Infinite games and automata theory

Christof Löding
RWTH Aachen, Germany
`http://automata.rwth-aachen.de/~loeding`

Lecture notes for the tutorial presented at the GAMES spring school, June 2009, Bertinoro, Italy

This is a preliminary version (May 26, 2009). It needs further revision, is not yet complete, and references are missing.

Abstract

The tutorial is an introduction to the connection between automata theory and the theory of two player games of infinite duration. We illustrate how the theory of automata on infinite words can be used to solve games with complex winning conditions, for example specified by logical formulas. Vice versa, infinite games are a useful tool to solve problems for automata on infinite trees such as complementation and the emptiness test.

Contents

1 Basic notations and definitions

For a set X we denote by X^* the set of finite sequences over X, and by X^ω the set of infinite sequences. For $\alpha \in X^\omega$ let

$$\mathrm{Inf}(\alpha) = \{x \in X \mid x \text{ occurs infinitely often in } \alpha\}.$$

We can view an infinite sequence $\alpha \in X^\omega$ as a mapping $\alpha : \mathbb{N} \to X$. Consequently, we write $\alpha(i)$ to denote the element at position i in α, i.e.,

$$\alpha = \alpha(0)\alpha(1)\alpha(2)\cdots$$

A *game graph* (also called *arena*) is a tuple $G = (V_\mathsf{E}, V_\mathsf{A}, E, c)$ with

- V_E: vertices of Eva (player 1, circle),
- V_A: vertices of Adam (player 2, box),
- $E \subseteq V \times V$: edges with $V = V_\mathsf{E} \cup V_\mathsf{A}$,
- $c : V \to C$ with a finite set of colors C.

A *play* in G is an infinite sequence $\alpha = v_0v_1v_2\cdots$ of vertices such that $(v_i, v_{i+1}) \in E$ for all $i \geq 0$. By $c(\alpha)$ we denote the corresponding sequence of colors $c(v_0)c(v_1)c(v_2)\cdots$

A *game* is a pair of a game graph and a *winning condition* $\mathcal{G} = (G, \mathit{Win})$ with $\mathit{Win} \subseteq C^\omega$. Eva wins a play α if $c(\alpha) \in \mathit{Win}$. Otherwise Adam wins.

A *strategy for Eva* is a function

$$\sigma : V^*V_\mathsf{E} \to V$$

with $\sigma(xv) = v'$ implies $(v, v') \in E$. A play $v_0v_1v_2\cdots$ is *played according to* σ if

$$\forall i : v_i \in V_\mathsf{E} \to \sigma(v_0 \cdots v_i) = v_{i+1}$$

Strategies for Adam are defined similarly with V_A instead of V_E.

Given an initial vertex v_0 and a strategy σ (for Eva or Adam), the set $\mathit{Out}(\sigma, v_0)$ contains all plays starting in v_0 that are played according to σ (the possible outcomes of σ).

A strategy σ for Eva in a game $\mathcal{G} = (G, \mathit{Win})$ is a *winning strategy from vertex* v_0 if $\mathit{Out}(\sigma, v_0) \subseteq \mathit{Win}$, i.e., if Eva wins all plays that start in v_0 and that are played according to σ.

Similarly, for Adam a strategy σ is winning from v_0 if $\mathit{Out}(f, v_0) \cap \mathit{Win} = \emptyset$.

The *winning area for Eva* is the set W_E of all vertices from which Eva has a winning strategy:

$$W_\mathsf{E} = \{v \in V \mid \text{Eva has a winning strategy from } v\}.$$

The *winning area for Adam* is denoted W_A and is defined in the same way.

A game $\mathcal{G} = (G, \mathit{Win})$ is *determined* if from each node either Eva or Adam has a winning strategy, i.e., if $W_\mathsf{E} \cup W_\mathsf{A} = V$.

The notion of strategy is very general because it allows the players to base their decision of the next move on the whole history of the play. This makes strategies infinite objects in general. Very often simpler types of strategies are sufficient. The simplest such strategies are *positional strategies*, which only depend on the current vertex. For Eva a positional strategy corresponds to a function $\sigma : V_{\mathsf{E}} \to V$ with $(v, \sigma(v)) \in E$ for each $v \in V_{\mathsf{E}}$ (and similarly for Adam).

A generalization of this concept are *finite memory strategies*. These are given by a deterministic finite automaton $(Q, C, q_{in}, \delta, \sigma)$ with input alphabet C, state set Q, initial state q_{in}, and transition function δ, and (instead of final states) the strategy function $\sigma : Q \times V_{\mathsf{E}} \to V$ with $(v, \sigma(q, v)) \in E$ for each $v \in V_{\mathsf{E}}$ and $q \in Q$. The idea is that the finite automaton reads the (color sequence of) the history of the play, and the strategy function chooses the next move depending on the current vertex and the state of the automaton. In this way Eva can use a bounded amount of information on the history of the play.

In the above definition the winning condition is given by some set $Win \subseteq C^{\omega}$. Usually, this set win is defined in terms of the colors that appear infinitely often during a play. Some of the most common winning conditions are listed below:

- Büchi conditions, which are given by a set $F \subseteq C$ of colors. The set Win contains all plays α such that $\mathrm{Inf}(\alpha) \cap F \neq \emptyset$, i.e., Eva wins if the at least one color from F occurs infinitely often in α.

- Muller conditions, which are given by a family $\mathcal{F} \subseteq 2^C$ of sets of colors. The set Win contains all plays α such that $\mathrm{Inf}(\alpha) \in \mathcal{F}$, i.e., Eva wins if the set of colors that occur infinitely often in α is a set that is listed in $\mathcal{F}$.

- Parity conditions, for which the set of colors is a finite set of natural numbers, $C \subseteq \mathbb{N}$. The set Win contains all plays α for which the maximal number that occurs infinitely often in α is even.

Clearly, Büchi conditions and parity conditions are special cases of Muller conditions. For a Büchi condition defined by F one obtains an equivalent Muller condition using $\mathcal{F} = \{D \subseteq C \mid D \cap F \neq \emptyset\}$. For a parity condition defined over $C \subseteq \mathbb{N}$ one obtains an equivalent Muller condition by $\mathcal{F} = \{D \subseteq C \mid \max(D) \text{ is even}\}$.

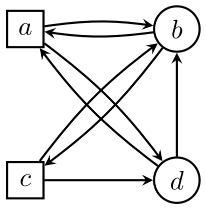
These types of games have been considered in the tutorial "Algorithms for solving infinite games on graphs" by Marcin Jurdziński. The central role of parity conditions in the theory of infinite games of graphs is due to the following result.

Theorem 1 (Emerson/Jutla'88,Mostowski'91) *Parity games are determined with positional strategies.*

2 Transformation of winning conditions

The goal of this section is to show how to use automata theory to transform complex winning conditions into simpler ones such that solving the simpler game also allows to solve the original game.

We start with an illustrating example. Consider a game graph with colors $C = \{a, b, c, d\}$. The winning condition is specified by a regular expression r (over the alphabet C): Eva wins if the play never matches the regular expression r. An example for such a game is shown in Figure 1. The regular expression in this example defines all words such that there exist two occurrences of c



$$C^*c(a+b)^*cC^* + C^*d(a+b)^*dC^*$$

Figure 1: Eva wins a play if it never matches the regular expression

without any occurrence of c or d inbetween, or two occurrences of d without an occurrence of c or d inbetween. Eva wins if no prefix of the play satisfies this property. In the depicted game graph she can achieve this goal by the following strategy σ:

- From the d vertex she always moves to b, and
- from the b vertex she moves
  - to c if b was reached from d, and
  - to a if b was reached from a or c.

We now illustrate a general method to compute such strategies. Instead of developing a direct algorithm, we use the fact that regular expressions can be translated into equivalent deterministic finite automata (DFA). Given such an automaton $\mathcal{A}$ we can build the product of the game graph G and $\mathcal{A}$ in such a way that $\mathcal{A}$ reads the play in G. This is illustrated in Figure 2, where on the left-hand side the game graph and the DFA $\mathcal{A}$ are shown, and on the right-hand side the product game graph. The meaning of the gray vertices and the dashed edges is explained below. For the moment they can be considered as normal edges and vertices.

Now the goal of Eva is to avoid to reach the final state of $\mathcal{A}$: In the game graph this event is represented by the bottom vertex named 3 (in the full product there would be vertices $(a,3),\dots,(d,3)$ but since Adam wins as soon as one of these vertices is reached they are collapsed to a single vertex).

So the goal of Eva in the new game is to avoid the vertex 3. Such a game is called a safety game because Eva has to remain in the safe area of the game (corresponding to the states $0,1,2$ of $\mathcal{A}$). To solve such a game we can simply compute the vertices from which Adam can force to reach the bad vertex. The set of these vertices is called $Attr_{\mathsf{A}}(3)$, the attractor for Adam of the vertex 3. In general, for a set R of vertices, the attractor $Attr_{\mathsf{A}}(R)$ is computed in iterations as follows:

$$\begin{array}{lcl} Attr^0_{\mathsf{A}}(R) & = & R \\ Attr^{i+1}_{\mathsf{A}}(R) & = & Attr^i_{\mathsf{A}}(R) \cup \\ & & \{v \in V_{\mathsf{A}} \mid \exists u \in Attr^i_{\mathsf{A}}(R) \,:\, (v,u) \in E\} \\ & & \{v \in V_{\mathsf{E}} \mid \forall u \,:\, (v,u) \in E \rightarrow u \in Attr^i_{\mathsf{A}}(R)\}. \end{array}$$

The set $Attr^i_{\mathsf{A}}(R)$ contains those vertices from which Adam can ensure to reach a vertex in R after at most i moves. For a finite game graph there exists an

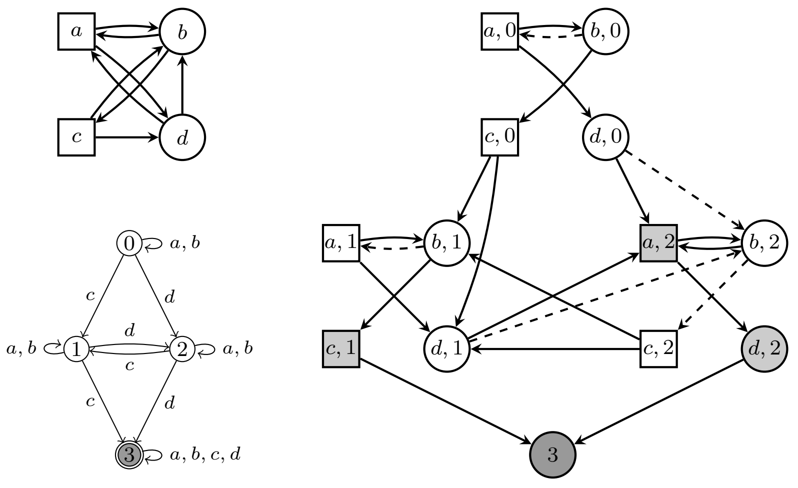


Figure 2: Product of the game graph with a DFA yields an equivalent safety game

index i such that $Attr^i_{\mathsf{A}}(R) = Attr^{i+1}_{\mathsf{A}}(R)$ and we set $Attr_{\mathsf{A}}(R) = Attr^i_{\mathsf{A}}(R)$ for this index i.

From the vertices that are not inside $Attr_{\mathsf{A}}(R)$ Eva has a simple winning strategy to avoid a visit of R: She always moves to a vertex outside of $Attr_{\mathsf{A}}(R)$. The definition of the attractor ensures that such a move is always possible. Furthermore, from outside the attractor Adam does not have the possibility to move inside (again by definition of the attractor).

In the product game graph from the example in Figure 2 the attractor for Adam of the vertex 3 consists of the gray vertices $(c, 1), (d, 2)$ (in the first iteration), and $(a, 2)$ (in the second iteration). A strategy for Eva to avoid the attractor is given by the dashed arrows.

In the original game, Eva can realize the strategy by running the DFA $\mathcal{A}$ on the play and making her decision based on the current vertex, the current state of $\mathcal{A}$, and the strategy from the product game. In the example this results in the same σ strategy explained above at the beginning of this section.

Exercise 1 The method illustrated above uses a translation of regular expressions into deterministic finite automata. Analyze the method when using nondeterministic finite automata instead and show that it does not work in general.

For the above example we used a translation from regular expressions to DFAs. For infinitary conditions (something happens infinitely/finitely often) standard DFAs are not enough. To treat such conditions we can use ω-automata.

2.1 ω-Automata

Automata on infinite words are defined in similar way as automata on finite words. The main difference is that a run of such an automaton does not have a last state because the input is infinite. For automata on finite words, acceptance

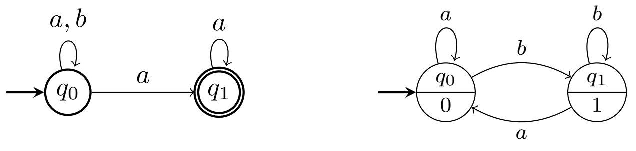


Figure 3: A nondeterministic Büchi automaton and a deterministic parity automaton accepting the words containing finitely many b

of a word by a run is defined in terms of the last state of the run: it has to be a final state. The definition is replaced by other mechanisms for acceptance, similar to the winning conditions in infinite games.

An *ω-automaton* is of the form $\mathcal{A} = (Q, \Sigma, q_0, \Delta, Acc)$, where Q, Σ, q_0, Δ are as for standard finite automata, i.e., Q is a finite set of states, Σ is the input alphabet, q_0 is the initial state, and $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation, and Acc defines the acceptance condition and is explained below.

For an infinite word $\alpha \in \Sigma^\omega$, a run of $\mathcal{A}$ on α is an infinite sequence of states $\rho \in Q^\omega$ that starts in the initial state, $\rho(0) = q_0$, and respects the transition relation, $(\rho(i), \alpha(i), \rho(i+1)) \in \Delta$ for all $i \geq 0$.

It remains to define when such a run is accepting. We are mainly interested in two types of acceptance conditions:

- In a *Büchi automaton Acc* is given as set $F \subseteq Q$ of states. A run is accepting if it contains infinitely often a state from F.
- In a *parity automaton Acc* is given as a priority mapping $pri : Q \to \mathbb{N}$. A run is accepting if the maximal priority appearing infinitely often is even.

Deterministic automata are defined as usual: there is at most one transition per state and letter.

Figure 3 shows a nondeterministic Büchi-automaton (on the left-hand side) accepting the language of infinite words over $\Sigma = \{a, b\}$ that contain finitely many b. A simple argument shows that there is no deterministic Büchi automaton for this language. But it is very easy to construct a deterministic parity automaton for the same language using the priorities 0 and 1. Such an automaton is shown on the right-hand side of Figure 3.

One can show that the two models of nondeterministic Büchi automata and deterministic parity automata are in fact equivalent in expressive power. The difficult direction is the construction of a deterministic parity automaton from a nondeterministic Büchi automaton.

THEOREM 2 (MCNAUGHTON'66,SAFRA'88) *For each nondeterministic Büchi automaton with n states there is an equivalent deterministic parity automaton with $2^{\mathcal{O}(n \log n)}$ states.*

EXERCISE 2 Show that each deterministic parity automaton can be transformed into an equivalent nondeterministic Büchi automaton. Hint: The Büchi automaton guesses an even priority at some point and verifies that it occurs infinitely often and that it is the maximal priority from this point onwards.

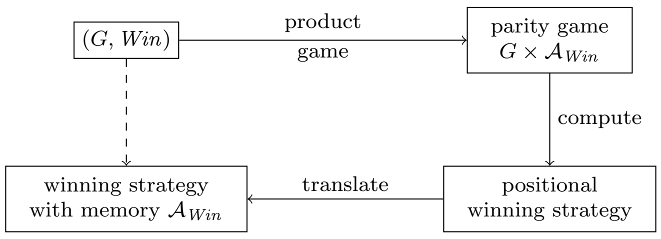

(G, Win)
product game
parity game $G \times \mathcal{A}_{Win}$
compute
winning strategy with memory $\mathcal{A}_{Win}$
translate
positional winning strategy

Figure 4: Schema for game reductions

We call languages that can be accepted by nondeterministic Büchi automata (or equivalently by deterministic parity automata) *ω-regular*.

2.2 Game reductions

The general idea for game reductions is to transform games with complicated winning condition into games with a simpler winning condition (over a bigger game graph) such that a winning strategy for the simpler but bigger game can be transformed into a winning strategy for the original game (using additional memory).

At the beginning of this section we have already seen how to translate a winning condition that specifies forbidden prefixes by a regular expression into a safety game. The general translation scheme that we apply is as follows (illustrated in Figure 4):

- Start from a game $\mathcal{G} = (G, Win)$ with some ω-regular winning condition $Win \subseteq C^\omega$.
- Construct a deterministic ω-automaton $\mathcal{A}_{Win}$ for Win.
- Take the product of G and $\mathcal{A}_{Win}$ (the automaton reads the labels of the vertices in G).
- The new winning condition is the acceptance condition of $\mathcal{A}_{Win}$ (e.g., a parity condition).
- Winning strategies on the product game are winning strategies on the original game with (additional) memory $\mathcal{A}_{Win}$.

We show how to use this technique to reduce Muller games to parity games. For this purpose we construct a deterministic parity automaton that reads sequences α from C^ω and accepts if α satisfies the given Muller condition.

The construction is based on the data structure of "latest appearance record" (LAR). The underlying idea is to recall the order in which the colors of the play appeared (starting with an arbitrary order). An LAR is thus a permutation of the colors over which the Muller condition is defined. When reading the next color of the sequence, it is moved to the first position in the ordering. To define the parity condition of the automaton we additionally mark the old position of the color that has been moved. This idea is illustrated in Figure 5 for the color set $C = \{a, b, c, d\}$. The LARs are written vertically, and when reading the next

	d	b	b	d	c	b	a	b	a	c	b	a	b	a	c
a	d	b	$\underline{b}$	d	c	b	a	b	a	c	b	a	b	a	c
b	a	d	d	$\underline{b}$	d	c	b	$\underline{a}$	$\underline{b}$	a	c	b	$\underline{a}$	$\underline{b}$	a
c	b	$\underline{a}$	a	a	b	$\underline{d}$	c	c	c	$\underline{b}$	$\underline{a}$	$\underline{c}$	c	c	$\underline{b}$
$\underline{d}$	$\underline{c}$	c	c	c	$\underline{a}$	a	$\underline{d}$	d	d	d	d	d	d	d	d
7	7	5	1	4	7	5	7	3	3	6	6	6	3	3	6

Figure 5: Latest appearance records for a given sequence of colors from $\{a, b, c, d\}$ for the Muller condition $\mathcal{F} = \{\{b, d\}, \{a, b, c\}\}$

color it is moved to the top. The numbers below the LARs are the assigned priorities and are explained later.

The marked positions are underlined. We point out that it is not the underlined color that is marked, but the position. For example, in the fifth LAR in the picture, it is not b that is marked but the second position because this LAR was obtained by moving d from the second position to the front.

Note that in this way the colors that appear only finitely often gather at the end (bottom) of the LAR and that the marker eventually stays in the part of the LAR that keeps changing.

Formally, a *latest appearance record (LAR)* over C is an ordering $d_1 \cdots d_n$ of the elements of C with one marked position h:

$$LAR(C) = \{[d_1 \cdots d_n, h] \mid d_i \in C, d_i \neq d_j \text{ for all } i \neq j, \text{ and } 1 \leq h \leq n\}$$

The set of LARs serves as set of states for the parity automaton. The transition function (update of the LARs) is defined as explained above:

$$\delta_{LAR}([d_1 \cdots d_n, h], d) = [dd_1 \cdots d_{i-1}d_{i+1} \cdots d_n, i]$$

for the unique i with $d = d_i$.

It remains to assign the priorities to the states, i.e., to the LARs. This is done depending on the size of the part of the LAR that has changed in the last transition. As explained above the biggest part of the LAR that changes infinitely often represents the set of colors that appear infinitely often and hence the parity automaton should accept if this set belongs to the Muller condition.

If C contains n colors, then we assign the priorities as follows:

$$c_{LAR}([d_1 \cdots d_n, h]) = \begin{cases} 2h-1 & \{d_1, \ldots, d_h\} \notin \mathcal{F}, \\ 2h & \{d_1, \ldots, d_h\} \in \mathcal{F}. \end{cases}$$

In the example from Figure 5 this is shown for the Muller condition $\mathcal{F} = \{\{b, d\}, \{a, b, c\}\}$.

Combining all this we obtain the LAR automaton

$$\mathcal{A}_{LAR} = (LAR(C), C, q_0, \delta_{LAR}, c_{LAR})$$

and the following theorem:

THEOREM 3 *For a Muller condition $\mathcal{F}$ over C the corresponding deterministic parity automaton $\mathcal{A}_{LAR}$ accepts precisely those $\alpha \in C^\omega$ that satisfy the Muller condition $\mathcal{F}$.*

Now, given a Muller game, we can take the product with the LAR automaton. This results in a parity game for which we can compute the winner and a positional winning strategy. This winning strategy can be implemented over the Muller game using the LAR automaton as memory.

THEOREM 4 *Muller games are determined with the LAR automaton as memory.*

2.3 Logical winning conditions

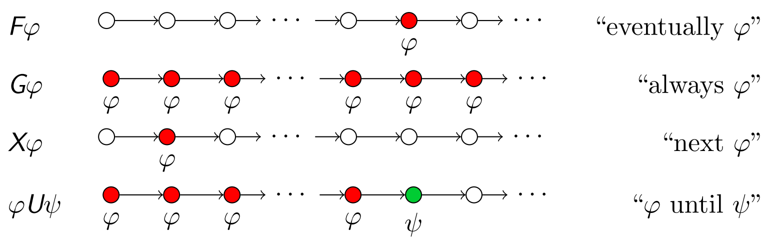
In this section we show examples for game reductions where the winning conditions of the games are given by logical formulas. We consider two logics, names the linear time temporal logic LTL, and monadic second order logic over infinite words.

LTL

The formulas of LTL are defined over a set $P = \{p_1, \ldots, p_n\}$ of atomic propositions, and are evaluated over infinite sequences of vectors of size n. The ith entry of such a vector codes the truth value of p_i (1 = true, 0 = false).

LTL formulas are build up from

- atomic formulas of the form p_i,
- Boolean combinations, and
- temporal operators:



We illustrated the semantics by giving some examples. An atomic formula p_i is true if p_i is true at the beginning of the word, i.e., if the ith entry of the first vector is 1. A formula $\mathsf{X}\varphi$ is true if φ is true in the model starting from the second position. For example, $p_1 \wedge X\neg p_2$ is true in

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

as witnessed by the underlined entries.

A formula $\mathsf{F}\varphi$ is true if there exists some position $i \geq 0$ in the word such that φ is true in this position (i.e. in the suffix of the word starting at this position i). A formula $\mathsf{G}\varphi$ is true if φ is true at each position of the word. For example, $\mathsf{G}p_2 \wedge \mathsf{F}p_1$ is true in

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

again witnessed by the underlined entries.

$\alpha =$	$\binom{1}{0}$	$\binom{0}{1}$	$\binom{1}{1}$	$\binom{0}{0}$	$\binom{1}{0}$	$\binom{0}{1}$	$\cdots$
$\neg p_1$	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	$\cdot$	$\cdot$	$\cdots$

Figure 6: A Büchi automaton guesses valuations for subformulas

A formula $\varphi \mathsf{U} \psi$ is true if there is some position where ψ is true, and φ is true in all previous positions. Consider the formula $\mathsf{F}(p_3 \wedge \mathsf{X}(\neg p_2 \mathsf{U} p_1))$. It states that there is some position where p_3 is true such that from the next position p_2 is true until a position is reached where p_1 is true. The words satisfying this formula are of the following shape, where again the underlined entries are those making the formula true:

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

We are interested in games with LTL winning conditions, i.e., games of the form (G, φ) for an LTL formula φ. Eva wins the game if the play satisfies φ. To be able to interpret LTL formulas in infinite plays we assume that the set C of colors of the game graph is $\{0,1\}^n$.

To apply the method of game reduction it suffices to construct an equivalent automaton for a given formula. We explain the underlying idea for transforming an LTL formula into an equivalent (nondeterministic) Büchi automaton: The automaton "guesses" for each subformula of the given formula φ its truth value at the current position and verifies its guesses. Thus, the states of the automaton are mappings that associate to each subformula of φ a truth value 0 or 1. Figure 6 illustrates how the Büchi automaton works. The dots in the table indicate that the truth value of the formulas at this position depends on the continuation of the input word α.

The verification of guesses works as follows:

- Atomic formulas and Boolean combinations can be verified directly because the truth values of the atomic formulas are coded in the input letter by definition.
- The operators X, G can be verified using the transitions: If the automaton guesses that a formula $\mathsf{X}\psi$ is true at the current position, then ψ has to be true at the next position. If a formula $\mathsf{G}\psi$ is guessed to be true, then $\mathsf{G}\psi$ and ψ have to be true in the next position.
- The operators F, U are verified using the acceptance condition. We explain the principle for the operator F. This can easily be generalized to U.

If $\mathsf{F}\psi$ is guessed to be true, then either $\mathsf{F}\psi$ has to be true again in the next position, or ψ itself has to be true in the current position. Using the acceptance condition, the automaton has to ensure that the second option is taken at some point, i.e., that ψ indeed becomes true eventually. For this purpose, we use a slight generalization of Büchi automata that have several sets $F_1, \dots, F_k$ of final states. A run is accepting if all sets are visited infinitely often. Now, for each formula $\mathsf{F}\psi$ we introduce one set of final states that contains all states in which ψ is true or $\mathsf{F}\psi$ is false. Like this, once $\mathsf{F}\psi$ is guessed to be true, ψ has to become true eventually, because otherwise the set of final states for $\mathsf{F}\psi$ will not be visited anymore.

The same principle applies to subformulas with the until operator: A formula $\psi_1 \mathsf{U} \psi_2$ is true if either ψ_1 is true now and $\psi_1 \mathsf{U} \psi_2$ is true again in the next position, or if ψ_2 is true now.

In the first position the automaton has to guess that the whole formula φ is true because it is supposed to accept exactly those words which satisfy the formula.

Exercise 3 In the same way as for formulas $\mathsf{F}\psi$ we introduce a set of final states for each subformula $\psi_1 \mathsf{U} \psi_2$. How should the definition of this set of states look like?

Exercise 4 The construction from LTL formulas to automata explained above yields a generalized Büchi automaton with several sets of states. Find a construction that transforms a generalized Büchi automaton into an equivalent Büchi automaton with only one set of final states. (Hint: Cycle through the different sets of final states by introducing copies of the Büchi automaton.)

Using the idea illustrated above we obtain the following theorem

Theorem 5 (Vardi/Wolper'86) *For each LTL formula φ one can construct an equivalent Büchi automaton $\mathcal{A}_\varphi$ of size exponential in φ.*

Using the determinization theorem for ω-automata and the method of game reduction, we obtain the following theorem.

Theorem 6 *Games (G, φ) with a winning condition given by an LTL formula can be solved in doubly exponential time.*

S1S

We now consider monadic second-order logic over the natural numbers with successor function, or the "second-order theory of one successor" (S1S). The underlying structure $(\mathbb{N}, +1)$ consists of the natural numbers as domain and the successor function.

Monadic second-order logic is the extension of first-order logic by the ability to quantify over sets of elements. We do not give the precise definition but only illustrate the syntax with an example. In the formulas we use small letters $x, y, \dots$ as first-order variables denoting elements, and capital letters $X, Y, \dots$ for set variables denoting sets of natural numbers.

Consider the formula

$$\varphi(X) = \exists Y\Big(\begin{array}{l} 0 \in Y \land \\ \forall x(x \in Y \leftrightarrow x+1 \notin Y) \land \\ \forall x(x \in X \rightarrow x \in Y)\Big)\end{array}$$

It has one free set variable X denoting a set of natural numbers. We can view a set of natural numbers as an ω-word over the alphabet $\{0,1\}$: The positions that are in X are labeled 1, and the other positions are labeled 0.

Using this interpretation, φ defines the set of all ω-words over $\{0,1\}$ such that 1 can only occur on even positions: The formula states that there is a set Y that contains position 0, and it contains exactly every second position (i.e., Y contains exactly the even positions), and it contains X. Thus, this formula is true for each interpretation of the free variable X by a set containing only even positions.

In general we consider formulas $\varphi(X_1,\dots,X_n)$ with n free set variables defining ω-languages over $\{0,1\}^n$. We have already seen that LTL formulas (which also define languages over the alphabet $\{0,1\}^n$) can be translated into automata. For S1S formulas we even have a stronger result that the two formalisms are equivalent.

Theorem 7 (Büchi'62) *A language $L \subseteq (\{0,1\}^n)^\omega$ is definable by an S1S formula iff it can be accepted by a nondeterministic Büchi automaton.*

Proof. A detailed version of this proof can be found in [2]. We only give a brief sketch of the ideas.

From formulas to automata one uses an inductive translation, based on the closure properties of automata. To make this approach work, one first introduces a variant of S1S that uses only set variables (and has a predicate saying that a set is a singleton and a predicate for set inclusion). Atomic formulas are easily translated into equivalent automata. For the Boolean combinations one uses the closure properties of automata, and for the existential quantification the projection.

From Büchi automata to formulas on writes a formula that describes the existence of an accepting run. For each state q one uses a set X_q that contains the positions of the run where the automaton is in state q. Then one can easily express that the run starts in the initial state, that infinitely many final states occur, and that the transitions are respected in each step. □

As for LTL winning conditions, we can now consider games with winning conditions specified by S1S formulas. Using the translation into nondeterministic Büchi automata and the determinization theorem we can solve such games.

Theorem 8 *For games (G,φ) with a winning condition given by an S1S formula φ one can decide the winner and can compute a corresponding winning strategy.*

The complexity of the inductive translation of formulas into automata is rather high because each negation in the formula requires a complementation

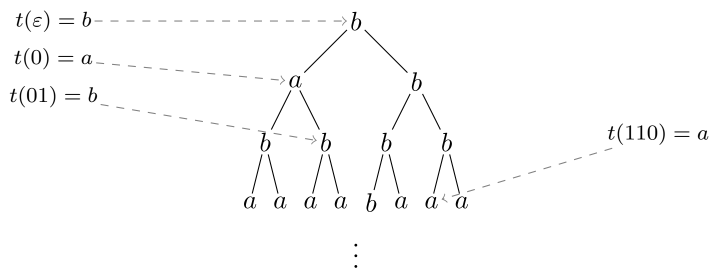


Figure 7: The initial part of an infinite tree over the alphabet $\{a, b\}$

of the automaton, which is exponential. Based on lower bound results for the translation of formulas into automata one can show that this also applies to the memory required for winning strategies in S1S games. The size of the memory required for a winning strategy in a game with an S1S winning condition cannot be bounded by a function of the form

$$\left.2^{2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}}\right\} k$$

for a fixed k.

3 Tree automata

Infinite trees are a useful tool to model the behavior of discrete systems (circuits, protocols etc.). These can be described by transitions graphs, and the possible behaviors or executions of such a system are captured by an infinite tree: the unraveling of the transition graph. Properties of the system behavior can thus be specified as properties of infinite trees.

For simplicity we restrict ourselves to complete binary trees. The nodes are labeled by symbols from a finite alphabet Σ. When modeling system executions by infinite trees this alphabet captures properties of the system we are interested in.

Formally, a tree is a mapping $t : \{0, 1\}^* \to \Sigma$. The root of the tree corresponds to the empty string ε. For some node $u \in \{0, 1\}^*$ the left successor is $u0$, and the right successor is $u1$. This is illustrated in Figure 7 showing the initial part of a tree over the alphabet $\{a, b\}$. The name of a node corresponds to the sequence of left (0) and right(1) moves that lead to this node from the root.

We now introduce an automaton model that defines sets (languages) of such infinite trees. This model can be seen as an extension of ω-automata to trees, and it is also an extension of the model of automata on finite trees. As for ω-automata, one can study different models that are distinguished by the form of their acceptance condition. We focus here on parity tree automata, which are defined as follows.

A parity tree automaton (PTA) is of the form $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, pri)$ with a finite set Q of states, a label alphabet Σ, an initial state q_{in}, a transition relation $\Delta \subseteq Q \times \Sigma \times Q \times Q$, and a priority function $pri : Q \to \mathbb{N}$.

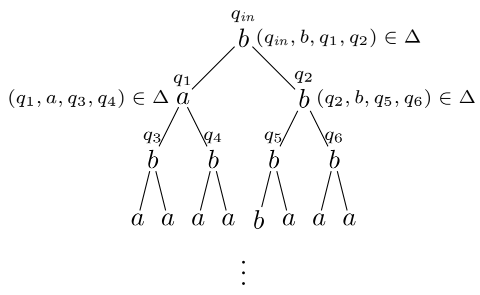

q_{in}
b $(q_{in}, b, q_1, q_2) \in \Delta$
q_1
$(q_1, a, q_3, q_4) \in \Delta$ a
q_2
b $(q_2, b, q_5, q_6) \in \Delta$
q_3 q_4 q_5 q_6
b b b b
a a a a b a a a

Figure 8: A run of a PTA has to respect the transition relation

A run ρ of a PTA on a tree t is a mapping $\rho : \{0,1\}^* \to Q$ (a Q-labeled tree) that starts in the initial state, $\rho(\varepsilon) = q_{in}$, and that respects the transitions, $(\rho(u), t(u), \rho(u0), \rho(u1)) \in \Delta$ for all $u \in \{0,1\}^*$. This is illustrated in Figure 8.

The acceptance condition of a PTA is defined via the priority mapping. From games and ω-automata we already know that a sequence of states is accepting if the maximal priority that appears infinitely often is even. We extend this to trees by defining a run to be accepting if the state sequence on each infinite path through the run is accepting, i.e., if on each infinite path the maximal priority that occurs infinitely often is even.

As usual, a tree is accepted if there is an accepting run on this tree. The language of trees accepted by $\mathcal{A}$ is denoted by $T(\mathcal{A})$. We call a language of infinite trees regular if it can be accepted by a parity tree automaton.

Before we come to the properties of PTAs, we make some remarks on other acceptance conditions:

- As for games or ω-automata we can define, e.g., automata with Muller or Büchi acceptance conditions.
- One can show that Büchi tree automata are weaker than parity tree automata. The language of all trees over $\{a, b\}$ such that on each path there are finitely many b cannot be accepted by a Büchi tree automaton but by a PTA. The PTA can easily be obtained by running the parity word automaton from Figure 3 on every branch of the tree using the transitions (q_0, a, q_0, q_0), (q_0, b, q_1, q_1), (q_1, a, q_0, q_0), and (q_1, b, q_1, q_1). A proof that a Büchi tree automaton cannot accept this language can be found in [2] and in Chapter 8 of [1].
- The expressive power of Muller tree automata and parity tree automata is the same. To transform Muller automata into parity tree automata one can use the LAR construction presented in the previous section.

We now analyze algorithmic and closure properties of PTAs. Some closure properties are rather easy to obtain.

PROPOSITION 9 *The class of regular languages of infinite trees is closed under union, intersection, and projection (relabeling).*

Proof. For the closure under union and intersection one can use a classical product construction where the two given automata are executed in parallel.

The acceptance condition becomes a Muller condition that expresses that both automata accept for the intersection, or that at least one of the automata accepts for the union. As mentioned above, one can use the LAR construction to turn a Muller automaton into an equivalent parity automaton.

For the projection let $h : \Sigma \to \Gamma$ be a relabeling. It is applied to trees by applying it to each label of the tree. Given a PTA $\mathcal{A}$ for a tree language T we want to construct a PTA for the tree language $h(T) = \{h(t) \mid t \in T\}$. For this purpose we simply replace every transition (q, a, q_0, q_1) in $\mathcal{A}$ by the transition $(q, h(a), q_0, q_1)$. □

The connection to games is used for the closure under complementation and the emptiness test, as explained in the following.

3.1 Complementation

Let us first analyze why the complementation problem for PTAs is a difficult problem. By definition, a tree is accepted if

$$\exists \text{run} \forall \text{path}.(\text{path satisfies acceptance condition})$$

By negation of this statement we obtain that a tree is not accepted if

$$\forall \text{run} \exists \text{path}.(\pi \text{path does not satisfy acceptance condition})$$

This exchange of quantifiers makes the problem difficult. But there are two observations to make that lead towards the solution. First of all, statements of the form $\exists\forall\cdots$ are very close to the nature of games and strategies: there exists a move of one player that ensures the win against all moves of the other player. In this case the game has just two rounds: The first player picks a run on the input tree, and the second player picks a path through the run. The first player wins if the path satisfies the acceptance condition. We will modify this game such that the players do not choose these objects in one shot but incrementally build them. This allows to express the acceptance of a tree in the form

$$\exists \text{strategy for Eva } \forall \text{strategies for Adam } (...)$$

Then we use determinacy of these games allowing us to express non-acceptance of a tree as

$$\exists \text{strategy for Adam } \forall \text{strategies for Eva } (...)$$

This statement is still in the form of $\exists\forall\cdots$ and we show how to construct an automaton that checks this property.

We start by defining the game that characterizes the membership of a tree t in the language of a given PTA $\mathcal{A} = (Q, \Sigma, q_{in}, \Delta, pri)$. Since one player tries to construct an accepting run and the other player tries to show that the run is not accepting by selecting a path through the run, we call the players Constructor and Spoiler.

The rules of the game are as follows:

- The game starts a the root of the tree in the initial state of $\mathcal{A}$, i.e., in the position (q_{in}, ε).

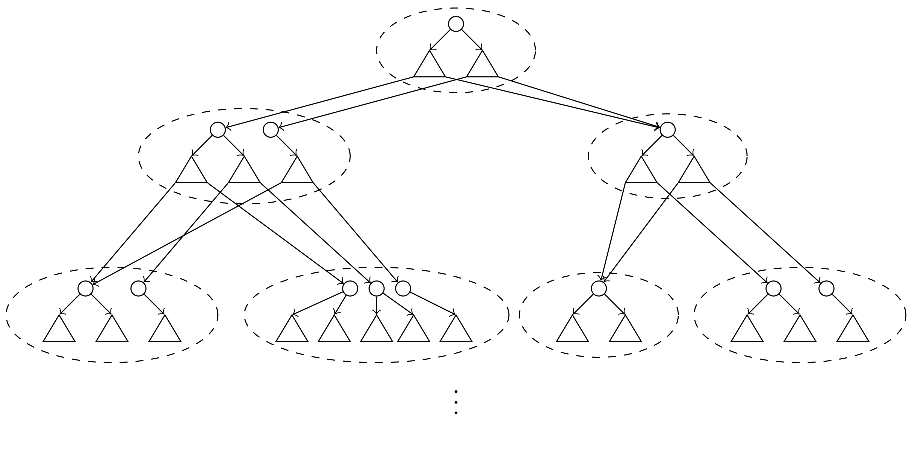


Figure 9: Shape of the membership game

- The moves of the game from a position (u, q) where $u \in \{0,1\}^*$ is a node of t, and q is a state of $\mathcal{A}$ are:
  1. Constructor picks a transition (q, a, q_0, q_1) that matches q and the label a of t at u, i.e., $a = t(u)$.
  2. Spoiler chooses a direction and the game moves on to position $(u0, q_0)$ or $(u1, q_1)$.
- A play of this game is an infinite sequence of states and transitions together with the nodes in the tree. For the winning condition only the sequence of states is interesting: Constructor wins this state sequence satisfies the acceptance condition of $\mathcal{A}$.

The shape of this game is illustrated in Figure 9. The dashed lines represent the tree nodes. For each tree node the circles represent states, i.e., the positions of Constructor, and the triangles transitions. The arrows from the circles to the triangles indicate that Constructor chooses for a given state a possible transition, and the arrows exiting from the triangles correspond to the choices of Spoiler who moves either to the left or to the right into the corresponding state of the transition.

With this picture in mind, it is rather easy to see that there is a correspondence between the runs of $\mathcal{A}$ on t and the strategies for Constructor:

- Fixing the transition to be used at the root is the same as defining the first move of Constructor.
- Then Spoiler can only move to the left or to the right. The states at these successors are already fixed by the transition. Defining the strategy for Constructor at these successors is again the same as fixing the transitions for a run, and so on.

By the definition of the winning condition and positional determinacy of parity games we obtain the following lemma.

Lemma 10 *A tree t is in $T(\mathcal{A})$ iff there is a positional winning strategy for* Constructor *in the membership game.*

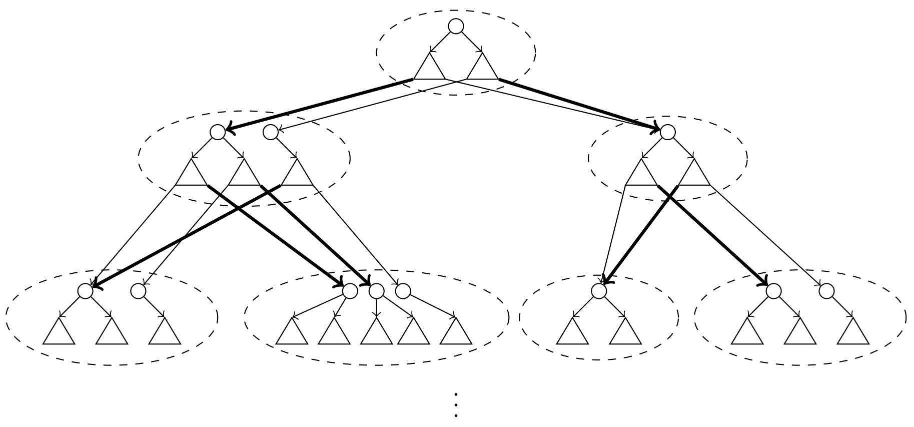


Figure 10: Shape of positional strategies for SPOILER in the membership game

Applying the positional determinacy for the negation of the statement, we obtain:

LEMMA 11 *A tree t is not in $T(\mathcal{A})$ iff there is a positional winning strategy for* SPOILER *in the membership game.*

The next goal is to construct an automaton that checks for a tree t if there is a positional winning strategy for SPOILER in the membership game. By the previous lemma such an automaton suffices to recognize the complement language of $\mathcal{A}$. We start by analyzing positional strategies for SPOILER.

A move for SPOILER consists in choosing a direction for a given pair of tree node and transition, as illustrated by the thick edges in Figure 10.

Such a positional strategy for SPOILER can be written as a mapping

$$\sigma : \{0,1\}^* \to (\Delta \to \{0,1\})$$

that defines for each tree node how the choices of SPOILER for the different transitions are. Let us denote the finite set $\Delta \to \{0,1\}$ of mappings from transitions to directions $0,1$ by Γ. Then a positional strategy for SPOILER is simply a tree over the alphabet Γ, and hence can be processed by a tree automaton.

Now the next steps in the construction of the complement automaton $\mathcal{C}$ are the following:

- Construct an automaton $\mathcal{A}_{\text{strat}}$ that reads trees of the form $t \times \sigma$, i.e., annotated with a positional strategy for SPOILER such that $\mathcal{A}_{\text{strat}}$ accepts $t \times \sigma$ if σ is winning for SPOILER in the membership game for $\mathcal{A}$ and t.
- Obtain $\mathcal{C}$ from $\mathcal{A}_{\text{strat}}$ by omitting the strategy annotation in the labels. This operation corresponds to a simple projection that removes the Γ component of the labels. This can be seen as $\mathcal{C}$ nondeterministically guessing the strategy for SPOILER.

Looking at Figure 10, $\mathcal{A}_{\text{strat}}$ has to check that the paths obtained by following the thick edges (the strategy of SPOILER) do not satisfy the acceptance condition

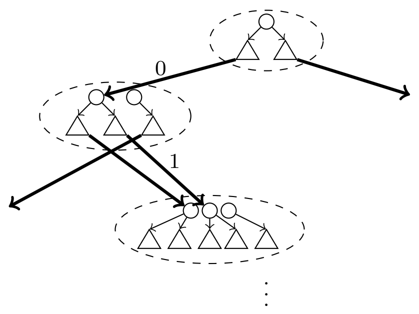


Figure 11: A single branch through a coding of a positional strategy of Spoiler

(the strategy is winning for Spoiler). For this task we first focus on single branches of the tree.

We code these branches by adding to the labels on the branch the next direction 0 or 1 taken by the branch. In this way we obtain infinite words over $\Sigma \times \Gamma \times \{0, 1\}$. This is indicated in Figure 11, where a single branch from the tree shown in Figure 10 is selected.

Now we construct an ω-automaton that checks whether on such branches the strategy is winning for Spoiler. This ω-automaton has to check whether all paths that can be obtained by following the thick arrows (the strategy of Spoiler) *along the given branch* do not satisfy the acceptance condition of $\mathcal{A}$. We obtain this ω-automaton as follows:

- Construct a nondeterministic Büchi automaton that guesses a path along the strategy edges of Spoiler and accepts if it does satisfy the acceptance condition of $\mathcal{A}$.
- Determinize and complement this Büchi automaton, and obtain a deterministic parity word automaton $\mathcal{A}^{\text{path}}_{\text{strat}}$ that accepts those branches of a tree on which Spoiler's strategy is winning.

Now we can run this deterministic parity word automaton $\mathcal{A}^{\text{path}}_{\text{strat}}$ in parallel along all branches of the given tree by merging the transitions of $\mathcal{A}^{\text{path}}_{\text{strat}}$ for the two directions 0 and 1 into a single transition of a tree automaton as follows:

$$\begin{array}{ll} \mathcal{A}^{\text{path}}_{\text{strat}}: & \delta(q, (a, \gamma, 0)) = q' \qquad \delta(q, (a, \gamma, 1)) = q'' \\[1em] \mathcal{A}_{\text{strat}}: & \qquad\qquad (q, (a, \gamma), q', q'') \end{array}$$

for all $q \in Q$, $a \in \Sigma$, and $\gamma \in \Gamma$.

Then obtain the automaton $\mathcal{C}$ by omitting the strategy encoding:

$$(q, (a, \gamma), q', q'') \text{ becomes } (q, a, q', q'').$$

From the explanations above it follows that $\mathcal{C}$ indeed accepts the complement language of $\mathcal{A}$, resulting in the following theorem.

Theorem 12 (Rabin'69) *For a given tree automaton one can construct a tree automaton for the complement language.*

The main steps of the construction described in this section are summarized as follows:

- Characterize acceptance in terms of winning strategies in the membership game.
- Positional determinacy for parity games yields: t is not accepted iff SPOILER has positional winning strategy in the membership game.
- Construct an automaton that checks if a given strategy of SPOILER is winning. This construction is based on the determinization of ω-automata.
- Obtain the desired automaton by projection (removing the strategy annotations).

The size of the complement automaton is determined by the complexity of the determinization construction for word automata. If $\mathcal{A}$ is a PTA with n states and k priorities, then the Büchi word automaton that has to be determinized is of size $\mathcal{O}(nk)$, and thus the resulting PTA $\mathcal{C}$ is of size $2^{\mathcal{O}(nk\log(nk))}$.

The positional determinacy of parity games is only used for the correctness of the construction.

As an application one can now use the closure properties of PTAs to show their equivalence to the logical formalism S2S which is defined in the same way as S1S but now over the structure $(\{0,1\}^*, S_0, S_1)$ consisting of the domain $\{0,1\}^*$ and the two successor function S_0 and S_1 for moving left or right in the tree.

In analogy to the results for ω-automata and S1S presented in Section 2.3 we obtain the equivalence of S2S and tree automata.

THEOREM 13 (BÜCHI'62) *A tree language L over the alphabet $\{0,1\}^n$ is definable by an S2S formula iff it can be accepted by a parity tree automaton.*

If we want to check the satisfiability of S2S formulas, we can translate them into tree automata and check these for emptiness. This latter problem is the subject of the next section.

3.2 Emptiness

We now want to develop a method that checks for a given PTA $\mathcal{A}$ whether $T(\mathcal{A})$ is empty or not. For the complementation construction we have used the membership game to characterize the acceptance of a tree by the automaton. The idea is that CONSTRUCTOR builds a run and SPOILER chooses a path through this run. The resulting game arena is infinite because the moves available at a certain node depend on the label of the tree at this node. For the emptiness problem we want to know whether there is a tree on which there is a run such that the acceptance condition is satisfied on each path:

$$\exists \text{tree} \exists \text{run} \forall \text{path}. (\text{path satisfies acceptance condition})$$

Accordingly, we modify the membership game such that CONSTRUCTOR now builds a tree t and a run ρ on t at the same time. This allows to remove the tree nodes from $\{0,1\}^*$ from the game positions because the moves do not depend on these nodes anymore.

The emptiness game for a PTA $\mathcal{A}$ has the following rules:

- The game positions are $Q \cup \Delta$ (states and transitions of $\mathcal{A}$).
- The initial position is q_{in}.
- The moves of the game from a position q (a state of $\mathcal{A}$):
  1. Constructor picks a transition (q, a, q_0, q_1),
  2. Spoiler chooses a direction and the game moves on to position q_0 or q_1.
- A play of this game is an infinite sequence of states and transitions. For the winning condition only the sequence of states is relevant: The winning condition for Constructor is the acceptance condition of $\mathcal{A}$.

This is a parity game on a finite game graph. In the same way as for the membership game we get the following lemma.

Lemma 14 *The language of the PTA $\mathcal{A}$ is not empty iff* Constructor *has a winning strategy in the emptiness game for $\mathcal{A}$.*

Consider the following example PTA $\mathcal{A}$ over the alphabet $\{a, b\}$ with state set $\{q_1, q_1', q_2, q_+\}$, initial state q_1, and the following transitions and priorities

$$\begin{array}{llll}
(q_1, a, q_1, q_+) & (q_1', a, q_+, q_1') & (q_+, a, q_+, q_+) & c(q_1) = c(q_1') = 1 \\
(q_1, a, q_2, q_+) & (q_1', b, q_+, q_+) & (q_+, b, q_+, q_+) & c(q_2) = c(q_+) = 2 \\
(q_2, a, q_1, q_1') & & &
\end{array}$$

This automaton accepts the trees t satisfying the following properties:

1. The left branch does not contain b, i.e., $t(0^i) = a$ for all i.
2. From the left branch infinitely often a b is reachable by only moving to the right, i.e., there are infinitely many nodes 0^i such that $t(0^i1^j) = b$ for some j.

The shape of an accepting run of $\mathcal{A}$ is depicted in Figure 12. The missing parts of the run are labeled with the state q_+. The emptiness game together with a winning strategy for Constructor is shown in Figure 13.

From a winning strategy one can deduce a finitely generated tree as shown in Figure 14.

Theorem 15 (Rabin'69) *The emptiness problem for parity tree automata is decidable. If the language is not empty, then one can construct a finite representation of a tree in the language.*

Corollary 16 (Rabin'69) *S2S is decidable.*

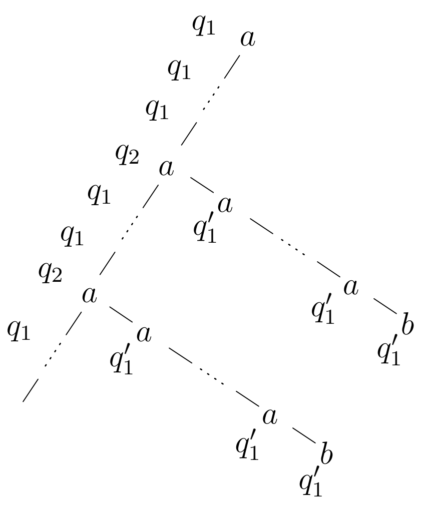


Figure 12: The shape of an accepting run of the example PTA

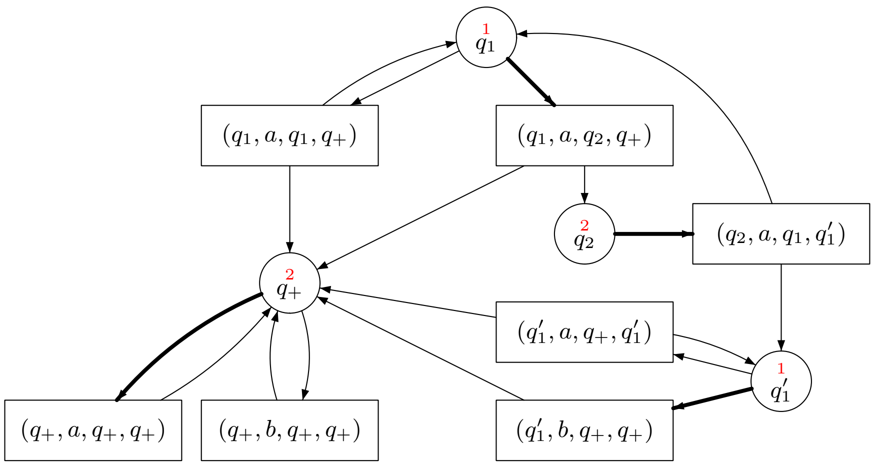


Figure 13: The emptiness game for the example PTA. The thick arrows define a winning strategy for Constructor

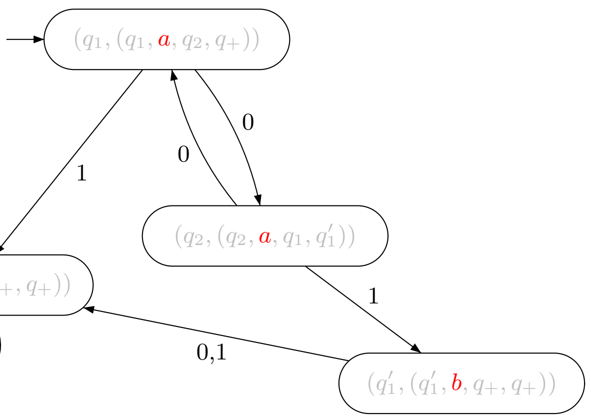


Figure 14: A finitely generated (regular) tree corresponding to the winning strategy depicted in Figure 13

References

[1] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[2] Wolfgang Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Language Theory*, volume III, pages 389–455. Springer, 1997.