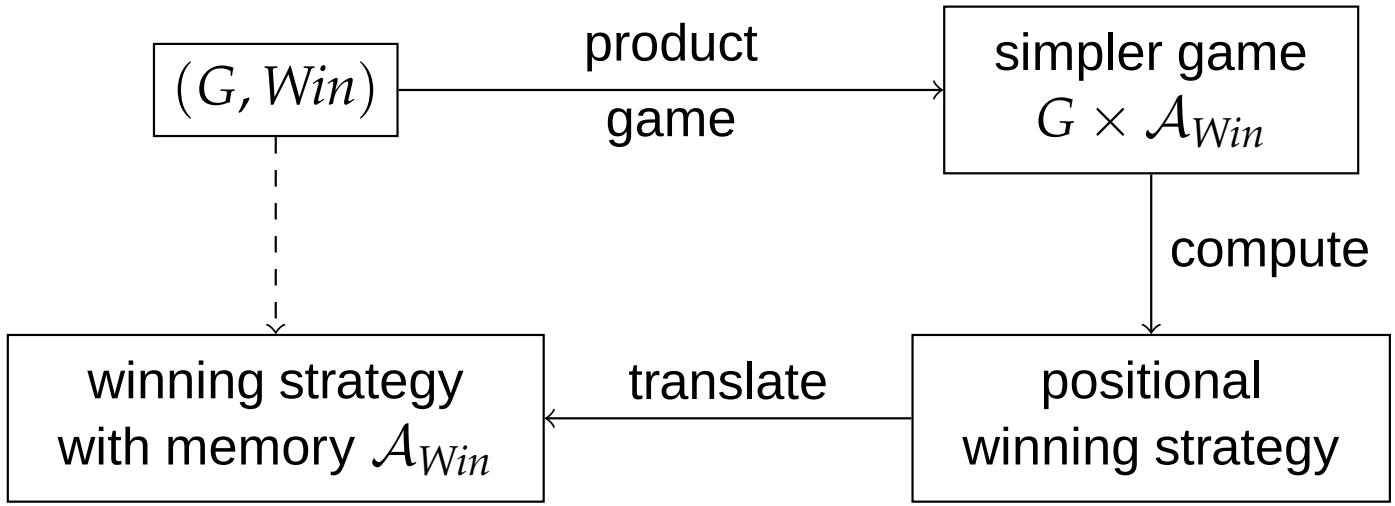
Eva wins if she can avoid the final states of the DFA.



Strategy for Eva with three memory states.

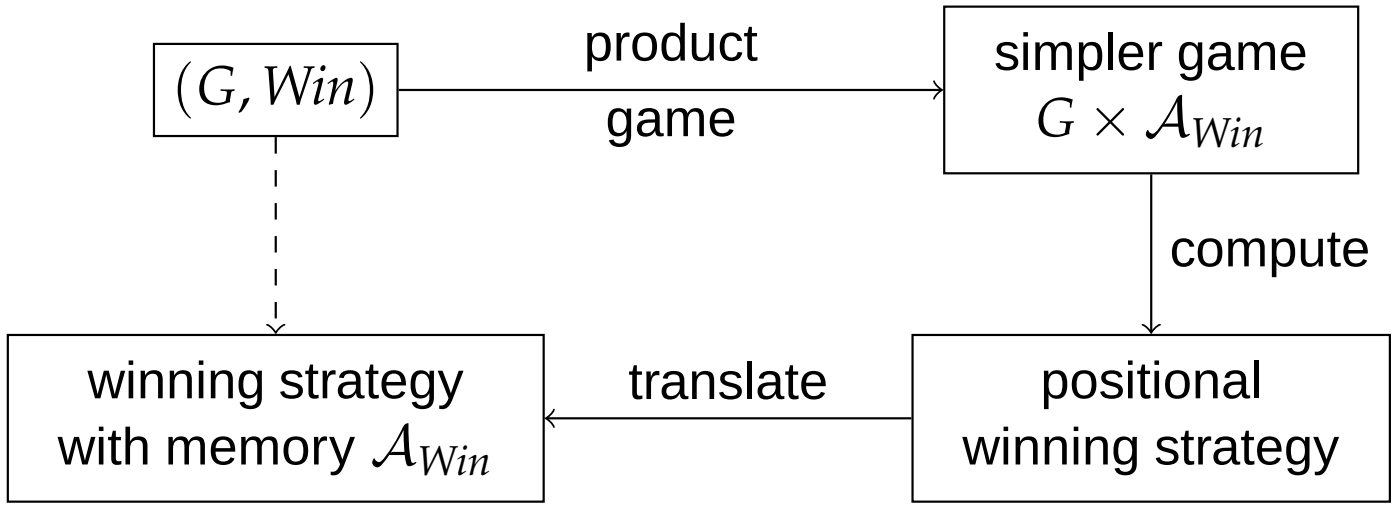
Summary of the Method

- Construct a deterministic automaton for the specification.
- Take the product with the game graph $\rightsquigarrow$ acceptance condition of the automaton determines the winning condition.
- Obtain a simpler game: larger game graph but a simpler winning condition.
- Solve the simple game and transfer the strategy back to the original game.



Summary of the Method

- Construct a deterministic automaton for the specification.
- Take the product with the game graph $\rightsquigarrow$ acceptance condition of the automaton determines the winning condition.
- Obtain a simpler game: larger game graph but a simpler winning condition.
- Solve the simple game and transfer the strategy back to the original game.



How to handle MSO winning conditions?

Büchi Automata

A *Büchi automaton* is of the form $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, F)$ with the same components as a standard nondeterministic finite automaton.

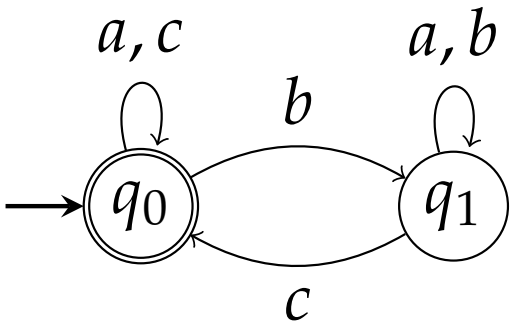
- A run is an infinite state sequence that starts in the initial state and is compatible with the transitions.

Büchi Automata

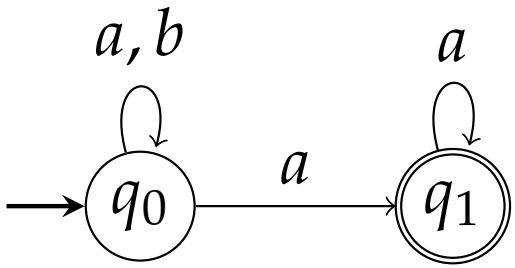
A *Büchi automaton* is of the form $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, F)$ with the same components as a standard nondeterministic finite automaton.

- A run is an infinite state sequence that starts in the initial state and is compatible with the transitions.
- A run is accepting if it contains infinitely often a state from F.
- The language $L(\mathcal{A})$ accepted by $\mathcal{A}$ is the set of all infinite words for which the automaton has an accepting run.

Examples:



"if b then later c"



"finitely many b"

Logic to Automata

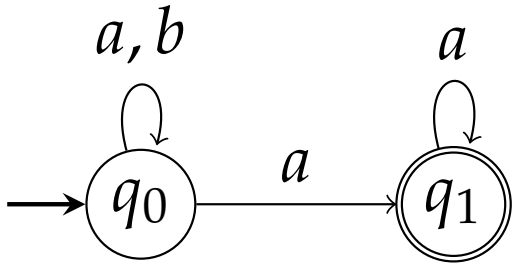
Theorem (Büchi'62). For each MSO formula over infinite words there is an equivalent nondeterministic Büchi automaton.

Logic to Automata

Theorem (Büchi'62). For each MSO formula over infinite words there is an equivalent nondeterministic Büchi automaton.

Deterministic Büchi automata are not enough:

There is no deterministic Büchi automaton for "finitely many b"

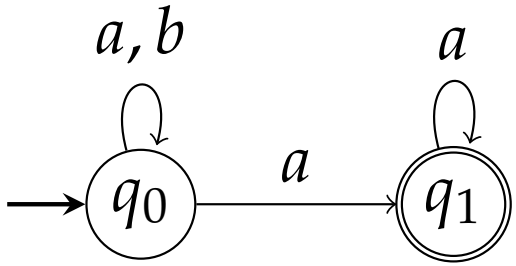


Logic to Automata

Theorem (Büchi'62). For each MSO formula over infinite words there is an equivalent nondeterministic Büchi automaton.

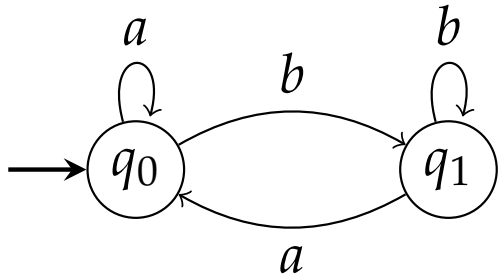
Deterministic Büchi automata are not enough:

There is no deterministic Büchi automaton for "finitely many b"



A deterministic automaton with a different acceptance condition:

visit q_1 only finitely often!



Parity Automata

It turns out that Boolean combinations of states being visited infinitely or finitely often are sufficient for deterministic automata (Muller conditions).

Parity Automata

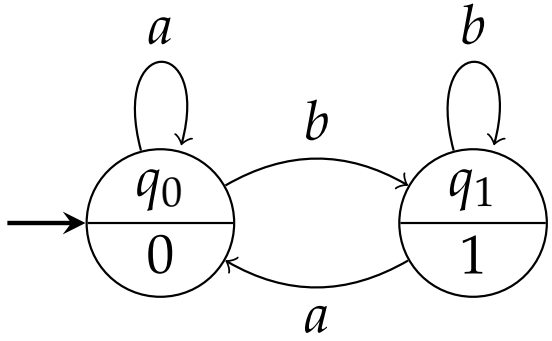
It turns out that Boolean combinations of states being visited infinitely or finitely often are sufficient for deterministic automata (Muller conditions).

The parity condition is a specific normal form where the states are hierarchically ordered for the acceptance condition.

A *parity automaton* is of the form $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, pri)$ with a priority mapping $pri : Q \to \mathbb{N}$.

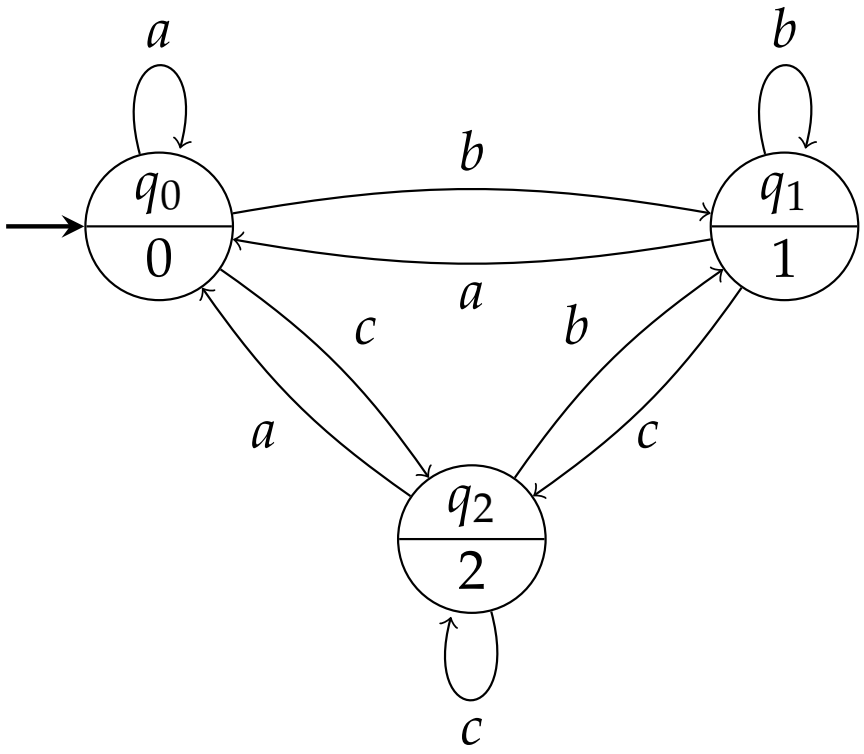
A run is accepting if the maximal priority seen infinitely often is even.

Parity automaton for “finitely many b”:



Further Example

infinitely many c or finitely many b



Logic to Parity Games

Theorem (Büchi 1962). For each MSO formula over infinite words there is an equivalent nondeterministic Büchi automaton.

Theorem (McNaughton 1966, Mostowski 1983). For each nondeterministic Büchi automaton there is an equivalent deterministic parity automaton.

Logic to Parity Games

Theorem (Büchi 1962). For each MSO formula over infinite words there is an equivalent nondeterministic Büchi automaton.

Theorem (McNaughton 1966, Mostowski 1983). For each nondeterministic Büchi automaton there is an equivalent deterministic parity automaton.

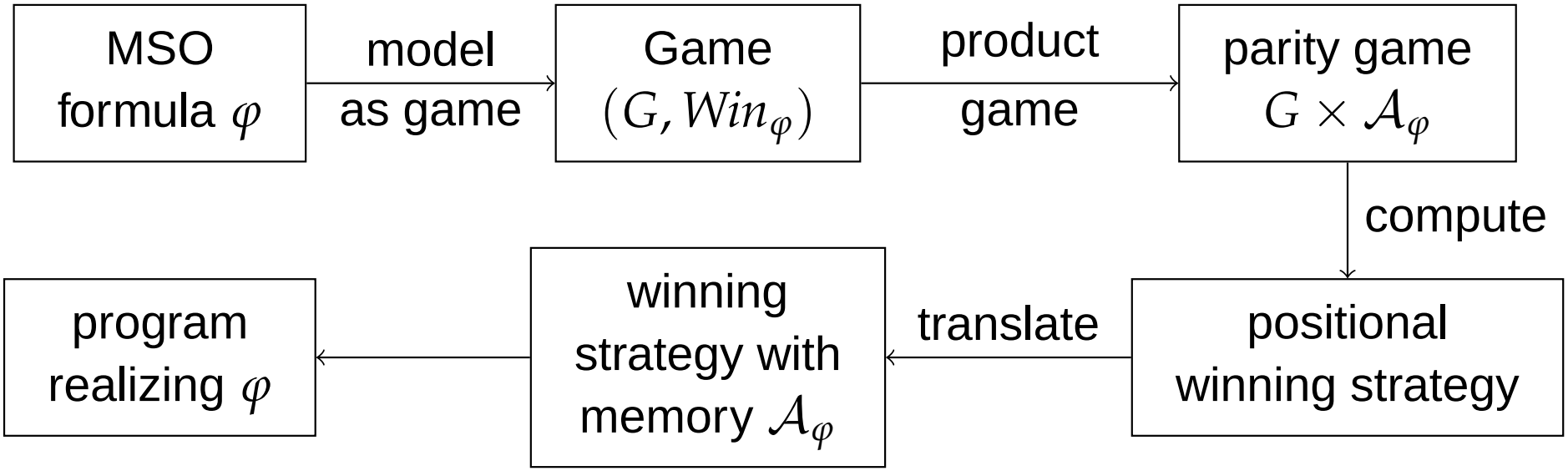
Theorem (Emerson/Jutla 1988, Mostowski 1991). Parity games are determined with positional strategies.

Büchi-Landweber Theorem

Theorem (Büchi/Landweber 1969). The synthesis problem for MSO specifications is solvable. If a solution exists, then a finite automaton solution can be constructed.

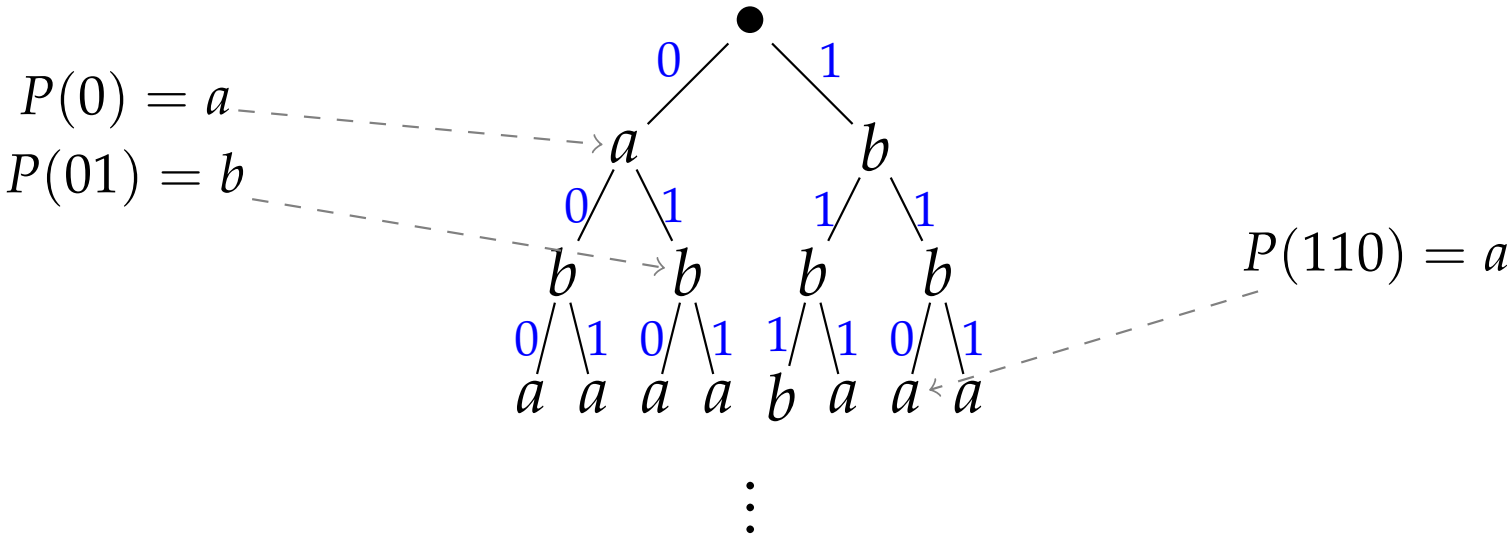
Büchi-Landweber Theorem

Theorem (Büchi/Landweber 1969). The synthesis problem for MSO specifications is solvable. If a solution exists, then a finite automaton solution can be constructed.



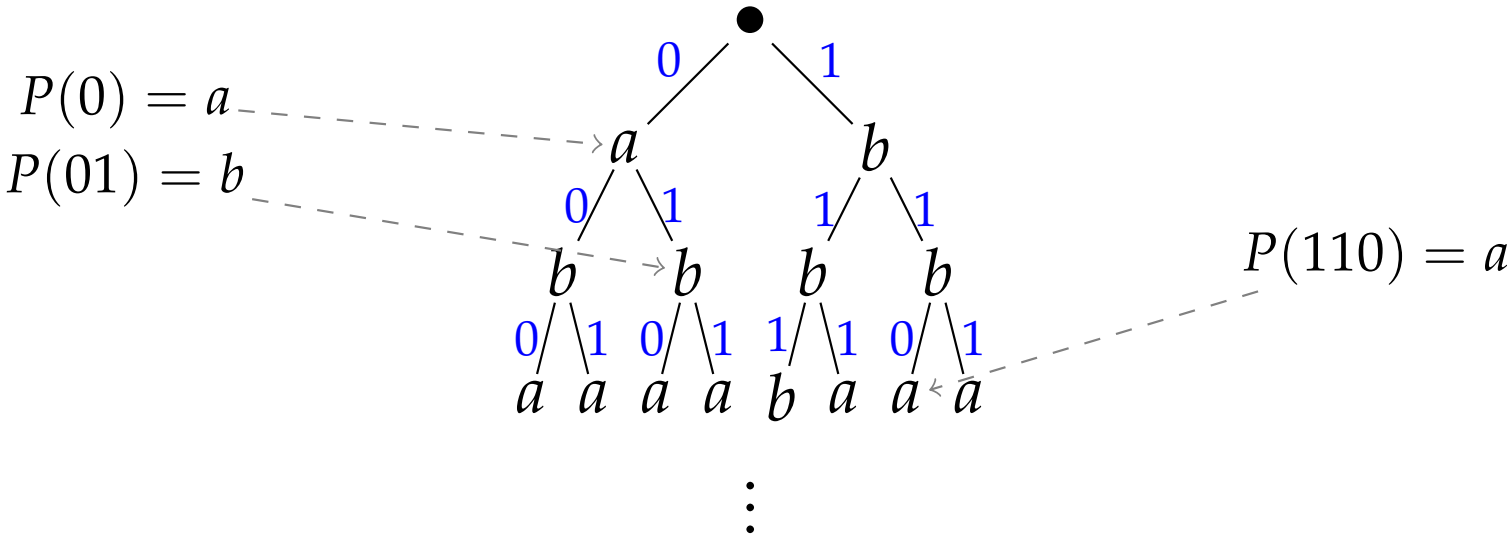
Rabin's Approach

- We are looking for a program $P : I^* \to J$
- Such a program P can be described by an infinite tree.
  Example: $I = \{0, 1\}$ and $J = \{a, b\}$



Rabin's Approach

- We are looking for a program $P : I^* \longrightarrow J$
- Such a program P can be described by an infinite tree.
  Example: $I = \{0,1\}$ and $J = \{a,b\}$



- The synthesis problem for MSO specifications can be reduced to satisfiability of MSO over infinite trees.

Theorem (Rabin 1969). Satisfiability of MSO over infinite trees is decidable.

Variations of the Problem

Classes of effectively solvable games?

result in game theory. More generally, the following type of game problem is naturally suggested by automata theory. Given a class of games G: (1) can one effectively decide, for any $\mathfrak{G} \in G$, which player has a winning strategy? (2) Just how simple winning strategies do exist for games in G? For example, is there a recursive or even a finite automata winning strategy for $\mathfrak{G} \in G$? This general problem was

Variations of the Problem

Classes of effectively solvable games?

result in game theory. More generally, the following type of game problem is naturally suggested by automata theory. Given a class of games G: (1) can one effectively decide, for any $\mathfrak{G} \in G$, which player has a winning strategy? (2) Just how simple winning strategies do exist for games in G? For example, is there a recursive or even a finite automata winning strategy for $\mathfrak{G} \in G$? This general problem was

More general solutions (programs)?

PROBLEM. Can one algorithmically determine whether or not for a condition $\mathfrak{G}(X, Y)$ stated in SC there exists an h such that $\mathfrak{G}$ admits an h-shift, but no $(h+1)$-shift solution for Y?

Outline

1. Büchi-Landweber Theorem

2. Extensions
   - Pushdown Games
   - Delay Games

3. Implementations

Non-Regular Specifications

MSO

$\longrightarrow$ nondeterministic Büchi automata

$\longrightarrow$ deterministic Muller/parity automata

$\longrightarrow$ Muller/parity games on finite graphs

What happens if we take pushdown ω-automata?

Non-Regular Specifications

MSO
$\longrightarrow$ nondeterministic Büchi automata
$\longrightarrow$ deterministic Muller/parity automata
$\longrightarrow$ Muller/parity games on finite graphs

What happens if we take pushdown ω-automata?

Undecidability results for nondeterminisitc pushdown automata on finite words easily yield:

Theorem. The synthesis problem for specifications given by nondeterministic pushdown automata is undecidable.

Non-Regular Specifications

MSO
$\longrightarrow$ nondeterministic Büchi automata
$\longrightarrow$ deterministic Muller/parity automata
$\longrightarrow$ Muller/parity games on finite graphs

What happens if we take pushdown ω-automata?

Undecidability results for nondeterminisitc pushdown automata on finite words easily yield:

Theorem. The synthesis problem for specifications given by nondeterministic pushdown automata is undecidable.

deterministic Muller/parity pushdown automata
$\longrightarrow$ Muller/parity games on pushdown graphs

Pushdown Games

control states $P_\circ = \{p_0\}$ and $P_\square = \{p_1, p_2\}$

stack alphabet $\{a, b, c\}$

pushdown rules $\Delta = \left\{ \begin{array}{l} (p_0 a \to p_1 b a),\ (p_0 a \to p_0),\ (p_0 b \to p_0), \\ (p_1 b \to p_2 c a),\ (p_1 b \to p_1 b b),\ (p_2 c \to p_0 b) \end{array} \right\}$

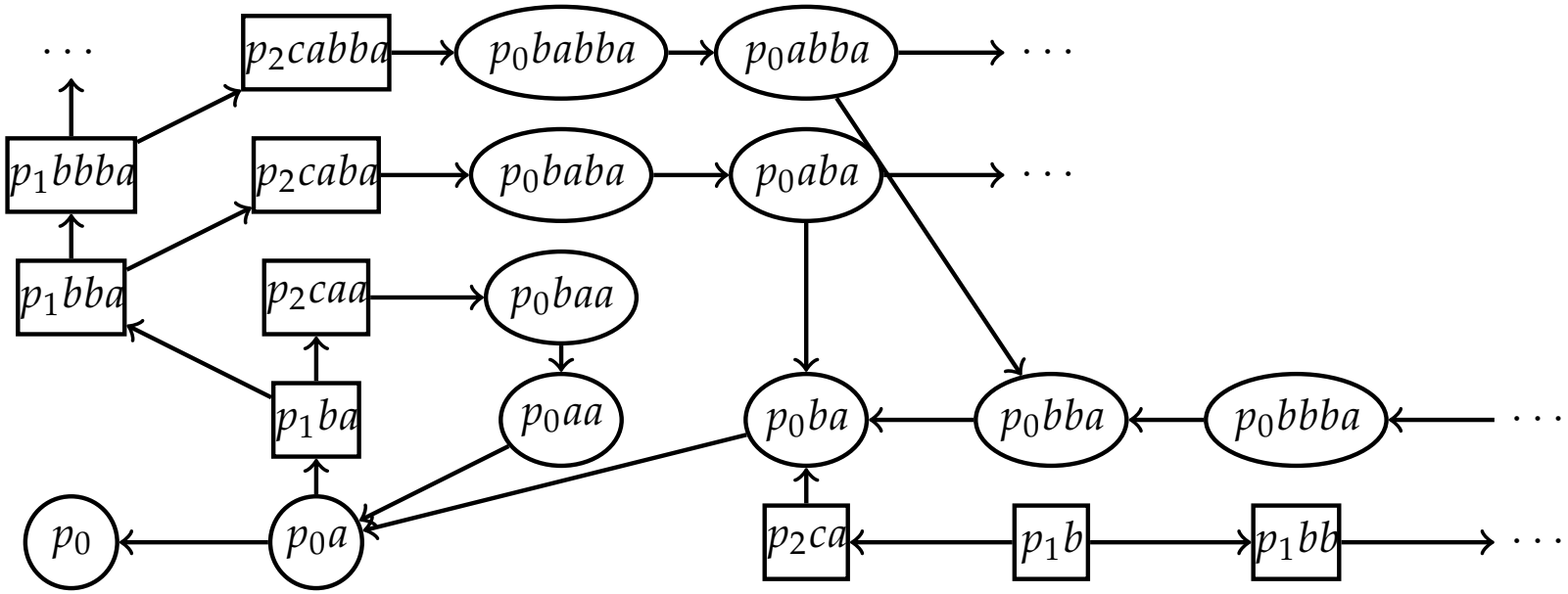
Pushdown Games

control states $P_\circ = \{p_0\}$ and $P_\square = \{p_1, p_2\}$

stack alphabet $\{a, b, c\}$

pushdown rules $\Delta = \left\{ \begin{array}{l} (p_0 a \to p_1 ba),\ (p_0 a \to p_0),\ (p_0 b \to p_0), \\ (p_1 b \to p_2 ca),\ (p_1 b \to p_1 bb),\ (p_2 c \to p_0 b) \end{array} \right\}$

Part of the game graph $G_{\mathcal{P}}$:



Winning condition: Muller/parity condition on the states

Pushdown Games

Theorem (Walukiewicz 1996). Infinite games on pushdown graphs with regular winning conditions (Muller/parity) are determined with pushdown winning strategies.

Pushdown Games

Theorem (Walukiewicz 1996). Infinite games on pushdown graphs with regular winning conditions (Muller/parity) are determined with pushdown winning strategies.

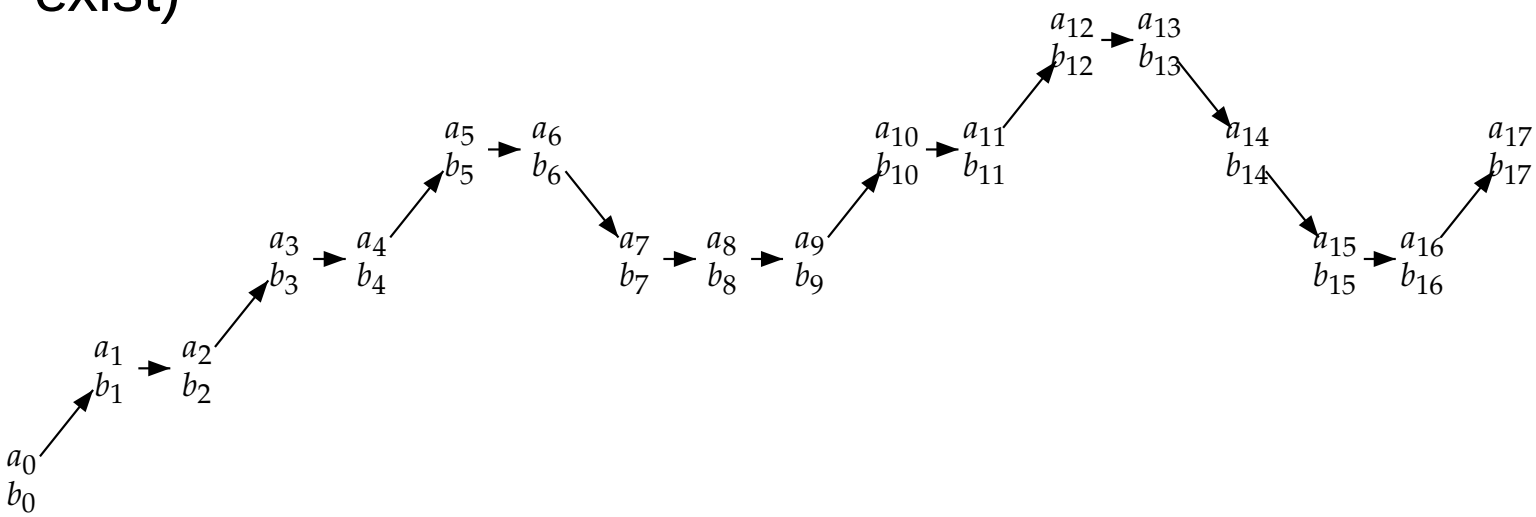
Idea: Reduction to a game on a finite graph (exponential in the size of the pushdown automaton)

A Logic for Pushdown Specifications

- Instead of viewing the symbols in $\int$ as output symbols, we can view them as actions in a computation.
- If we model recursive computations, then some actions denote function/procedure calls and some denote returns.
- This yields a natural relation on word positions that relates the call positions with their corresponding return positions (if they exist)

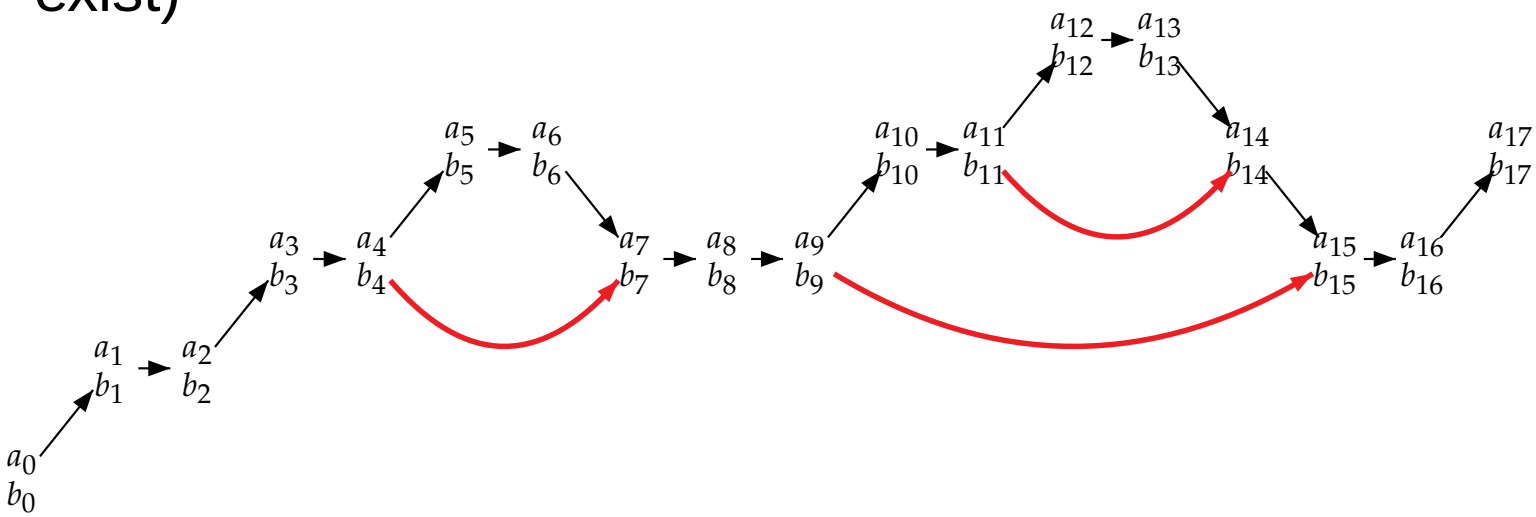
A Logic for Pushdown Specifications

- Instead of viewing the symbols in J as output symbols, we can view them as actions in a computation.
- If we model recursive computations, then some actions denote function/procedure calls and some denote returns.
- This yields a natural relation on word positions that relates the call positions with their corresponding return positions (if they exist)



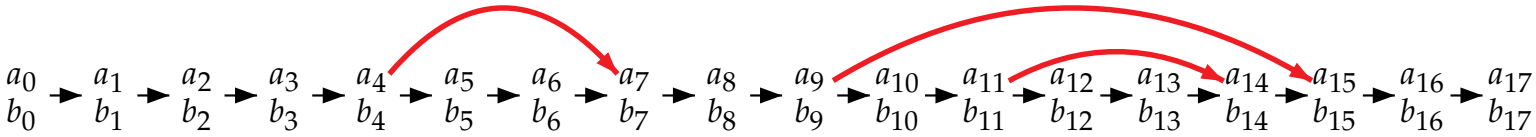
A Logic for Pushdown Specifications

- Instead of viewing the symbols in J as output symbols, we can view them as actions in a computation.
- If we model recursive computations, then some actions denote function/procedure calls and some denote returns.
- This yields a natural relation on word positions that relates the call positions with their corresponding return positions (if they exist)



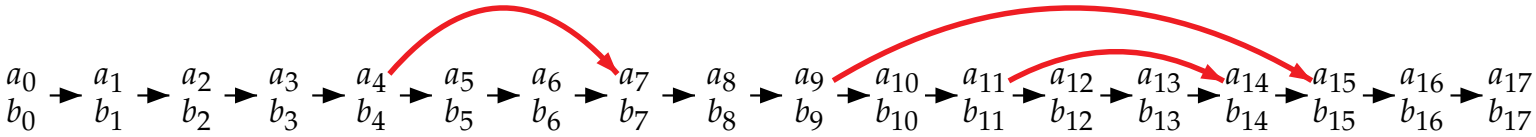
A Logic for Pushdown Specifications

- Instead of viewing the symbols in J as output symbols, we can view them as actions in a computation.
- If we model recursive computations, then some actions denote function/procedure calls and some denote returns.
- This yields a natural relation on word positions that relates the call positions with their corresponding return positions (if they exist)



A Logic for Pushdown Specifications

- Instead of viewing the symbols in J as output symbols, we can view them as actions in a computation.
- If we model recursive computations, then some actions denote function/procedure calls and some denote returns.
- This yields a natural relation on word positions that relates the call positions with their corresponding return positions (if they exist)



Nested word MSO: MSO over the structure $(\mathbb{N}, +1, <, R)$

Nested Word MSO Specifications

Theorem (L.,Madhusudan,Serre 2004).

- Nested word MSO specifications can be translated into (a specific class of) deterministic pushdown ω-automata.
- The synthesis problem for nested word MSO specifications is solvable and pushdown strategies can be constructed.

Further Results

- Pushdown games with winning condition “there is a configuration that is visited infinitely often” (Cachat, Duparc, Thomas 2002)
- Pushdown games with unboundedness and regular conditions (Bouquet, Serre, Walukiewicz 2003)
- Games with Winning Conditions of High Borel Complexity (Serre 2004)

Further Results

- Pushdown games with winning condition “there is a configuration that is visited infinitely often” (Cachat, Duparc, Thomas 2002)
- Pushdown games with unboundedness and regular conditions (Bouquet, Serre, Walukiewicz 2003)
- Games with Winning Conditions of High Borel Complexity (Serre 2004)
- Parity games on higher order pushdown graphs (Cachat 2003)
- Parity games on higher order pushdown graphs with collapse (Hague, Murawski, Ong, Serre 2008)

Outline

1. Büchi-Landweber Theorem

2. Extensions
   - Pushdown Games
   - Delay Games

3. Implementations

Delaying the Output

I

J

Delaying the Output

I a_0

J

Delaying the Output

$I \qquad a_0$

$J \qquad b_0$

Delaying the Output

I a_0 a_1

J b_0

Delaying the Output

I a_0 a_1

J b_0 skip

Delaying the Output

I $a_0\ a_1\ a_2$

J b_0

Delaying the Output

I $a_0\ a_1\ a_2$

J $b_0\ b_1$

Delaying the Output

I $a_0\ a_1\ a_2\ a_3$

J $b_0\ b_1$

Delaying the Output

I $a_0\ a_1\ a_2\ a_3$

J $b_0\ b_1$ skip

Delaying the Output

I $a_0\ a_1\ a_2\ a_3\ a_4$

J $b_0\ b_1$

Delaying the Output

$I \quad a_0\ a_1\ a_2\ a_3\ a_4$

$J \quad b_0\ b_1\ b_2$

Delaying the Output

$$I \quad a_0\ a_1\ a_2\ a_3\ a_4 \cdots \models \varphi$$

$$J \quad b_0\ b_1\ b_2 \cdots$$

Delaying the Output

$$\begin{array}{ll} I & a_0\ a_1\ a_2\ a_3\ a_4 \cdots \\ J & b_0\ b_1\ b_2 \cdots \end{array} \models \varphi$$

Question:

Given a specification, does there exist a strategy (program) $P : I^* \to (J \cup \{\mathsf{skip}\})$ that

- produces an infinite output sequence for each input and
- realizes the specification?

Delaying the Output

$$\begin{array}{ll} I & a_0\ a_1\ a_2\ a_3\ a_4 \cdots \\ J & b_0\ b_1\ b_2 \cdots \end{array} \models \varphi$$

Question:

Given a specification, does there exist a strategy (program) $P : I^* \longrightarrow (J \cup \{\mathsf{skip}\})$ that

- produces an infinite output sequence for each input and
- realizes the specification?

The delay function for a strategy assigns to each number i the number of steps that the input is ahead, when symbol number i is output.

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$
  Output has to skip once at the beginning and then plays b iff the input is a.

  I

  J

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$
  Output has to skip once at the beginning and then plays b iff the input is a.

  I $\quad$ a'

  J

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$
  Output has to skip once at the beginning and then plays b iff the input is a.

I	a'
J	skip

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$
  Output has to skip once at the beginning and then plays b iff the input is a.

  $I \quad a' \; a'$

  J

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$
  Output has to skip once at the beginning and then plays b iff the input is a.

  I $\quad$ a' a'

  J $\quad$ b'

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$
  Output has to skip once at the beginning and then plays b iff the input is a.

  $I \quad a' \; a' \; a \cdots$

  $J \quad b'$

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$
  Output has to skip once at the beginning and then plays b iff the input is a.

  $I \quad a' \; a' \; a \cdots$

  $J \quad b' \; b \cdots$

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$
  Output has to skip once at the beginning and then plays b iff the input is a.

  $I \quad a' \; a' \; a \cdots$

  $J \quad b' \; b \cdots$

- $\exists x(a(x) \rightarrow b(0)) \wedge \forall x(a'(x) \rightarrow b'(0))$

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$
  Output has to skip once at the beginning and then plays b iff the input is a.

  $I \quad a' \; a' \; a \cdots$

  $J \quad b' \; b \cdots$

- $\exists x(a(x) \rightarrow b(0)) \wedge \forall x(a'(x) \rightarrow b'(0))$
  There is no strategy with delay for this specification.

  I

  J

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$
  Output has to skip once at the beginning and then plays b iff the input is a.

  $I \quad a' \; a' \; a \cdots$

  $J \quad b' \; b \cdots$

- $\exists x(a(x) \rightarrow b(0)) \wedge \forall x(a'(x) \rightarrow b'(0))$
  There is no strategy with delay for this specification.

  $I \quad a'$

  J

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$
  Output has to skip once at the beginning and then plays b iff the input is a.

  I $\quad a'\ a'\ a \cdots$

  J $\quad b'\ b \cdots$

- $\exists x(a(x) \rightarrow b(0)) \wedge \forall x(a'(x) \rightarrow b'(0))$
  There is no strategy with delay for this specification.

  I $\quad a'$

  J $\quad$ skip

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$
  Output has to skip once at the beginning and then plays b iff the input is a.

  $I \quad a' \; a' \; a \cdots$

  $J \quad b' \; b \cdots$

- $\exists x(a(x) \rightarrow b(0)) \wedge \forall x(a'(x) \rightarrow b'(0))$
  There is no strategy with delay for this specification.

  $I \quad a' \; a'$

  J

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$
  Output has to skip once at the beginning and then plays b iff the input is a.

  I $\quad a'\ a'\ a \cdots$

  J $\quad b'\ b \cdots$

- $\exists x(a(x) \rightarrow b(0)) \wedge \forall x(a'(x) \rightarrow b'(0))$
  There is no strategy with delay for this specification.

  I $\quad a'\ a'$

  J $\quad$ skip

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$
  Output has to skip once at the beginning and then plays b iff the input is a.

  $I \quad a' \; a' \; a \cdots$

  $J \quad b' \; b \cdots$

- $\exists x(a(x) \rightarrow b(0)) \wedge \forall x(a'(x) \rightarrow b'(0))$
  There is no strategy with delay for this specification.

  $I \quad a' \; a' \; a' \cdots$

  J

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$
  Output has to skip once at the beginning and then plays b iff the input is a.

  I $\quad a'\ a'\ a \cdots$

  J $\quad b'\ b \cdots$

- $\exists x(a(x) \rightarrow b(0)) \wedge \forall x(a'(x) \rightarrow b'(0))$
  There is no strategy with delay for this specification.

  I $\quad a'\ a'\ a'$

  J $\quad$ skip

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$
  Output has to skip once at the beginning and then plays b iff the input is a.

  $I \quad a' \; a' \; a \cdots$

  $J \quad b' \; b \cdots$

- $\exists x(a(x) \rightarrow b(0)) \wedge \forall x(a'(x) \rightarrow b'(0))$
  There is no strategy with delay for this specification.

  $I \quad a' \; a' \; a' \; a'$

  J

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$
  Output has to skip once at the beginning and then plays b iff the input is a.

  $I \quad a' \; a' \; a \cdots$

  $J \quad b' \; b \cdots$

- $\exists x(a(x) \rightarrow b(0)) \wedge \forall x(a'(x) \rightarrow b'(0))$
  There is no strategy with delay for this specification.

  $I \quad a' \; a' \; a' \; a'$

  $J \quad b'$

Examples

$I = \{a, a'\}$ and $J = \{b, b'\}$

- $\forall x(b(x) \leftrightarrow a(x+1))$
  Output has to skip once at the beginning and then plays b iff the input is a.

  $I \quad a' \; a' \; a \cdots$

  $J \quad b' \; b \cdots$

- $\exists x(a(x) \rightarrow b(0)) \wedge \forall x(a'(x) \rightarrow b'(0))$
  There is no strategy with delay for this specification.

  $I \quad a' \; a' \; a' \; a' \; a \cdots$

  $J \quad b'$

Bounded Delay in Regular Games

Theorem (Holtmann,Kaiser,Thomas 2010). For MSO specifications it is decidable if there is a strategy with delay realizing the specification. Furthermore, strategies with bounded delay are sufficient.

Delay for Pushdown Specifications

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{a, b\}$):

$$\binom{a}{a}^{\omega} \text{ or}$$

Delay for Pushdown Specifications

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{a, b\}$):

$$\binom{a}{a}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n}\binom{b}{J}\binom{I}{J}^{\omega} \text{ or}$$

Delay for Pushdown Specifications

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{a, b\}$):

$$\binom{a}{a}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n}\binom{b}{J}\binom{I}{J}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n+1}\binom{b}{J}\binom{I}{J}^{\omega}$$

Delay for Pushdown Specifications

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{a, b\}$):

$$\binom{a}{a}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n}\binom{b}{J}\binom{I}{J}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n+1}\binom{b}{J}\binom{I}{J}^{\omega}$$

There is a strategy with linear delay:

I

J

Delay for Pushdown Specifications

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{a, b\}$):

$$\binom{a}{a}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n}\binom{b}{J}\binom{I}{J}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n+1}\binom{b}{J}\binom{I}{J}^{\omega}$$

There is a strategy with linear delay:

I a

J

Delay for Pushdown Specifications

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{a, b\}$):

$$\binom{a}{a}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n}\binom{b}{J}\binom{I}{J}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n+1}\binom{b}{J}\binom{I}{J}^{\omega}$$

There is a strategy with linear delay:

I	a
J	skip

Delay for Pushdown Specifications

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{a, b\}$):

$$\binom{a}{a}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n}\binom{b}{J}\binom{I}{J}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n+1}\binom{b}{J}\binom{I}{J}^{\omega}$$

There is a strategy with linear delay:

I a a

J

Delay for Pushdown Specifications

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{a, b\}$):

$$\binom{a}{a}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n}\binom{b}{J}\binom{I}{J}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n+1}\binom{b}{J}\binom{I}{J}^{\omega}$$

There is a strategy with linear delay:

I	a	a
J	a	

Delay for Pushdown Specifications

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{a, b\}$):

$$\binom{a}{a}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n}\binom{b}{J}\binom{I}{J}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n+1}\binom{b}{J}\binom{I}{J}^{\omega}$$

There is a strategy with linear delay:

I a a a

J a

Delay for Pushdown Specifications

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{a, b\}$):

$$\binom{a}{a}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n}\binom{b}{J}\binom{I}{J}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n+1}\binom{b}{J}\binom{I}{J}^{\omega}$$

There is a strategy with linear delay:

I	a	a	a
J	a	skip	

Delay for Pushdown Specifications

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{a, b\}$):

$$\binom{a}{a}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n}\binom{b}{J}\binom{I}{J}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n+1}\binom{b}{J}\binom{I}{J}^{\omega}$$

There is a strategy with linear delay:

I	a	a	a	a
J	a			

Delay for Pushdown Specifications

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{a, b\}$):

$$\binom{a}{a}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n}\binom{b}{J}\binom{I}{J}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n+1}\binom{b}{J}\binom{I}{J}^{\omega}$$

There is a strategy with linear delay:

I	a	a	a	a
J	a	a		

Delay for Pushdown Specifications

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{a, b\}$):

$$\binom{a}{a}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n}\binom{b}{J}\binom{I}{J}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n+1}\binom{b}{J}\binom{I}{J}^{\omega}$$

There is a strategy with linear delay:

I	a	a	a	a	a
J	a	a			

Delay for Pushdown Specifications

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{a, b\}$):

$$\binom{a}{a}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n}\binom{b}{J}\binom{I}{J}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n+1}\binom{b}{J}\binom{I}{J}^{\omega}$$

There is a strategy with linear delay:

I	a	a	a	a	a
J	a	a	skip		

Delay for Pushdown Specifications

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{a, b\}$):

$$\binom{a}{a}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n}\binom{b}{J}\binom{I}{J}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n+1}\binom{b}{J}\binom{I}{J}^{\omega}$$

There is a strategy with linear delay:

I		a	a	a	a	a	b
J		a	a				

Delay for Pushdown Specifications

Example: Specification allows the following pairs of input/output sequences (with $I = J = \{a, b\}$):

$$\binom{a}{a}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n}\binom{b}{J}\binom{I}{J}^{\omega} \text{ or } \binom{a}{a}^{n}\binom{a}{b}^{n+1}\binom{b}{J}\binom{I}{J}^{\omega}$$

There is a strategy with linear delay:

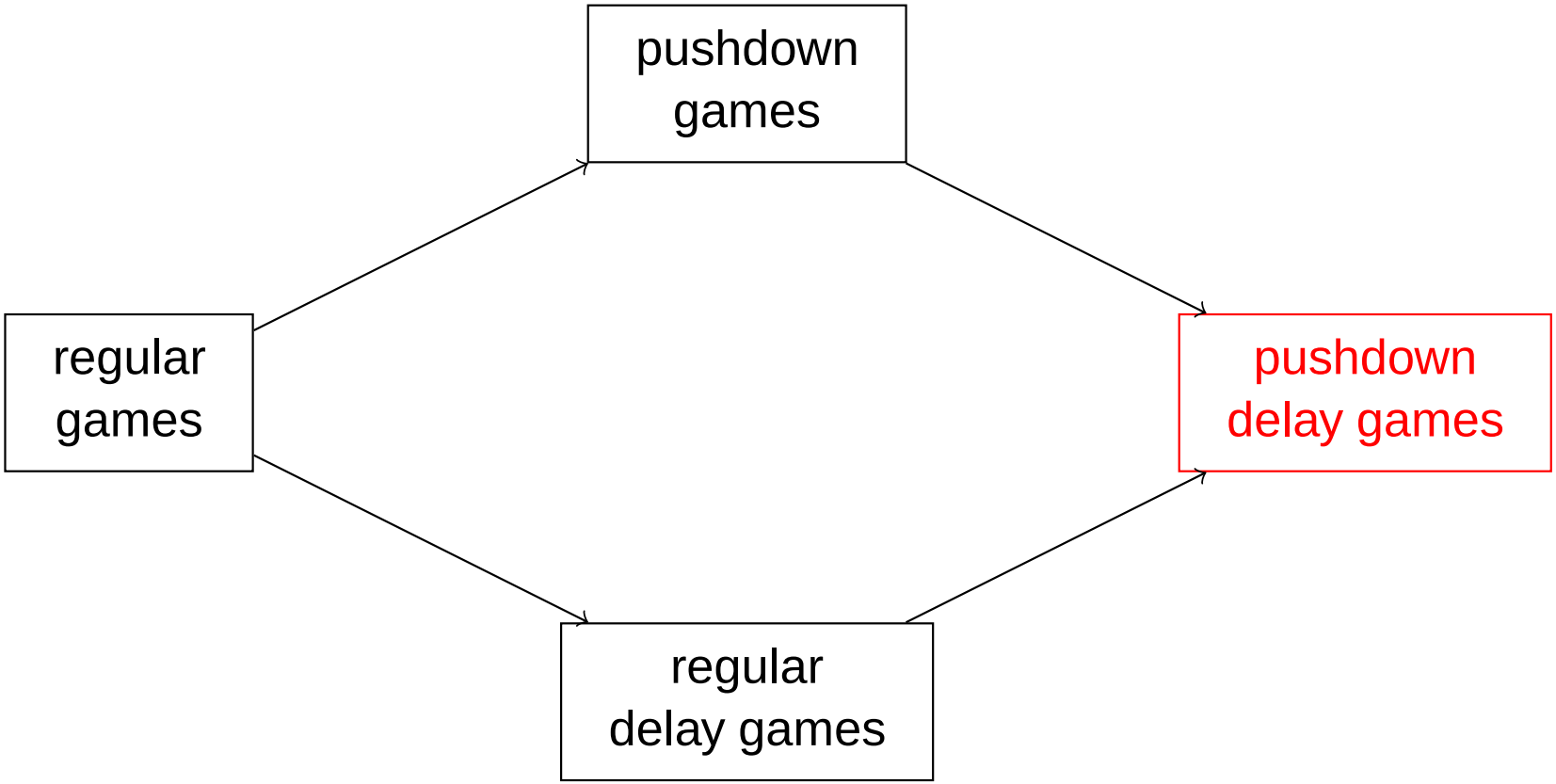
I		a	a	a	a	a	b
J		a	a	b			

Undecidability for Pushdown Delay Games

Theorem (Fridman,L.,Zimmermann 2011).

- There are deterministic pushdown specifications for which there is a delay strategy but each such strategy needs non-elementary delay.
- For deterministic pushdown specifications it is undecidable if there is a strategy with delay realizing the specification.

Summary


pushdown
games
regular
games
pushdown
delay games
regular
delay games

Outline

1. Büchi-Landweber Theorem

2. Extensions
   - Pushdown Games
   - Delay Games

3. Implementations

Complexity

Using Determinancy of Games to Eliminate Quantifiers.

J. Richard Büchi, Purdue University

⋮

Some have dreamed about implementing such decision methods on the beautiful modern machines. Some feel that the complexity boys have spoiled these dreams. But then, if one was to heed present complexity theory, how would he dare implement propositional calculus, and how else was he going to use the machine, if he was to believe it can't handle truth tables.

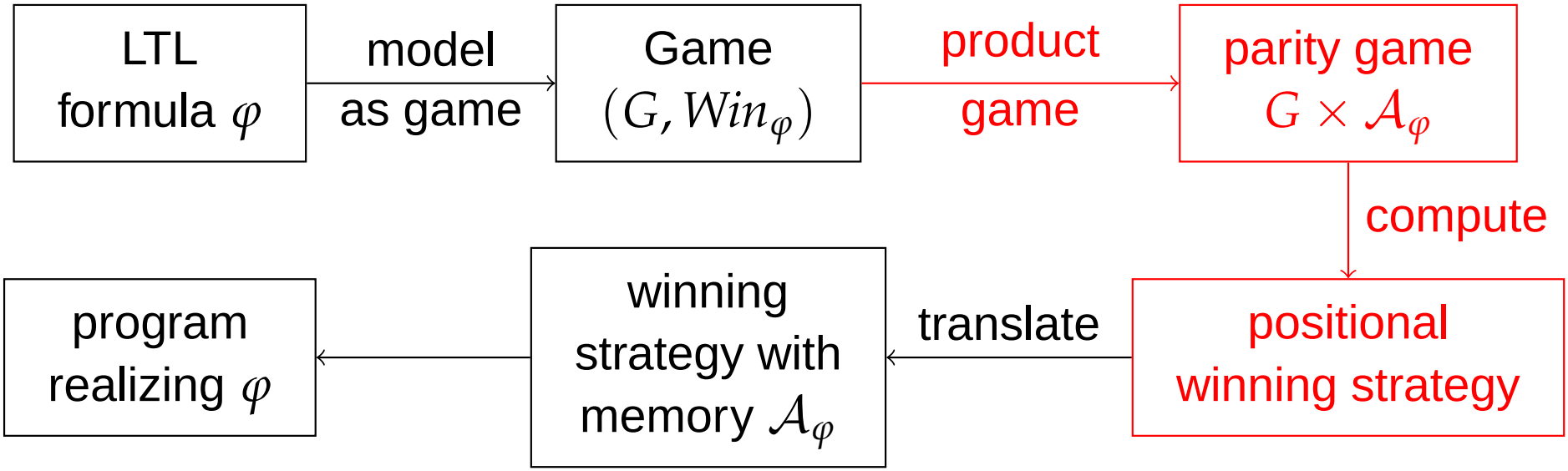
From Theory to Practice

Use the results to synthesize reactive programs from specifications.

Specification language:

- The translation from MSO into ω-automata yields automata of size non-elementary in the length of the formula (negations in the formula require complementation).
- Instead of using MSO, implementations use temporal logics like LTL as specification language.
- LTL allows a rather simple translation to nondeterministic Büchi automata (singly exponential) that is also used in verification.

Complexity Issues



Problems:

- From nondet. Büchi with n states to det. parity with $2n^n n!$ states and $2n$ priorities.
- Determinization of ω-automata is complex. No good optimizations, difficult to implement symbolically.
- Solving parity games is exponential in the number of priorities.

Implementations

- Lily - a LInear Logic sYnthesizer
  [B. Jobstmann, R. Bloem 2006]
  `http://www.iaik.tugraz.at/content/research/design_verification/lily/`
- Anzu
  [R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, M. Weiglhofer 2007]
  `http://www.ist.tugraz.at/staff/jobstmann/anzu/`
- Acacia: LTL Realizability Check and Winning Strategy Synthesis using Antichains
  [E. Filiot, N. Jin, J.-F. Raskin 2009]
  `http://www.antichains.be/acacia/`
- Unbeast – Symbolic Bounded Synthesis
  [R. Ehlers 2010] building on work of B. Finkbeiner and S. Schewe
  `http://react.cs.uni-sb.de/tools/unbeast/`

Summary and Outlook

The work of Büchi and Landweber has laid the foundations for many interesting developments in the area of synthesis and infinite game theory.

Some current topics

- Pushing further the decidability results for infinite graphs
- Extended models (probabilities, time)
- Distributed synthesis (partial information)
- Improve algorithms to obtain good implementations