Automata and Logic

Igor Walukiewicz
Warsaw University[1]

Contents

[1]Institute of Informatics, Warsaw University, Banacha 2, 02-097 Warszawa, Poland. e-mail:igw@mimuw.edu.pl; www: http://zls.mimuw.edu.pl/∼igw

1 Introduction

The connections between automata theory and logic are long and fruitful. Good examples of this are Büchi's proof of decidability of monadic second-order (MSO) theory of infinite words and Rabin's proof of decidability of MSO theory of infinite binary trees. This last theory is one of the strongest known decidable logical theories with many other decidability results being an easy consequence. Recently, automata play a prominent role in understanding of many logical formalisms used in Computer Aided Verification. Of particular interest to us here will be various versions of the μ-calculus.

An equivalence between MSOL, automata and the μ-calculus is an important and useful property. MSOL gives the guarantee of expressive power as the MSOL properties are by definition closed under Boolean operations and quantification. Automata are the main technical tool in analysing the logic and in particular for inexpressibility results. They are also crucial in the algorithmic problems. The μ-calculus offers a logical formalism which is usually of much smaller complexity than MSOL. Still, it is a usable formalism in which many interesting properties can be formulated succinctly. Sometimes, as in the case of graphs, the μ-calculus gives a recursive syntax for an interesting, but not recursive, subset of MSOL.

In these notes we will show many situations in which the three formalisms are equivalent. We start with classical equivalences between MSOL, automata and the μ-calculus over words and trees. The tools we develop allow

us then to study and compare hierarchies in all of the formalisms. Finally, we describe extensions of the classical results in three directions. We consider a more general relational setting, obtaining so called guarded logics. We also discuss the extensions to trace models and to real-time models.

These notes are not intended to be a survey of the results in the area. Some important aspects, as for example the relations with first-order logic, are omitted here. We refer the reader to excellent surveys and books [51, 52, 49, 7].

2 Finite words

Let Σ be a *finite alphabet*. A *finite word* over Σ is a sequence $w = a_0 \dots a_n$ of letters from Σ, or equivalently a function from $\{0, \dots, n\}$ to Σ. We use $|w|$ for the length of w, i.e., $n+1$ in this case. We use $\mathrm{dom}(w)$ for the domain of w, i.e., $\{0, \dots, n\}$. The empty word is denoted by ε. We write Σ^* for the set of all finite words over Σ.

The word w as above can be represented as a relational structure:

$$\mathcal{M}_w = \langle \mathrm{dom}(w), \leq, (P_a^w)_{a \in \Sigma} \rangle$$

where $\leq$ is the standard linear order on $\mathrm{dom}(w)$ and P_a^w are unary predicates with the interpretation: $P_a^w = \{i \in \mathrm{dom}(w) : w(i) = a\}$.

2.1 First-order and monadic second-order logics

The structures as above can be described in first-order or second-order logics which we are going to define now. Let $Var_1 = \{x, y, \dots\}$ be the set of *first order variables*. The set of first-order formulas is build from *atomic formulas* of the form:

$$x \leq y, \quad P_a(x) \qquad \text{for every } a \in \Sigma$$

using the connectives $\vee$, $\neg$ and the quantifier $\exists$. A *sentence* is a formula without free variables.

The meaning of a formula in a model of a form $\mathcal{M}_w$ is defined in a standard way. In particular the variables range over positions in w. If $p_1, \dots, p_n \in \mathrm{dom}(w)$ are positions in w and $\varphi(x_1, \dots, x_n)$ is a formula then $\mathcal{M}_w, p_1, \dots, p_n \models \varphi(x_1, \dots, x_n)$ means that φ holds in $\mathcal{M}_w$ when each x_i is interpreted as the position p_i. Other connectives as $\wedge$, $\Rightarrow$ and universal quantifier $\forall$ are definable in the usual way. We admit the empty model, $\mathcal{M}_\varepsilon$, as interpretation. In this model all existential sentences $\exists x.\varphi$ are false.

The *language defined by a sentence* φ is the set of words:

$$L(\varphi) = \{w \in \Sigma^* : \mathcal{M}_w \vDash \varphi\}$$

For example the sentence $\forall x.P_a(x) \Rightarrow (\exists y.x \leq y \wedge P_b(y))$ defines the language of words where after each letter a there is a letter b; in case $\Sigma = \{a, b\}$ these are the words ending in b. A language $L \subseteq \Sigma^*$ is *FO-definable* if there is a first-order sentence φ such that $L = L(\varphi)$.

Monadic second-order logic is an extension of first-order logic with quantification over sets of elements. Let $Var_2 = \{X, Y, \ldots\}$ be the set of *second-order variables*. The syntax of monadic second order logic extends first-order logic with atomic formulas $X(y)$ and quantification $\exists Y$ over second order variables. To interpret such formulas we need now valuations

$$V : (Var_1 \to \mathrm{dom}(w)) \times (Var_2 \to \mathcal{P}(\mathrm{dom}(w)))$$

assigning to each first-order variable an element of the domain and to each second-order variable a set of elements in the domain. The term monadic refers to the fact that second order variables range over sets of elements and not over relations of higher arity.

As before we write $L(\varphi)$ for a set of words defined by a MSOL sentence φ. For example the sentence: $\exists X.\ \big(\forall y.\ X(y) \Leftrightarrow \neg X(y+1)\big) \wedge \forall z.\ P_b(z) \Rightarrow X(z)$ expresses the fact that b appears only on odd or only on even positions in the word. Here we use $X(y+1)$ as a shorthand for saying that the position $y+1$ belongs to X. Clearly $+1(y, z)$ relation is definable in first-order logic over words.

When proving something by induction on the structure of MSOL formulas it will be convenient to have a small set of connectives and atomic formulas. Consider the set of formulas given by the grammar

$$\varphi ::= X \subseteq Y \mid X \subseteq P_a \mid \mathrm{Succ}(X, Y) \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \exists X.\varphi \qquad (1)$$

In these formulas there are no first-order variables and there are two new binary predicates $\subseteq$ and Succ. As for the meaning of such formulas in a structure $\mathcal{M}_w$ and valuation $V : Var_2 \to \mathcal{P}(\mathrm{dom}(w))$ we have:

- $\mathcal{M}_w, V \vDash X \subseteq Y$ if $V(X) \subseteq V(Y)$,
- $\mathcal{M}_w, V \vDash X \subseteq P_a$ if $V(X) \subseteq P_a^w$,
- $\mathcal{M}_w, V \vDash \mathrm{Succ}(X, Y)$ if $V(X)$ and $V(Y)$ are singletons p_1 and p_2, respectively, and $p_1 + 1 = p_2$.

It should be clear that both $\subseteq$ and Succ are definable in MSOL. The converse is also not difficult:

Lemma 1 Every MSOL formula without free first order variables is equivalent to a formula generated by the grammar (1).

Proof
First, we define singleton sets. Then, we simulate first-order variables and first-order quantification using singletons. □

2.2 Automata

A finite automaton is a tuple:

$$\mathcal{A} = \langle Q, \Sigma, q^0 \in Q, \delta \subseteq Q \times \Sigma \times Q, F \subseteq Q \rangle$$

where: Q is a finite set of states, Σ is the input alphabet, q^0 is the initial state, δ is the transition relation, and F is the set of final states.

A *run* of the automaton on a word $w = a_0 \dots a_n$ is a sequence $q_0, \dots, q_{n+1}$ such that: $q_0 = q^0$ is the initial state, and $(q_i, a_i, q_{i+1}) \in \delta$ for all $i = 0, \dots, n$. A run is *successful* if $q_{n+1} \in F$. The *language recognized by* $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words accepted by $\mathcal{A}$.

Definition 2 A language $L \subseteq \Sigma^*$ is *regular* iff it is the language recognized by some automaton.

It is well known that the class of regular languages is closed under:

- *sum*: if L_1 and L_2 are regular then $L_1 \cup L_2$ is regular
- *intersection*: if L_1 and L_2 are regular then $L_1 \cap L_2$ is regular,
- *complement*: if L is a regular language over Σ then $\Sigma^* \setminus L$ is regular,
- *projection*: if L is a regular language over $\Sigma = \{0,1\} \times \Sigma'$ then $\pi_2(L) = \{\pi_2(w) \in \Sigma' : w \in L\}$ is regular; where we write $\pi_2(w)$ for a word $a_0 \dots a_n \in \Sigma'$ whenever $w = (b_0, a_0)(b_1, a_1) \dots (b_n, a_n) \in \Sigma^*$.

The following theorem gives the connection between the languages accepted by automata and those defined in MSOL.

Theorem 3 (Büchi, Elgot)
A language of finite words is definable by a MSOL sentence iff it is the language recognized by some finite automaton. The translations in both directions are effective.

Proof
Let $\mathcal{A} = \langle Q, \Sigma, q^0, \delta, F\rangle$ be a finite automaton recognizing L. We need to write a formula $\varphi_{\mathcal{A}}$ which holds in a model $\mathcal{M}_w$ iff $w \in L$.

The formula $\varphi_{\mathcal{A}}$ says that there exist sets $S_0, \ldots, S_{|Q|}$ such that:

1. the sets form a partition of the domain;
2. the first element of the domain belongs to S_0, and the last to some S_k such that $q_k \in F$;
3. for every element different from the last, if the element belongs to some S_i and its successor to some S_j then $(q_i, a, q_j) \in \delta$, where a is the label of the element.

It should be clear that all these requirements can be formulated in MSOL and that the resulting formula expresses the existence of an accepting run of $\mathcal{A}$. This shows the right to left implication of the theorem.

For the implication from left to right we are going to construct an automaton for every MSOL formula. By Lemma 1 we can do this by induction on the reduced syntax of MSOL given by (1).

A small complication here is that we are going to translate formulas with free second-order variables. Our inductive translation will be simpler if we fix a set of variables $\{X_1, \ldots, X_n\}$ and provide the translation for formulas using only these variables. This is not a loss of generality as the set is arbitrary.

A formula with free variables in $\{X_1, \ldots, X_n\}$ defines the set of pairs $(\mathcal{M}, V)$ consisting of a model and a valuation in which the formula is satisfied. Such a pair can be coded as a word over the alphabet $\Sigma \times \mathcal{P}(\{1, \ldots, n\})$ where a position m is labelled with (a, S) iff a is the label of m in $\mathcal{M}$ and $S = \{i \in \{1, \ldots, n\} : m \in V(X_i)\}$.

By induction on the syntax of the formula φ we construct an equivalent automaton $\mathcal{A}_\varphi$, i.e., such that $\mathcal{A}_\varphi$ accepts the word representations of exactly those pairs $(\mathcal{M}, V)$ for which $\mathcal{M}, V \vDash \varphi$ holds.

The automaton for an atomic formula of the form $X_i \subseteq X_j$ is very simple. It checks that for all the letters (a, S) appearing in the word we have that whenever $i \in S$ then $j \in S$. The constructions of automata for $X_i \subseteq P_a$ and $\mathrm{Succ}(X_i, X_j)$ are also straightforward.

Consider the induction step. If $\varphi = \varphi_1 \vee \varphi_2$ then $L(\mathcal{A}_\varphi) = L(\mathcal{A}_{\varphi_1}) \cup L(\mathcal{A}_{\varphi_2})$. Similarly negation of a formula corresponds to complementation of the language, and existential quantification corresponds to projection. Hence the inductive step follows from well known constructions on finite automata. □

Let us present a small application of this theorem. An easy pumping argument shows that the language $\{a^n b^n : n \in \mathbb{N}\}$ is not recognized by any finite automaton, hence not definable in MSOL. We will use this fact to show that existence of Hamiltonian cycle is not expressible in MSOL over graphs. This logic is exactly the same as MSOL we have described above but with $\leq$ relation replaced by E relation interpreted as the edge relation of the graph.

A balanced bipartite graph is a graph whose set of vertices can be divided into two sets of the same size such that there are no edges between vertices of the same set. First, we show that there is no MSOL formula defining balanced bipartite graphs. Suppose to the contrary that ψ defines such graphs. We show that then the language $\{a^n b^n : n \in \mathbb{N}\}$ would be definable in MSOL which is impossible.

A word $a^m b^n$ defines a bipartite graph $K_{m,n}$ with vertices $\{(a,i) : i = 1,\ldots,m\} \cup \{(b,j) : j = 1,\ldots,n\}$ and edges connecting each a vertex with each b vertex. By our assumption we have $K_{m,n} \vDash \psi$ iff $n = m$. Replace each occurrence of $E(x,y)$ in ψ by the formula $Q_a(x) \wedge Q_b(y)$. Call the resulting formula $\widehat{\psi}$. An easy induction argument shows that $K_{n,m} \vDash \psi$ iff $a^n b^m \vDash \widehat{\psi}$. Hence $\widehat{\psi} \wedge \forall x,y.\ P_a(x) \wedge P_b(y) \Rightarrow x \leq y$ defines $\{a^n b^n : n \in \mathbb{N}\}$ which is impossible.

So, there is no MSOL formula ψ over graphs such that $K_{m,n} \vDash \psi$ iff $m = n$. In $K_{m,n}$ there is a Hamiltonian cycle iff $m = n$. The graphs of the form $K_{m,n}$ are definable in MSOL, hence there cannot be a MSOL formula defining Hamiltonicity.

2.3 Complexity

Let us shortly summarize the complexity results for MSOL and automata on finite words.

The *emptiness problem* is to decide whether a given automaton accepts some word. It is easy to see that the problem is equivalent to the reachability problem in finite graphs, hence it is NLOGSPACE-complete. Indeed, given an automaton we can construct a graph with states of the automaton as nodes and an edge whenever there is a transition on some letter between the states. The automaton accepts some word iff there is a path in the graph from the initial state to a final state.

The *universality problem* is to decide whether $L(\mathcal{A}) = \Sigma^*$ for a given automaton $\mathcal{A}$. This problem is PSPACE-complete [46]. A PSPACE algorithm for the problem is to determinize the automaton and look for a word that is not accepted. The deterministic automaton may be of exponential size but we never keep the whole of it in memory; we just calculate its states on

demand. For PSPACE hardness one shows that the language of words that are not computations of a given $O(n)$ space bounded Turing machine can be recognized by a small nondeterministic automaton.

The *satisfiability problem* for MSOL over finite words is to decide whether for a given formula φ there is a word w such that φ holds in $\mathcal{M}_w$. By Theorem 3 the problem is decidable. Meyer [35] has shown that the problem is nonelementary even for first-order logic.

Maybe it is worth to clarify the use of the term nonelementary here. Elementary functions were introduced by Grzegorczyk [24]. These are functions obtained from some basic functions by operations of limited summation and limited multiplication. Consider the function $Tower(n, k)$ defined by:

$$Tower(n, 0) = n \qquad Tower(n, k+1) = 2^{Tower(n,k)}$$

Grzegorczyk has shown that every elementary function in one argument is bounded by $\lambda n.\, Tower(n, c)$ for some fixed c. Hence, the term nonelementary refers to a function that grows faster than any such function. In particular for the case of FOL over finite words it is known that the complexity of the satisfiability problem is of order $Tower(n, cn)$ for some constant c [15].

3 Infinite words

An *infinite word* (or *ω-word*) over an alphabet Σ is an infinite sequence $w = a_0 a_1 \ldots$, or equivalently a function from $\mathbb{N}$ to Σ. We use Σ^ω for the set of infinite words over Σ. An infinite word w can be presented as a relational structure:

$$\mathcal{M}_w = \langle \mathbb{N}, \leq, P_a^w \rangle$$

where $\leq$ is the standard order on $\mathbb{N}$ and $P_a^w(i)$ holds iff $w(i) = a$. A more precise term for an infinite word is ω-word. Of course one can consider words for bigger ordinals than ω, but we will not do it here. Hence, we will use the two terms interchangeably.

The notions of first-order and second-order definability extend smoothly from finite to infinite words. A sentence φ now defines a set $\{w \in \Sigma^\omega : \mathcal{M}_w \models \varphi\}$ of infinite words.

The extension of automata recognizability to infinite words requires more work. An ω-automaton has the form:

$$\mathcal{A} = \langle Q, \Sigma, q^0 \in Q, \delta \subseteq Q \times \Sigma \times Q, Acc \subseteq Q^\omega \rangle$$

where Q is the finite set of states, Σ is the alphabet, q^0 is the initial state, δ is the transition relation, and Acc defines which infinite sequences are

accepting. Of course, if we want our automata to be finite, we need some finitary ways to describe the set Acc. These are discussed below.

A run of $\mathcal{A}$ on a word w is a sequence $r : \mathbb{N} \to Q$ such that $r(0) = q^0$, and $(r(i), w(i), r(i+1)) \in \delta$ for all $i \in \mathbb{N}$. A word w is *accepted* by $\mathcal{A}$ iff there is a run r of $\mathcal{A}$ on w such that $r \in Acc$. The *language recognized by* $\mathcal{A}$ is the set of words accepted by $\mathcal{A}$.

We are now going to define several standard ways of describing the set Acc. Each of these ways leads to a different notion of automaton. All of these ways refer to the states appearing infinitely often in the run. Hence we introduce the notation:

$$\mathrm{In}(r) = \{q \in Q : r(i) = q \text{ for infinitely many } i\}$$

The most frequently used acceptance conditions are the following requirements on the set $\mathrm{In}(r)$:

Büchi condition is specified by a set $F \subseteq Q$. We have
$Acc = \{r : \mathrm{In}(r) \cap F \neq \emptyset\}$.

Muller condition is specified by a set $\mathcal{F} \subseteq \mathcal{P}(Q)$. We have
$Acc = \{r : \mathrm{In}(r) \in \mathcal{F}\}$.

Rabin condition is specified by a set $\{(R_1, G_1), \ldots, (R_k, G_k)\}$, where $R_i, G_i \subseteq Q$. We have
$Acc = \{r : \exists i.\ \mathrm{In}(r) \cap R_i = \emptyset \text{ and } \mathrm{In}(r) \cap G_i \neq \emptyset\}$

Streett condition is also specified by a set $\{(R_1, G_1), \ldots, (R_k, G_k)\}$, where $R_i, G_i \subseteq Q$. We have
$Acc = \{r : \forall i.\ \mathrm{In}(r) \cap R_i = \emptyset \text{ or } \mathrm{In}(r) \cap G_i \neq \emptyset\}$

Mostowski condition Is specified by a function $\Omega : Q \to \mathbb{N}$. We have
$Acc = \{r : \min(\Omega(\mathrm{In}(r))) \text{ is even}\}$

Rabin condition is sometimes called "pairs condition"; Streett condition is called "complement pairs condition"; Mostowski condition is called "parity condition".

Automata are named after acceptance conditions so we have Büchi automata, Muller automata, etc.

Example: Suppose that $Q = \{q_1, q_2, q_3\}$. We will show how to express with different types of conditions the fact that q_1 appears only finitely often and q_2 appears infinitely often. This property cannot be expressed with Büchi conditions. The property is expressed with the Muller condition

$\{\{q_2\}, \{q_2, q_3\}\}$. The Rabin condition for the property is $\{(\{q_1\}, \{q_2\})\}$. The equivalent Streett condition is $\{(\{q_1\}, \emptyset), (\emptyset, \{q_2\})\}$. Finally, the Mostowski condition for the property is given by the function $\Omega(q_i) = i$ for $i = 1, 2, 3$. □

Fact 4 For every Büchi condition there is an equivalent Mostowski condition. Every Mostowski condition has equivalent Rabin and Streett conditions. Every Rabin or Streett condition has an equivalent Muller condition.

Proof
A Büchi condition $F \subseteq Q$ is equivalent to a Mostowski condition $\Omega : Q \to \{0, 1\}$, where $\Omega(q) = 0$ iff $q \in F$. A Mostowski condition $\Omega : Q \to \{0, \ldots, k\}$ is equivalent to a Rabin condition $\{(R_i, G_i) : i = 0, \ldots, k/2\}$ where $R_i = \{q : \Omega(q) < 2i\}$ and $G_i = \{q : \Omega(q) = 2i\}$. The translation to Streett condition is similar. It is obvious that any condition can be translated to a Muller condition. □

Fact 5 Mostowski and Muller conditions are closed under negation. The negation of a Rabin condition is a Streett condition and vice versa.

Proof
The complement of a Muller condition $\mathcal{F} \subseteq \mathcal{P}(Q)$ is $\overline{\mathcal{F}} = \mathcal{P}(Q) \setminus \mathcal{F}$. The complement of a Mostowski condition $\Omega : Q \to \mathbb{N}$ is given by $\overline{\Omega}(q) = \Omega(q) + 1$. The complement of a Rabin condition $\{(R_1, G_1), \ldots, (R_n, G_n)\}$ is $\{(G_1, R_1), \ldots, (G_n, R_n)\}$ interpreted as a Streett condition. □

Fact 6 Nondeterministic Büchi-, Muller-, Rabin-, Streett-, and Mostowski-automata all recognize the same class of ω-languages.

Proof
Every acceptance condition is a special form of Muller acceptance condition. Hence it is enough to show how to translate Muller automata to Büchi automata. Roughly, a Büchi automaton nondeterministically picks a set S from the Muller condition and checks that S is precisely the set of states appearing infinitely often. □

This fact allows us to formulate the definition:

Definition 7 A language $L \in \Sigma^\omega$ is *regular* if it is the language recognized by some nondeterministic Büchi automaton.

Unlike the case of finite words it is not true that every automaton can be determinized. Büchi automata cannot be determinized as the following

example shows. Let $L_a \in \{a,b\}^\omega$ be the set of words containing only finitely many occurrences of a. It is easy to construct a nondeterministic automaton for the language. This automaton just guesses a position and checks that after this position there is no occurrence of a.

Fact 8 There is no deterministic Büchi automaton recognizing L_a.

Proof
Suppose conversely that $\mathcal{A} = \langle Q, \Sigma, q^0, \delta : Q \times \Sigma \to Q, F\rangle$ is a Büchi automaton for the language. Take the word b^ω. There is an accepting run of $\mathcal{A}$ on this word. Let i_1 be the position on which a state from F appears on the run. Consider now the word $b^{i_1}ab^\omega$. It is also accepted, and as the automaton is deterministic, the run is the same up to position i_1. Take a position $i_2 > i_1$ on which a state from F appears. Repeat this process $n = |Q| + 1$ times. We get a word $b^{i_1}ab^{i_2}a \dots b^{i_n}ab^\omega$ which is accepted by $\mathcal{A}$ and such that the run of $\mathcal{A}$ on this word has accepting states at positions i_j for $j = 1, \dots, n+1$. By the choice of n there are two positions, say i_k and i_l where the same state appears. We have that there is an accepting run on the word $b^{i_1}ab^{i_2}a \dots b^{i_k-1}(ba(b^{i_{k+1}}a \dots b^{i_l-1})^\omega$ □

There is a deterministic automaton with Mostowski conditions for L_a. This is an automaton that signals 1 when it reads a and 2 when it reads b. By the definition of the Mostowski condition this automaton accepts iff it signals 1 only finitely often.

This is not a coincident that there is a deterministic Mostowski automaton for L_a. The following important fact shows that deterministic automata with all but Büchi conditions have the same expressive power. In the next section we will see that they are equivalent to nondeterministic automata.

Theorem 9 (Mostowski [36])
For every deterministic Muller automaton there is an equivalent deterministic Mostowski automaton.

Proof
Take a Muller automaton $\mathcal{A} = \langle Q, \Sigma, 1, \delta, \mathcal{F}\rangle$ and assume that $Q = \{1, \dots, n\}$. The states of the Mostowski automaton $\mathcal{A}'$ will be permutations of Q with an additional index. Such a permutation is called *last appearance record* (LAR) and the position distinguished by the index is called *hit position*. So the set of LAR's is:

$$Q' = \mathrm{Perm}(\{1, \dots, n\}) \times \{1, \dots, n\}$$

The idea is that a LAR keeps the order between the last occurrences of states up to the present point. That is, if a state q appears before q' in the

permutation then the last occurrence of q up to the present position is before the last occurrence of q'. In particular the present state is on the last place of the permutation. The hit position shows what was the position of the present state in the previous permutation.

More formally, if the original automaton moves from a state l to a state l' then the simulating automaton $\mathcal{A}'$ is in a state $(i_1, \ldots, i_n, h)$, with $i_n = l$, and changes it to $(i_1, \ldots, i'_k, \ldots, i'_{n-1}l', k)$ where k is the position of l' in the sequence (i.e. $i_k = l$) and $i'_j = i'_{j+1}$ for $k \leq j < n$.

Let $r = q_0, q_1, \ldots$ be a run of the original automaton $\mathcal{A}$. Suppose that $F = \mathrm{In}(r)$ is the set of states appearing infinitely often in the run. Let us analyse the run of the automaton $\mathcal{A}'$ from the state $(n, \ldots, 1, 1)$. We will say that a state is an F-state if it is of the form $(i_1, \ldots, i_k, \ldots, i_n, h)$ with $\{i_k, \ldots, i_n\} = F$ and $h \geq k$.

Assume first that there is a position m such that only states from F appear after m and the state of $\mathcal{A}'$ at m is an F-state. It is easy to see from the definition of $\mathcal{A}'$ that after m all the states will be F-states. Moreover, the hit position will be k at some position after m. This is because the state i_k is going to appear after m. So, this argument really shows that the hit position will be k infinitely often after m.

Now, let us see why we are bound to reach such a position m as assumed in the previous paragraph. Let m_1 be a position after which no state outside F appears. Let $(i_1, \ldots, i_k, \ldots, i_n, h)$ be a state of $\mathcal{A}'$ at this position. Take a position m_2 such that between m_1 and m_2 all the states from F appeared at least once. We claim that $m_2 + 1$ is the required m. By the definition of the transition relation of $\mathcal{A}'$ in the permutation at position m_2 all the states from F occur after the states not in F. As the state q_{m_2} is from F the state of $\mathcal{A}'$ at the position m_2 will have the hit position $\geq k$.

So we have shown that if $q_0, q_1, \ldots$ is a run of $\mathcal{A}$ and F is the set of states appearing infinitely often on it then in the corresponding run of $\mathcal{A}'$ almost all states are F-states and the hit position is equal $k = n - |F| + 1$ infinitely often.

Now, we define the Mostowski acceptance condition on states of $\mathcal{A}'$. We put

$$\Omega(i_1, \ldots, i_n, h)) = \begin{cases} 2h & \text{if } \{i_h, \ldots, i_n\} \in \mathcal{F} \\ 2h+1 & \text{if } \{i_h, \ldots, i_n\} \notin \mathcal{F} \end{cases}$$

We claim that $\mathcal{A}'$ is equivalent to $\mathcal{A}$. Let F be a set of states appearing infinitely often on the run of $\mathcal{A}$. By the above considerations almost all states on the run of $\mathcal{A}'$ are F-states and the hit position is equal $k = n - |F| + 1$ infinitely often. Hence priority appearing infinitely often on the run of $\mathcal{A}'$ is either $2(n - |F| + 1)$ or $2(n - |F| + 1) + 1$. It is even iff $F \in \mathcal{F}$. $\square$

It is natural to ask what is the complexity of translating form one form of automaton to the next. From the above we can deduce:

Corollary 10 For every deterministic Muller, Streett or Rabin automaton with n states there is a deterministic Mostowski automaton with $\mathcal{O}(2^{n\log(n)})$ states.

A survey of the results on this subject is presented in [32]. In particular the bound in the corollary is essentially optimal.

3.1 Closure properties of ω-automata

It is easy to see that regular ω-languages are closed under sum. The construction is exactly the same as in the case of automata on finite words. This also true for the closure under projection. The closure under intersection is not that immediate. The construction from the case of finite words needs to be modified because now there is no last letter on which two runs can be synchronized.

Differences between finite and infinite words show up acutely in the case of closure under complement. In the case of automata on finite words a simple powerset construction is enough. In the case of ω-automata a very refined version of a powerset construction is required. Below, instead of complementation we consider the stronger property of determinization.

Theorem 11 (McNaughton)
For every Büchi automaton there is an equivalent deterministic Rabin automaton.

An immediate corollary of this result is the equivalence between MSOL and Büchi automata. Recall that by the results of the previous section all but deterministic Büchi automata are equivalent to (nondeterministic) Büchi automata.

Corollary 12 (Büchi) A language of ω-words is MSOL definable iff it is the language recognized by some Büchi automaton. The translations in both directions are effective.

Proof
The construction of a formula for a given automaton is almost the same as in the case of finite words. The proof in the other direction is also very similar. We build an automaton by induction on the syntax of a given formula. It is easy to construct automata for atomic formulas. As noted above, Büchi

automata are closed under sum and projection. The closure under complementation follows from Theorem 11 and Fact 6 saying that every Rabin automaton can be converted into a nondeterministic Büchi automaton. □

Actually, in 1962 when Büchi proved the above result he did not use determinization construction. The determinization construction was given in 1966 by McNaughton. At that time there was no notion of Rabin automaton. He has shown the determinization theorem for Muller automata. We follow here the construction given by Safra in 1988 [45]. This construction gives better complexity bounds than the original one. Later we will describe how to modify the construction to get an automaton with Mostowski conditions. The proof of the determinization theorem will take the rest of this subsection.

Let us start by examining why the standard subset construction does not work. Take a Büchi automaton:

$$\mathcal{A} = \langle Q, \Sigma, q_0, \delta, F \rangle$$

The states of the powerset automaton $\mathcal{A}^P$ are nonempty subsets of Q. We call these states *macro-states*. The transition function δ^P of this automaton is defined by $\delta^P(S,a) = \{q' : \exists q \in S.\ (q,a,q') \in \delta\}$. So $\mathcal{A}^P$ is a deterministic automaton and there is a run of $\mathcal{A}^P$ on w iff there is a run of $\mathcal{A}$ on w. The problem comes when we want to define an acceptance condition. A first attempt can be to define a Büchi acceptance condition F^P by taking all the macro-states S containing a state from F. The resulting automaton may accept too much. The problem is presented in part (a) of Figure 1. The big bubbles represent macro-states. The arrows show the transition relation of the original automaton. We assume that $F = \{q_f\}$. In case (a) there is no accepting run of the original automaton because there is no way to prolong a run from an accepting state.

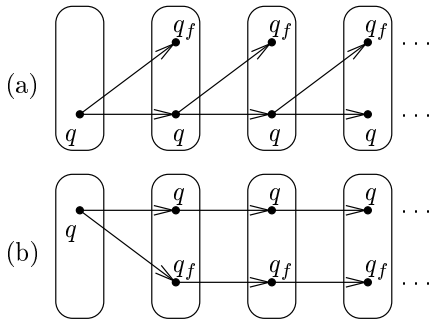

Figure 1: Problems with the powerset construction.

An alternative construction can extend powerset construction with recording some information about past of each state. As this will be a binary information we will use the metaphor of painting states. In the initial macro-state no state is painted. A state q' of a macro-state is painted green if $q' \in F$ or it is a successor of some green state, i.e., q' is obtained from q on the previous position and q was green in the previous macro-state. If the powerset automaton reaches a state with all the components painted green then it signals acceptance and removes the paint from all the components. Then the whole process repeats. An easy application of König's lemma shows that if such powerset automaton signals acceptance infinitely often then there is an accepting run of the original automaton. Unfortunately, as example (b) on Figure 1 shows, there may be an accepting run of the original automaton but the powerset automaton will not be a able to signal acceptance. The problem here is that apart from an accepting run we have a run that does not go through q_f at all. Hence at each macro-state there is a not painted state. If we were allowed to guess, than we would guess that we should restrict the above construction only to q_f and its successors. This would remove the upper part of the run from the considerations and the powerset automaton would accept. Because we are to construct a deterministic automaton we are not allowed to guess. What we will do is to consider all the possible guesses at the same time. We will need a clever way of keeping all these guesses together so that the sates do not get to big. This is the role of Safra trees defined below.

An ordered tree $(t, \leq)$ is a finite-tree with a partial order $\leq$ relating two nodes of the tree iff they are siblings, i.e., they have a common father. This ordering defines *to the left* relation on nodes of the tree: $\mathit{left}(u, v)$ holds if there are two siblings $u' \neq v'$ such that $u' \leq v'$ and u is a descendant of u' and v is a descendant of v' (we allow for $u = u'$ or $v = v'$).

A *Safra tree* is an ordered tree labelled with nonempty subsets of Q and colors white or green. The tree must also satisfying some coherence conditions. Formally such a tree is a quadruple $\tau = (t, \leq, \lambda : t \to \mathcal{P}(Q), c : t \to \{\text{white}, \text{green}\})$. The conditions are:

1. the label of the father is a proper superset of the sum of the labels of the sons,

2. the labels of two nodes which are not ancestral are disjoint.

The intuition is that the label of the root represents a macro-state from the powerset automaton. The rest of the tree describes a decomposition of the macro-state.

Before defining the transition function on Safra-trees, consider the lemma pointing out properties of the structure of Safra-trees and giving the bound on their size.

Lemma 13 There are at most $|Q|$ nodes in a Safra tree.

Proof
By condition 2, if a state q belongs to a label of a node then it can belong to the label of at most one son of the node. Hence there is a uniquely determined lowest node containing q. By condition 1, every node is the lowest node for some state. Hence, there cannot be more nodes than states. □

The deterministic transition function $\widehat{\delta}$ transforms a given tree τ on a given input a into a tree obtained by the following sequence of actions:

1. Set the color of all the nodes to white,
2. For every node v, create a new node v' with $\lambda(v') = \lambda(v) \cap F$. Make v' a son of v which is bigger than all the other sons of v.
3. For every node v, define the labeling $\lambda'(v) = \bigcup_{q \in \lambda(v)} \delta(q, a)$. So we apply the successor function of the given automaton.
4. For every node v, define the labeling $\lambda''(v) = \lambda'(v) \setminus \bigcup_{left(v_1, v)} \lambda'(v_1)$. So we delete a state from the label if it appears somewhere to the left.
5. Remove all the nodes with empty λ'' labels.
6. For every node v which λ'' label is equal to the sum of the λ'' labels of its sons, remove the descendants of v and color v green.

The resulting tree $\tau' = \widehat{\delta}(\tau, a)$ has the vertices of τ plus the vertices added in step 2 minus the vertices removed in step 6. The labeling is λ'' and the colouring is defined by the last step.

We will show later that $\mathcal{A}$ accepts a word iff there is the sequence of Safra trees constructed according to $\widehat{\delta}$ and a vertex v which is never deleted and which is coloured green infinitely often.

Before defining a Safra automaton we need to solve a small technical problem. In step 2 we add new vertices. We need to bound the number of vertices that can be added in order to have a finite number of Safra trees. We do this by recycling the vertices that are deleted in step 6. As we have observed before there can be at most $|Q|$ vertices in a Safra tree. Hence, we can take a pool of $2|Q|$ vertices. In 2 we take vertices from the pool and in step 6 we put them back. This way we have:

Lemma 14 There are $2^{\mathcal{O}(|Q|\log(|Q|))}$ Safra-trees.

The Safra automaton is $\widehat{\mathcal{A}} = \langle \widehat{Q}, \Sigma, \widehat{q}^0, \widehat{\delta}, \widehat{Acc} \rangle$ where $\widehat{Q}$ is the set of Safra trees over Q; $\widehat{q}^0$ is the tree consisting only of root labelled with $\{q^0\}$; and $\widehat{Acc}$ contains all the sequences such that there is a vertex which is removed only finitely often and lights green infinitely often on the sequence.

We now show that $L(\widehat{\mathcal{A}}) \subseteq L(\mathcal{A})$. Let $\tau_0, \tau_1, \ldots$ be an accepting run of $\widehat{\mathcal{A}}$ on some word w. Let v be a vertex that lights green infinitely often in the run and is not removed after some position m. Take a position i where v is green and a position $j > i$ such that v is not green between i and j. An easy induction shows that if $q' \in \lambda_j(v')$ for some v' a son of v in τ_j then there is $q \in \lambda_i(v)$ and a run of $\mathcal{A}$ on $w_i w_{i+1} \ldots w_j$ from q to q' going through some state in F. Call a run green if it goes through F. Let $i_1 < i_2 < \cdots$ be the sequence of positions after m where v lights green. From the above we have that for every $j = 1, 2, \ldots$ and every $q' \in \lambda_{i_{j+1}}(v)$ there is $q \in \lambda_{i_j}(v)$ and a green run of $\mathcal{A}$ on $w_{i_j} w_{i+1} \ldots w_{i_{j+1}}$ from q to q'. By Königs lemma we can find and infinite sequence $q_1, q_2, \ldots$ such that for every $j = 1, 2, \ldots$ there is a green run of $\mathcal{A}$ on $w_{i_j} w_{i+1} \ldots w_{i_{j+1}}$ form q_j to q_{j+1}. This together with the observation that there is a run of $\mathcal{A}$ from q^0 to q_1 on $w_0 \ldots w_{i_1}$ gives us an accepting run of $\mathcal{A}$ on w.

Next, we show that $L(\mathcal{A}) \subseteq L(\widehat{\mathcal{A}})$. Let q_0, q_1 be an accepting run of $\mathcal{A}$ on w. Consider the unique run $\tau_0, \tau_1, \ldots$ of $\widehat{\mathcal{A}}$ on w. By the definition of the transition function we have that $q_i \in \lambda_i(v_0)$ where v_0 is the root of all the Safra trees and λ_i is the labeling from the tree τ_i. If the root lights green infinitely often then the run of $\widehat{\mathcal{A}}$ is accepting and we are done. If not then let m_0 be the position where $q_{m_0} \in F$ and after which the root does not light green, Hence q_{m_0} appears in some son of v_0. An easy induction shows that for every $i > m_0$ the state q_i will appear in some son of v_0. It may move from one son to some other but this other son must be smaller in the tree ordering. As there is a bounded number of smaller sons of the root, there must be a son v_1 where the run of $\mathcal{A}$ stays forever, i.e., $q_i \in \lambda_i(v_1)$ for all positions bigger than some m_1. If v_1 lights green infinitely often then we are done as v_1 is not removed after m_1. If not then we repeat the reasoning. This way wa obtain a path $v_0, v_1, \ldots$. This path cannot be infinite because Safra trees have bounded size. Hence, there must be v_j such that for all positions i bigger than some m_j we have $q_i \in \lambda_i(v_j)$ and v_j lights green infinitely often.

Finally, it remains to show that $\widehat{Acc}$ is a Rabin acceptance condition. Recall that Safra trees were constructed over the fixed set $\{v_1, \ldots, v_{2|Q|}\}$ of vertices. For each $i = 1, \ldots, 2|Q|$ we take a pair (R_i, G_i) where R_i are all the Safra trees without v_i and G_i are all the Safra trees where v_i is coloured green. Then $\widehat{Acc}$ is expressed by the Rabin condition $\{(R_1, G_1), \ldots, (R_{2|Q|}, G_{2|Q|})\}$.

A more efficient strategy of vertex recycling would allow us to mange with a pool of $|Q|$ instead of $2|Q|$ vertices. This would reduce the number of pairs in the Rabin condition to $|Q|$.

Corollary 15 For every Büchi automaton with n states there is an equivalent deterministic Rabin automaton with $2^{\mathcal{O}(n \log(n))}$ states and n pairs in the acceptance condition.

This construction can be combined with LAR construction from Theorem 9 to obtain a deterministic Mostowski automaton. A direct application of the theorem would give a Mostowski automaton of doubly exponential size and with big values of function Ω. A closer look shows that it is enough to keep LARs of vertices and not of the whole Safra trees. As there are n vertices, there are $2^{\mathcal{O}(n \log(n))}$ LARs. So the resulting Mostowski automaton is of not much bigger size than the Rabin automaton.

Corollary 16 Every Büchi automaton with n states is equivalent to a deterministic Mostowski automaton with $2^{\mathcal{O}(n \log(n))}$ states and the range of the acceptance condition contained in $\{0, \ldots, 2n\}$.

3.2 Complexity

Let us shortly discuss the complexity of some problems for MSOL and automata on ω-words.

Checking emptiness of an ω-automaton is NLOGSPACE-complete. The lower bound follows from the case of automata on finite words. The upper bound is a modification of the reachability algorithm.

Checking universality is PSPACE-complete for automata on finite words. It is also PSPACE-complete for ω-automata of any the discussed kinds. The argument is essentially the same as for finite words. One constructs a deterministic automaton equivalent to the given one. The states of the automaton are calculated on demand.

As there is an effective translation from MSOL formulas to ω-automata, it follows that the satisfiability problem for MSOL over ω-words is decidable. The complexity of the problem is nonelementary. The lower bound follows from the satisfiability problem of FOL over finite words.

4 Infinite trees

In this section we extend the concept of automaton even further. We consider automata running on full infinite binary trees. We will show that these

automata enjoy the same closure properties as word automata. This will allow us to get the equivalence with MSOL over trees.

The *full binary tree* is the set $\{0,1\}^*$ of finite words over a two element alphabet. The root of the tree is the empty word ε. A node $w \in \{0,1\}^*$ has the left son $w0$ and the right son $w1$. A Σ-*labelled full binary tree* is a function $t : \{0,1\} \to \Sigma$. We use Trees(Σ) for the set of all Σ-labelled binary trees.

Similarly as for words, Σ-labelled trees can be represented as relational structures. A tree $t : \{0,1\} \to \Sigma$ is represented by:

$$\mathcal{M}_t = \langle \{0,1\}^*, \leq, s_0, s_1, (P_a)_{a \in \Sigma} \rangle$$

where $u \leq v$ holds if u is a prefix of v and s_0 and s_1 are the binary left and right son relations respectively. As before the predicates $(P_a)_{a \in \Sigma}$ code the labeling function t.

With such representations of trees we can say that a set of trees L is *monadic second-order (first-order) definable* iff $L = \{t : \mathcal{M}_t \models \varphi\}$ for some monadic second-order (respectively first-order) formula.

The idea of extending automata to trees is simple. Before, being in some position in a word the automaton had to decide which state to assume in the successor position. Now, as every node in a tree has two successors, the automaton splits and sends one copy of itself to each of the successors. Formally an automaton on trees is:

$$\mathcal{A} = \langle Q, \Sigma, q^0, \delta \subseteq Q \times \Sigma \times Q \times Q, Acc \rangle$$

where all the components but the transition relation δ are the same as for automata on words. Depending on whether Acc is Büchi-, Muller-, etc. condition we call $\mathcal{A}$ Büchi-, Muller-, etc. automaton.

A run of $\mathcal{A}$ on t is a function $r : \{0,1\}^* \to Q$ labeling nodes of the tree with states so that the root is labelled by the initial state, i.e., $r(\varepsilon) = q^0$; and for every node w and its sons $w0$ and $w1$ we have that

$$(r(w), t(w), r(w0), r(w1)) \in \delta.$$

A run r is *accepting* iff for every path P of the tree the sequence of states appearing on the path belongs to Acc. To put it more formally a path is a sequence of nodes $v_0, v_1, \ldots$ such that $v_0 = \varepsilon$ and v_{i+1} is a son of v_i, for every i. A run r is accepting iff for every such path the sequence $r(v_0), r(v_1), \ldots$ belongs to Acc. A tree is *accepted by* $\mathcal{A}$ iff there is an accepting run of $\mathcal{A}$ on it. The *language recognized by* $\mathcal{A}$ is the set of trees accepted by $\mathcal{A}$.

Example: We show a Büchi automaton for the language $L_a^\infty \in \mathrm{Trees}(\{a,b\})$ of trees having a path with infinitely many a's. The states of the automaton are $q_a, q_b, \top$. The transition relation is given by:

$$\begin{aligned}\delta(q_*, a) =& \{(q_a, \top), (\top, q_a)\}\\ \delta(q_*, b) =& \{(q_b, \top), (\top, q_b)\}\\ \delta(\top, *) =& \{(\top, \top)\}\end{aligned}$$

where in the above $*$ stands for either a or b. The Büchi acceptance condition is $F = \{q_a, \top\}$. Clearly the automaton accepts every tree from the state $\top$. It is easy to see that any run from q_a consists of a single path labelled with states q_a, q_b, and the rest of the tree labelled with $\top$. There are infinitely many states q_a on this path iff there are infinitely many vertices labelled by a. Hence the automaton accepts a tree iff it can find a path with infinitely many a's on it. $\square$

An important difference with the case of words is that Büchi conditions are weaker than the other types of conditions. The following fact was shown by Rabin [43] (see also [51])

Fact 17 The class of languages accepted by Büchi automata is not closed under complement. The complement of the language L_a^∞ from the example above is not recognizable by a nondeterministic Büchi automaton.

This fact explains why we use Rabin automata in the following:

Definition 18 A tree language $L \subseteq \mathrm{Trees}(\Sigma)$ is *regular* iff there is a Rabin automaton recognizing L.

4.1 Closure properties of tree automata

As in the preceding sections our goal is to show that the class of regular languages coincides with those definable in MSOL. For this we need to check the closure properties of regular tree languages. Essentially the same constructions as for word automata show that the class of regular tree languages is closed under sum and projection. As in the case of infinite words, it is the closure under complement that brings the biggest problems.

Theorem 19 (Rabin)
Regular tree languages are closed under complementation. For every Rabin tree automaton $\mathcal{A}$ one can effectively construct a Rabin automaton accepting the complement of $L(\mathcal{A})$.

Once we prove this theorem we get the equivalence between automata and MSOL. The proof of the corollary below is very similar to the case of words.

Corollary 20 A tree language is definable by a MSOL sentence iff it is the language recognized by some Rabin automaton. The translations in both directions are effective.

In the rest of the section we will sketch the proof of the complementation theorem. Observe that it does not say that Rabin automata can be determinized. Indeed the following easy fact shows that there is no hope for determinization.

Fact 21 Let $L_a^{\exists} \subseteq \text{Trees}(\{a, b\})$ be the set of trees having at least one vertex labelled with a. The language $L_a^{\exists}$ is not recognizable by a deterministic tree automaton with any of the considered acceptance conditions.

Although determinization is not possible, still the determinization result for word automata is essential in the complementation proof for tree automata. Except for this result we will need an important fact from the theory of infinite games. We will now define games abstractly and then make the connection to tree automata.

A *game* $G = \langle V, V_0, V_1, E \subseteq V \times V, Acc_G \subseteq V^\omega \rangle$ is a bipartite labelled graph with the partition (V_0, V_1) of the set of vertices V. We say that a vertex v' is a *successor* of a vertex v if $E(v, v')$ holds. The set Acc_G is used to determine the winner in a play of the game.

A *play* from some vertex $v_0 \in V_0$ proceeds as follows: first player 0 chooses a successor v_1 of v_0, then player 1 chooses a successor v_2 of v_1, and so on ad infinitum unless one of the players cannot make a move. If a player cannot make a move he looses. The result of an infinite play is an infinite path $v_0, v_1, v_2, \ldots$ This *path is winning* for player 0 if it belongs to Acc_G. The play from vertices of V_1 is defined similarly but this time player 1 starts.

A *strategy* σ for player 0 is a function assigning to every sequence of vertices $\vec{v}$ ending in a vertex v from V_0 a successor vertex $\sigma(\vec{v}) \in V_1$. A strategy is *memoryless* iff $\sigma(\vec{v}) = \sigma(\vec{w})$ whenever $\vec{v}$ and $\vec{w}$ end in the same vertex. A *strategy is winning* iff it guarantees a win for player 0 whenever he follows the strategy. Similarly we define a strategy for player 1.

In our application to tree automata we will be interested only in games with Acc_G given by Mostowski conditions. Such a condition is determined by a function $\Omega : V \to \mathbb{N}$ with the additional requirement that the image of Ω is finite. In the case of finite automata we did not have to make this

finiteness assumption because the set of states was finite by definition. Here we do not assume that the set V of vertices is finite.

Definition 22 A game with Mostowski conditions is given by a labelled graph $\langle V, V_0, V_1, E \subseteq V \times V, \Omega : V \to \mathbb{N} \rangle$ such that the image of Ω is a finite set.

The following is the main theorem about games with this kind of conditions. The idea of a strategy with bounded memory was introduced by Gurevich and Harrington [25]. They have shown a bounded memory theorem for Rabin conditions. The simplification for Mostowski conditions was proved independently by Emerson and Jutla [19] and by Mostowski [37].

Theorem 23 (Memoryless determinacy)
Let G be a game with Mostowski conditions. From every node of G one of the players has a memoryless winning strategy.

Now we can make the connection between tree automata and games. For a given automaton $\mathcal{A}$ and a given tree t we will define an acceptance game $G_{\mathcal{A},t}$. Player 0 will have a strategy in this game iff $t \in L(\mathcal{A})$. This way $t \notin L(\mathcal{A})$ is equivalent to player 1 having a strategy in $G_{\mathcal{A},t}$. By the above theorem, in this situation there is a memoryless strategy for player 1. The existence of such a strategy can be checked by a finite automaton. This will be the automaton accepting the complement of $L(\mathcal{A})$.

Fix an automaton $\mathcal{A}$ and a tree t. We define the game $G_{\mathcal{A},t}$. The idea of the game is quite simple. Player 0 will try to show that $\mathcal{A}$ accepts t. So, in each position of the game which is a current vertex of the tree and the current state of the automaton player 0 will chose a transition of the automaton. Player 1 will try to show that the choices suggested by player 0 are not correct. To this end he will point to direction, left or right, asking player 0 to provide the evidence for the respective subtree. The result of such play is an infinite path. Player 0 is the winner if the the sequence of states on this path satisfies the acceptance condition of $\mathcal{A}$, otherwise player 1 wins. Formally the game $G_{\mathcal{A},t}$ is defined by:

- the set V_0 of vertices for player 0 is $\{0,1\}^* \times Q$,
- the set V_1 of vertices for player 1 is $\{0,1\}^* \times (Q \times Q)$,
- from each vertex $(v, q) \in V_0$, for each transition $(q, a, q_0, q_1) \in \delta$ with $t(v) = a$ we have an edge to $(v, (q_0, q_1))$,
- from each vertex $(v, (q_0, q_1)) \in V_1$ we have edges to $(v0, q_0)$ and $(v1, q_1)$,

- the acceptance condition Acc_G consists of the sequences

$$(v_0, q_0)(v_0, m_0)(v_1, q_1)(v_1, m_1)\dots$$

such that the sequence $q_0 q_1 \dots$ is in Acc, i.e., it belongs to the acceptance condition of the automaton. (Here we use letters m_i to denote the elements of $Q \times Q$.)

Directly from the definition of the game we have that there is one to one correspondence between accepting runs of $\mathcal{A}$ on t and winning strategies for player 0 in $G_{\mathcal{A},t}$.

Lemma 24 $t \in L(\mathcal{A})$ iff player 0 has a winning strategy from the position (ε, q^0), i.e., the position consisting of the root of the tree and the initial state of $\mathcal{A}$.

Hence, by Theorem 23, $t \notin L(\mathcal{A})$ iff player 1 has a winning strategy in $G_{\mathcal{A},t}$. We will now construct an automaton $\mathcal{B}$ which accepts a tree t iff player 1 has a memoryless strategy in the game $G_{\mathcal{A},t}$. This way we will have that $L(\mathcal{B})$ accepts the complement of $L(\mathcal{A})$.

A memoryless strategy for player 1 is a function $\sigma_1 : V_1 \to V_0$ which for each vertex $(v, (q_0, q_1)) \in V_1$ chooses either $(v0, q_0)$ or $(v1, q_1)$. An important point is that such a function can be coded as a labeling function

$$f : \{0,1\}^* \to Moves_1$$

where $Moves_1$ is the finite set $((Q \times Q) \to \{0,1\})$.

Consider infinite words over the alphabet $\Sigma' = \Sigma \times Moves_1 \times \{0,1\}$. Such a word $\xi = ((a_i, f_i, d_i))_{i \in \mathbb{N}}$ describes a path $\varepsilon, d_0, d_0 d_1, \dots$ in the tree. Letter a_i is the label of the vertex $d_0 \dots d_{i-1}$, and f_i is the strategy of player 1 in this vertex. A *play staying* ξ is a sequence $(v_0, q_0)(v_0, m_0)(v_1, q_1)(v_1, m_1)\dots$ such that

- $v_0 = \varepsilon$ and $q_0 = q^0$,
- $m_i \in \delta(q_i, a_i)$,
- $f_i(m_i) = d_i$ and $q_{i+1} = q^{d_i}$ where $m_i = (q^0, q^1)$.

That is the strategy always suggests the directions along the path determined by ξ.

Let $\mathcal{C}$ be a nondeterministic automaton accepting precisely those words ξ over Σ' which have a play staying in ξ that is winning for player 0. It is quite easy to construct such an automaton from the description above.

By Theorem 16 there is a deterministic Rabin automaton $\overline{\mathcal{C}}$ accepting the complement of this language. That is $\overline{\mathcal{C}}$ accepts those words ξ for which all the plays staying ξ are winning for player 1.

Consider a tree t and a strategy function f. If f is not winning then there is a play which is winning for player 0 when player 1 uses the strategy defined by f. This play proceeds along some path of the tree. Hence automaton $\mathcal{C}$ would accept description of such a path. If f is winning then none of the paths is accepted by $\mathcal{C}$. Hence every path is accepted by $\overline{\mathcal{C}}$. So f is a winning strategy for player 1 iff each path of $t \times f$ (i.e. the tree t labelled additionally with the values of f) is accepted by $\overline{\mathcal{C}}$.

The tree automaton $\mathcal{B}$ accepting the complement of $\mathcal{A}$ consists of two automata. The first guesses the value of strategy a function f_1 in each vertex. The second runs $\overline{\mathcal{C}}$ on every path of the tree. Automaton $\mathcal{B}$ accepts iff $\overline{\mathcal{C}}$ accepts on all the paths.

4.2 Complexity

The complexity of decision problems for tree automata is more subtle than for word automata.

Recall that the emptiness problem, is to decide if there is a tree accepted by a given automaton. For Büchi automata the problem is PTIME-complete. For Rabin automata it is NP-complete [18]. As Streett conditions are negations of Rabin conditions, the emptiness problem for these automata is CO-NP-complete. The exact complexity of the emptiness problem for Mostowski automata is not known. The problem is in NP and CO-NP [17]. Determining whether the problem is in PTIME is one of the main open problems in the area.

The universality problem is EXPTIME-complete even for automata on finite trees [46]. The EXPTIME-completeness result carries over to all kinds of automata discussed in this section.

The satisfiability problem for MSOL on binary trees is nonelementary. The decidability of the problem follows from the effective translation to Rabin automata. The lower bound is inherited from that for first-order logic over finite words.

5 The μ-calculus and alternating automata

In the previous sections we have described very classical equivalences between monadic second-order logic and automata. Here we will present another logical formalism equivalent to the two. This will be the μ-calculus, an

extension of modal logic with fixpoint operators. We will show a very direct connection between the μ-calculus and alternating automata. These are an extension of nondeterministic automata with universal moves, in the same way as alternating Turing machines are an extension of nondeterministic Turing machines.

5.1 Syntax and semantics of the μ-calculus

Here we will introduce the μ-calculus over binary trees. Later, we will be also interested in the μ-calculus over words and arbitrary graphs. These variations can be easily obtained by changing the set of modalities.

Let $Var_2 = \{X, Y, \dots\}$ be the set of second order variables. Let $\{P_a : a \in \Sigma\}$ be the set of propositional letters. The syntax of the μ-calculus is given by the following grammar:

$$X \mid P_a \mid \neg\alpha \mid \alpha \vee \beta \mid \langle 0\rangle\alpha \mid \langle 1\rangle\alpha \mid \mu X.\alpha(X)$$

where in the last construct we require that X appears only positively (under even number of negations) in $\alpha(X)$. In this construct μ binds X. This has the same consequences for substitution as in the case of quantifiers in first-order logic (free variables of the formula being substituted should not be captured by the binders of the other formula). We write $\alpha[\beta/X]$ for the result of substituting the formula β for the variable X in the formula α.

A binary tree is represented as a structure $\mathcal{M} = \langle \{0,1\}^*, (P_a^{\mathcal{M}})_{a\in\Sigma}\rangle$. Previously we had the successor relations s_1 and s_2 in the signature. Now we do not need them as we do not have them in the syntax of the logic. The meaning of a sentence is a set of nodes of a tree. To define the meaning of a formula with free variables we need a valuation $V : Var_2 \to \mathcal{P}(\{0,1\}^*)$ assigning to each variable a set of nodes of the tree. The meaning $[\![\alpha]\!]_V^{\mathcal{M}}$ of a formula α in a tree $\mathcal{M}$ and valuation V is defined inductively as follows:

- $[\![X]\!]_V^{\mathcal{M}} = V(X)$
- $[\![P_a]\!]_V^{\mathcal{M}} = P_a^{\mathcal{M}}$
- $[\![\neg\alpha]\!]_V^{\mathcal{M}} = \{0,1\}^* \setminus [\![\alpha]\!]_V^{\mathcal{M}}$
- $[\![\alpha \vee \beta]\!]_V^{\mathcal{M}} = [\![\alpha]\!]_V^{\mathcal{M}} \cup [\![\beta]\!]_V^{\mathcal{M}}$
- $[\![\langle 0\rangle\alpha]\!]_V^{\mathcal{M}} = \{v : v0 \in [\![\alpha]\!]_V^{\mathcal{M}}\}$
- $[\![\langle 1\rangle\alpha]\!]_V^{\mathcal{M}} = \{v : v1 \in [\![\alpha]\!]_V^{\mathcal{M}}\}$

- $[\![\mu X.\alpha(X)]\!]^{\mathcal{M}}_V = \bigcap\{S \subseteq \{0,1\}^* : [\![\alpha]\!]^{\mathcal{M}}_{V[S/X]} \subseteq S\}$

So, the meaning of $\mu X.\alpha(X)$ is the least fixpoint of an operator assigning to a set S the set $[\![\alpha(X)]\!]^{\mathcal{M}}_{V[S/X]}$. By our assumption on the positivity of X, this operator is monotone, i.e., $[\![\alpha(X)]\!]^{\mathcal{M}}_{V[S_1/X]} \subseteq [\![\alpha(X)]\!]^{\mathcal{M}}_{V[S_2/X]}$ if $S_1 \subseteq S_2$. Hence, the least fixpoint always exists in the complete lattice of sets of tree nodes.

If α is a sentence then its meaning does not depend on the valuation. We will then write $[\![\alpha]\!]^{\mathcal{M}}$ or even $[\![\alpha]\!]$ if $\mathcal{M}$ is clear from the context. We will sometimes write $\mathcal{M}, v \vDash \alpha$ instead of $v \in [\![\alpha]\!]^{\mathcal{M}}$. A sentence α defines the language of trees

$$L(\alpha) = \{t : \mathcal{M}_t, \varepsilon \vDash \alpha\}$$

So a formula defines a set of trees for which it holds in the root node.

The greatest fixpoint operator, denoted $\nu X.\alpha(X)$ is definable using the least fixpoint by

$$\nu X.\alpha(X) = \neg\mu X.\neg\alpha(\neg X)$$

We will use σ to mean μ or ν.

Let us look at some example properties expressible in the μ-calculus. The formula $\mu X.P_a \vee \langle 0\rangle X \vee \langle 1\rangle X$ holds in the root of a tree whenever there is a node labelled by a. If there is such a node v then the path from ε to v belongs to the least fixpoint of the operator defined by the formula (any fixed point should contain v and should be closed under father relation). If there is no node labelled by a then the empty set is a fixpoint.

The formula $\nu X.P_b \wedge (\langle 0\rangle X \vee \langle 1\rangle X)$ holds in the root of a tree if there is an infinite path from ε every node of which is labelled with b. Indeed, if there is such a path then it is a fixpoint of the operator. Other way around, if S is a fixpoint of the operator and $\varepsilon \in S$ then ε is labeled by b and ε has a successor $v \in S$. Hence, inductively we can construct an infinite path with all the vertices labelled by b. Observe that if in the above formula we replaced the greatest fixpoint with the least, obtaining $\mu X.P_b \wedge (\langle 0\rangle X \vee \langle 1\rangle X)$, then we would get an unsatisfiable formula.

Sometimes it will be convenient to work with formulas in *positive normal form*, i.e., formulas were negations occur only before propositional constants and variables. The next lemma says that for this we need to add conjunction and the greatest fixpoint operators to the syntax.

Lemma 25 Every formula of the μ-calculus is equivalent to a formula in a positive normal form, possibly using conjunction and the greatest fixpoint operator.

Proof
A formula in a positive normal form can be obtained by repetitive uses of

de Morgan laws and the equivalences:

$$\neg\langle 0\rangle\alpha \equiv \langle 0\rangle\neg\alpha \qquad \neg\langle 1\rangle\alpha \equiv \langle 1\rangle\neg\alpha$$
$$\neg\ X.\alpha(X) \equiv \nu X.\neg\alpha(\neg X)$$

□

5.2 Alternating automata

Alternating automata model extends nondeterministic automata with a notion of universal moves. Here we will present alternating tree automata postponing presentation of variations for words and graphs to later sections.

An alternating tree automaton is a tuple:

$$\mathcal{A} = \langle Q, Q_\exists, Q_\forall, \Sigma, q^0, \delta : Q \times \Sigma \to \mathcal{P}(Q \times \{0, 1, \varepsilon\}), Acc\rangle$$

There are two differences with nondeterministic automata. First, the set Q of states is partitioned into existential and universal states, $Q_\exists$ and $Q_\forall$ respectively. Next, the transition relation is a function and ε-transitions are allowed. If an automaton is in an existential state q and in a vertex labelled by a then it chooses a transition from $\delta(q, a)$ which it is going to execute. If q is universal then the automaton has to execute all the transitions from $\delta(q, a)$. To execute a transition (q', d) in a vertex v means to go to the vertex vd and change the state to q'. So, if $d = \varepsilon$ then the automaton stays in v, if $d = 0$ it moves to the left son of v.

It is the simplest to formalize the notion of a run and acceptance of alternating automata in terms of games. Given a tree t we define the *acceptance game* $G_{\mathcal{A},t}$ which is very similar to the one defined for nondeterministic automata. We have:

- the set V_0 of vertices for player 0 is $\{0, 1\}^* \times Q_\exists$,
- the set V_1 of vertices for player 1 is $\{0, 1\}^* \times Q_\forall$,
- from each vertex (v, q) and $(q', d) \in \delta(q, t(v))$ there is an edge to (vd, q').
- the acceptance condition Acc consists of the sequences

  $$(v_0, q_0)(v_1, q_1)\ldots$$

  such that the sequence $q_0 q_1 \ldots$ is in Acc, i.e., it belongs to the acceptance condition of the automaton.

We say that $\mathcal{A}$ *accepts* a tree t iff player 0 has a winning strategy in the game $G_{\mathcal{A},t}$. The *language recognized by* $\mathcal{A}$ is the set of trees accepted by $\mathcal{A}$.

Example: An automaton expressing that there is a descendant labelled by a can have two states $q, \top$ and the transition function:

$$(q,b) \mapsto \{(q,0),(q,1)\} \qquad (q,a) \mapsto \{(\top,\varepsilon)\}$$

where $\top$ is a state from which every tree is accepted. We make q an existential state and put $\Omega(q) = 1$. When such an automaton starts from the state q in a vertex labelled by a then it goes to the accepting state $\top$. If the vertex is labelled by b then it goes with the state q to one of its successors and the process repeats. It cannot repeat like this for ever because then we would have a path consisting only of q states and such a path is not accepting. Hence in order to accept the automaton must finally meet a vertex labelled by a. If we changed the status of q from existential to universal then the automaton would accept from q precisely those trees which have a on every path. $\square$

The following lemma shows that the complementation is easy for alternating automata. It is also easy to show that alternating automata are closed under disjunction and conjunction. Hence it seems that they may be a better candidate to use in the proof of Rabin's theorem than nondeterministic automata where we need a lot of work to show the closure under complement. Unfortunately, it is difficult to show that alternating automata are closed under projection. One essentially needs to convert an alternating automaton into a nondeterministic automaton. This in turn is as complicated as the closure under complement for nondeterministic automata.

Lemma 26 For every Mostowski alternating automaton $\mathcal{A}$ there is a dual Mostowski automaton $\overline{\mathcal{A}}$, such that $L(\overline{\mathcal{A}}) = \mathrm{Trees}(\Sigma) \setminus \mathcal{A}$. The size of $\overline{\mathcal{A}}$ is the same as the size of $\mathcal{A}$.

Proof
Let $\mathcal{A} = \langle Q, Q_\exists, Q_\forall, \Sigma, q^0, \delta, \Omega\rangle$ be an alternating automaton as above. The dual automaton is

$$\overline{\mathcal{A}} = \langle Q, Q_\forall, Q_\exists, \Sigma, q^0, \delta, \overline{\Omega}\rangle$$

where $\overline{\Omega}(q) = \Omega(q) + 1$ for every $q \in Q$. Hence, $\overline{\mathcal{A}}$ has the same set of states and the same transition function as $\mathcal{A}$. The difference is that the roles of existential and universal states are interchanged. Also the acceptance condition is negated.

To see that $L(\overline{\mathcal{A}})$ is the complement of $L(\mathcal{A})$ consider a tree $t \in \mathrm{Trees}(\Sigma)$ and the acceptance games $G_{\mathcal{A},t}$, $G_{\overline{\mathcal{A}},t}$. The graphs of these games are the

same but every position for player 0 in $G_{\mathcal{A},t}$ becomes a position for player 1 in $G_{\overline{\mathcal{A}},t}$ and vice versa. Moreover, for every infinite play in these games, the play is winning for player 0 in $G_{\mathcal{A},t}$ iff it is winning for player 1 in $G_{\overline{\mathcal{A}},t}$. Hence, player 0 has a winning strategy in $G_{\mathcal{A},t}$ iff player 1 has a winning strategy in $G_{\overline{\mathcal{A}},t}$ (this is just the same strategy). So, $t \in L(\mathcal{A})$ iff $t \notin L(\overline{\mathcal{A}})$. □

5.3 From the μ-calculus to alternating automata

The translation from the μ-calculus to alternating automata is quite direct. The interesting and not obvious part of the translation is the construction of acceptance conditions. The presented translation is based on the ideas from [50, 48]

Let us fix a sentence α in a positive normal form and such that each variable is bound at most once in α. Let $cl(\alpha)$ stand for the set of subformulas of α. As every variable is bound at most once, we can use β_X for the unique subformula such that $\sigma X.\beta_X \in cl(\alpha)$. If σ is μ then we call X a *μ-variable*, otherwise we call X a *ν-variable*. We construct an alternating automaton:

$$\mathcal{A}^\alpha = \langle Q^\alpha, Q^\alpha_\exists, Q^\alpha_\forall, \Sigma, \alpha, \delta^\alpha, Acc^\alpha \rangle$$

where $Q^\alpha = cl(\alpha) \cup \{\top, \bot\}$ is the set of subformulas of α with two additional states. The intended meaning of the additional states is that the automaton should accept everything from $\top$ and it should accept nothing from $\bot$. The initial state is the formula α itself. The partition of states is such that $Q_\exists$ contains all disjunctions (formulas of the form $\beta_1 \vee \beta_2$) and $Q_\forall$ contains the rest. When we will define transition function, it will be clear that from states other than disjunctions and conjunctions there is no choice. So it is irrelevant whether we put these states in $Q_\exists$ or $Q_\forall$.

The transition function is defined by:

- $\delta(P_a, a) = \{(\top, \varepsilon)\}$,
- $\delta(P_a, b) = \{(\bot, \varepsilon)\}$ for $a \neq b$,
- $\delta(X, a) = \{(\beta_X, \varepsilon)\}$,
- $\delta(\beta_1 \vee \beta_2, a) = \{(\beta_1, \varepsilon), (\beta_2, \varepsilon)\}$ and the same for conjunction,
- $\delta(\langle 0 \rangle \beta, a) = \{(\beta, 0)\}$ and similarly for 1,
- $\delta(\sigma X.\beta, a) = (\beta, \varepsilon)$,

Hence, the automaton just decomposes the formula and, in the case of modal formulas, it proceeds in an appropriate direction. In the case of a proposition P_a it checks the labeling of the current node. If the label is a then it accepts the rest of the tree, otherwise it rejects. In the case of a variable, it replaces it by its fixpoint definition. We have not shown the obvious transitions from $\top$ and $\bot$.

Before defining the acceptance condition we need one more notion.

Definition 27 The *dependence order* on bound variables of α is the smallest partial order such that $X <_\alpha Y$ if X occurs free in $\sigma Y.\beta_Y$. The *alternation depth* of a variable X, denoted $\mathrm{adepth}(X)$, is the maximal number of alternations between μ and ν-variables in a chain $X <_\alpha Z_1 <_\alpha \cdots <_\alpha Z_k <_\alpha Y$.

For example, in the formula $\mu X.(\nu Y.\langle 0\rangle Y) \wedge \langle 1\rangle X$ the alternation depths of both X and Y are 0. This is because X is not smaller than Y in the dependence order as it does not occur free in the Y subformula. On the other hand, in $\mu X.(\nu Y.\langle 1\rangle X \wedge \langle 0\rangle Y)$ the alternation depth of X is 1.

The acceptance condition Acc_α is the Mostowski condition $\Omega_\alpha : Q_\alpha \to \mathbb{N}$ defined by

$$\Omega_\alpha(\beta) = \begin{cases} 2 * (\max_d - \mathrm{adepth}(X)) & \text{if } \beta = X \text{ is a } \nu\text{-variable} \\ 2 * (\max_d - \mathrm{adepth}(X)) + 1 & \text{if } \beta = X \text{ is a } \mu\text{-variable} \\ M & \text{otherwise} \end{cases}$$

where $\max_d$ is the maximal alternation depth of a variable from $cl(\alpha)$, and M is the maximal value of Ω_α for variables. The intention is that meeting a ν-variable brings us closer to acceptance while meeting a μ-variable take us further away. The values of Ω_α for formulas other than variables do not matter.

The correctness of the construction is stated in the next theorem.

Theorem 28
For every tree t represented by a structure $\mathcal{M}_t$: $\mathcal{M}_t, \varepsilon \vDash \alpha$ iff $t \in L(\mathcal{A}_\alpha)$.

In the rest of the subsection we will sketch the proof of the theorem. The main point is to understand the behaviour of fixpoints. We do this by introducing fixpoint approximations. This will allow to use an induction argument on approximations.

Definition 29 An *approximation* of a formula $\mu X.\beta(X)$ has the form $\mu^\tau X.\beta(X)$

for some ordinal τ. The meaning of such an approximation is defined by:

$$[\![\mu^0 X.\beta(X)]\!]_V^{\mathcal{M}} = \emptyset \qquad [\![\mu^{\tau+1} X.\beta(X)]\!] = [\![\beta(X)]\!]^{\mathcal{M}}_{V[[\![\mu^\tau X.\beta(X)]\!]^{\mathcal{M}}_V/X]}$$
$$[\![\mu^\tau X.\beta(X)]\!]_V^{\mathcal{M}} = \bigcup_{\tau'<\tau} [\![\mu^{\tau'} X.\beta(X)]\!]_V^{\mathcal{M}} \quad \text{if } \tau \text{ is a limit ordinal}$$

Similarly we define approximations $\nu^\tau X.\beta(X)$ of ν-formulas.

Approximations are not themselves formulas of the μ-calculus. They are extensions of the syntax needed in the inductive argument showing correctness of our automaton. Recall that by Knaster-Tarski theorem we have:

$$[\![\mu X.\beta(X)]\!]_V^{\mathcal{M}} = \bigcup_{\tau \in Ord} [\![\mu^\tau X.\beta(X)]\!]_V^{\mathcal{M}}$$

Here the sum is over all ordinals, but it is enough to stop at the first ordinal whose cardinality is bigger than the cardinality of $\mathcal{M}$.

Definition 30 Let $X_1, \ldots, X_k$ be all the variables from α listed in the order respecting $<_\alpha$ relation (with smaller variables first). For a sequence of ordinals $\vec{\tau} = (\tau_1, \ldots, \tau_k)$ and a formula $\gamma \in cl(\alpha)$ we define:

$$\langle\!|\gamma|\!\rangle^\mu_{\vec{\tau}} = \gamma[\sigma'_k X_k.\beta_k/X_k] \ldots [\sigma'_1 X_1.\beta_1/X_1]$$

where $\sigma'_m = \nu$ if X_m is a ν-variable and $\sigma'_m = \mu^{\tau_m}$ otherwise. We define $\langle\!|\gamma|\!\rangle^\nu_{\vec{\tau}}$ similarly but with the roles of μ and ν-variables interchanged.

So $\langle\!|\gamma|\!\rangle^\mu_{\vec{\tau}}$ is the result of replacing sequentially every free variable by a fixpoint formula or its approximation.

Definition 31 If $v \in [\![\gamma]\!]_V^{\mathcal{M}}$ for some $\gamma \in cl(\alpha)$ then we define the *μ-signature*, $\mathrm{Sig}(\gamma, v)$, of γ in the vertex v of the model $\mathcal{M}$ to be the least in the lexicographical ordering tuple of ordinals $\vec{\tau}$ such that $v \in [\![\langle\!|\gamma|\!\rangle^\mu_{\vec{\tau}}]\!]_V^{\mathcal{M}}$.

If $v \notin [\![\gamma]\!]_V^{\mathcal{M}}$ then the *ν-signature* of γ, $\mathrm{Sig}^\nu(\gamma, v)$, is the least in the lexicographical ordering tuple of ordinals $\vec{\tau}$ such that $v \notin [\![\langle\!|\gamma|\!\rangle^\nu_{\vec{\tau}}]\!]_V^{\mathcal{M}}$.

Using Knaster-Tarski theorem one can check that μ and ν-signatures always exists when the conditions of the definition are satisfied. Having signatures we can formulate the signature decrease lemma which is the main tool in proving correctness of our automaton.

Lemma 32 (Signature decrease) For every vertex v, whenever the left hand-sides are defined we have:

- $\mathrm{Sig}(\beta_1 \wedge \beta_2, v) = \max(\mathrm{Sig}(\beta_1, v), \mathrm{Sig}(\beta_2, v))$.
- $\mathrm{Sig}(\beta_1 \vee \beta_2, v) = \mathrm{Sig}(\beta_1, v)$ or $\mathrm{Sig}(\beta_1 \vee \beta_2, v) = \mathrm{Sig}(\beta_2, v)$.
- $\mathrm{Sig}(\langle 0 \rangle \beta, v) = \mathrm{Sig}(\beta, v0)$ and similarly for 1.
- $\mathrm{Sig}(\nu X.\beta(Y), v) = \mathrm{Sig}(\beta(Y), v)$.
- $\mathrm{Sig}(\mu X_i.\beta(X_i), v)$ is the same as $\mathrm{Sig}(\beta(X_i), v)$ on the first $i-1$ positions.
- If Y is a ν-variable, $\mathrm{Sig}(Y, v) = \mathrm{Sig}(\beta_Y(Y), v)$.
- If X is a ν-variable, $\mathrm{Sig}(X_i, v)$ is bigger than $\mathrm{Sig}(\beta_{X_i}(X_i), v)$ and the difference is at position i.

Similarly for ν-signatures but with interchanged roles of μ with ν, and conjunction with disjunction.

Using the signature decrease lemma we can show that if $\mathcal{M}_t, \varepsilon \vDash \alpha$ then $t \in L(\mathcal{A}_\alpha)$. For this we show that there is a winning strategy for player 0 in the acceptance game $G(\mathcal{A}_\alpha, t)$. The only choice for player 0 in this game is in the case of a disjunction $\beta_1 \vee \beta_2$. In a position $(v, \beta_1 \vee \beta_2)$ he should choose (v, β_1) if $\mathrm{Sig}(\beta_1 \vee \beta_2, v) = \mathrm{Sig}(\beta_1, v)$ and (v, β_2) otherwise.

To see that such a strategy is winning for player 0 assume conversely that there is a play on which some odd priority p is the least priority appearing infinitely often. This means that on this play we infinitely often meet the μ-variable X_l where $l = (p-1)/2$. Moreover the variables with indices smaller than l appear only finitely many times on the play. Let m be a step of the play after which no such variable with smaller index appears. By the signature decrease lemma, the signatures of positions of the play after m never increase on the first l positions. They decrease every time we meet X_l. But this is impossible as the lexicographic order on l-tuples of ordinals is a well ordering. Hence, such a play cannot exist, and the strategy we have defined is winning for player 1.

The proof of the theorem in the other direction is very similar. We show that if $\mathcal{M}_t, \varepsilon \nvDash \alpha$ then $t \notin L(\mathcal{A}_\alpha)$. For this we present a winning strategy for player 1 in $G(\mathcal{A}_\alpha, t)$. Player 1 has a choice only in case of conjunction. In a position $(v, \beta_1 \wedge \beta_2)$ he should choose (v, β_1) if $\mathrm{Sig}^\nu(\beta_1 \vee \beta_2, v) = \mathrm{Sig}(\beta_1, v)$ and (v, β_2) otherwise.

5.4 From alternating automata to the μ-calculus

In the translation from alternating automata to the μ-calculus it will be convenient to use vectorial syntax as an intermediate step. Hence, we start

with a definition of μ-formulas in vectorial form. As it will turn out later, every μ-calculus formula is equivalent to the one in this form.

Let $n \in \mathbb{N}$. An n-array μ-calculus formula has the form

$$\sigma_1 \begin{pmatrix} X_1^1 \\ \vdots \\ X_n^1 \end{pmatrix} \ldots \sigma_m \begin{pmatrix} X_1^m \\ \vdots \\ X_n^m \end{pmatrix} . \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix}$$

where m is some integer; $\sigma_1, \ldots, \sigma_m$ are fixpoint operators; and $\alpha_1, \ldots, \alpha_n$ are formulas of the ordinary (scalar) μ-calculus without fixpoint operators.

The semantics of such a formula in a tree $\mathcal{M}$ and a valuation V is a set of n-tuples of vertices of $\mathcal{M}$. For a vectorial formula without fixpoints we have

$$[\![(\alpha_1, \ldots, \alpha_n)]\!]_V^{\mathcal{M}} = [\![\alpha_1]\!]_V^{\mathcal{M}} \times \cdots \times [\![\alpha_n]\!]_V^{\mathcal{M}}$$

For a fixpoint we have

$$[\![\mu \vec{X}.\vec{\beta}]\!]_V^{\mathcal{M}} = \bigcap \{\vec{S} \subseteq (\{0,1\}^*)^n : [\![\vec{\beta}]\!]_{V[\vec{S}/\vec{X}]}^{\mathcal{M}} \subseteq \vec{S}\}$$

So, this is a fixpoint of an operator over sets of n-tuples of vertices. The greatest fixpoint is defined similarly.

Finally, we introduce the projection operation. If $\vec{\beta}$ is an n-array formula then we write $(\vec{\beta})\downarrow_1$ for the value of the first component. So, $[\![(\vec{\beta})\downarrow_1]\!]_V^{\mathcal{M}}$ is the first component of $[\![\vec{\beta}]\!]_V^{\mathcal{M}}$.

A first useful observation is that vectorial μ-calculus is not more powerful than the ordinary one

Lemma 33 For every vectorial formula $\vec{\beta}$ there is a formula α of the ordinary (scalar) μ-calculus such that for every tree $\mathcal{M}$ and valuation V we have: $[\![(\vec{\beta})\downarrow_1]\!]_V^{\mathcal{M}} = [\![\alpha]\!]_V^{\mathcal{M}}$.

Proof
The proof is a rather tedious application of Bekic principle:

$$\left(\sigma \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} . \begin{pmatrix} \alpha_1(X_1, X_2) \\ \alpha_2(X_1, X_2) \end{pmatrix}\right)\downarrow_1 = \ \sigma X_1.\ \alpha_1(X_1, \sigma X_2.\ \alpha_2(X_1, X_2))$$

that is, the meaning of the first component of the formula on the left is the same as the meaning of the formula on the right. This principle holds for every complete lattice; hence in all interpretations we consider in these notes. $\square$

Let $\mathcal{A}$ be an alternating automaton with a Mostowski acceptance condition given by a function $\Omega : Q \to \mathbb{N}$. Let $q_1, \ldots, q_n$ be an ordering of the states of $\mathcal{A}$ such that $\Omega(q_i) \leq \Omega(q_j)$ for every $i < j$. The vectorial formula corresponding to $\mathcal{A}$ is

$$\alpha_{\mathcal{A}} = \sigma_1 \vec{X}_1 \ldots \sigma_n \vec{X}_n . \begin{pmatrix} \bigvee_{a \in \Sigma} (P_a \wedge \lceil \delta(q_1, a) \rceil) \\ \vdots \\ \bigvee_{a \in \Sigma} (P_a \wedge \lceil \delta(q_n, a) \rceil) \end{pmatrix}$$

where σ_i is μ if $\Omega(q_i)$ is odd and σ_i is ν otherwise. We also need to explain what $\lceil \delta(q_i, a) \rceil$ stand for. Recall that $\delta(q_i, a)$ is a subset of $Q \times \{0, 1, \varepsilon\}$. For a pair $(q_j, d) \in Q \times \{0, 1, \varepsilon\}$ we put:

$$\lceil (q_j, a) \rceil = \begin{cases} \langle 0 \rangle X_j^j & \text{if } d = 0 \\ \langle 1 \rangle X_j^j & \text{if } d = 1 \\ X_j^j & \text{if } d = \varepsilon \end{cases}$$

then we put

$$\lceil \delta(q_i, a) \rceil = \begin{cases} \bigvee \{ \lceil (q', d') \rceil : (q', d') \in \delta(q_i, a) \} & \text{if } q_i \in Q_\exists \\ \bigwedge \{ \lceil (q', d') \rceil : (q', d') \in \delta(q_i, a) \} & \text{if } q_i \in Q_\forall \end{cases}$$

Theorem 34
For every tree t: $t \in L(\mathcal{A})$ iff $\mathcal{M}_t, \varepsilon \vDash (\alpha_{\mathcal{A}}) \downarrow_1$

The proof is similar to the one for the translation form formulas to automata. It also uses the signature decrease lemma.

5.5 The μ-calculus and alternating automata over graphs

Originally [31] the μ-calculus was defined over arbitrary directed graphs and not just binary trees. In this setting, instead of $\langle 0 \rangle$ and $\langle 1 \rangle$ modalities it has one modality $\langle \cdot \rangle$. The models are Σ-labelled graphs $G = \langle V, E \subseteq V \times V, \lambda : V \to \Sigma \rangle$. Such a graph can be represented as a structure $\mathcal{M}_G = \langle V, E, (P_a)_{(a \in \Sigma)} \rangle$. The meaning of the modality is:

- $[\![\langle \cdot \rangle \alpha]\!]_V^{\mathcal{M}} = \{v : \exists v' .\ E(v, v') \text{ and } v' \in [\![\alpha]\!]_V^{\mathcal{M}}\}$

The rest of the clauses is the same as for the μ-calculus over binary trees. A new thing in the present situation is that the modality is "nondeterministic", i.e., there are several possible v' in the semantical clause above. Using negation we can define the dual modality $[\cdot]\alpha = \neg \langle \cdot \rangle \neg \alpha$. So we have:

- $[\![[\cdot]\alpha]\!]_V^{\mathcal{M}} = \{v : \forall v'.\ E(v,v') \Rightarrow v' \in [\![\alpha]\!]_V^{\mathcal{M}}\}$

We can also extend the notion of alternating automata to Σ-labelled graphs. Such an automaton has a form:

$$\mathcal{A} = \langle Q, \Sigma, Q_\exists, Q_\forall, q^0, \delta : Q \times \Sigma \to \mathcal{P}(Q \times \{\langle\cdot\rangle, [\cdot], \varepsilon\}), Acc\rangle$$

The only difference with the automaton on binary trees is in the transition function. Before we had 0 and 1 and now we have $\langle\cdot\rangle$ and $[\cdot]$. At first it may seem that $\langle\cdot\rangle$ should be enough, but we do not have negation in automata so we would have no means to express $[\cdot]$. Before we did not need negation of $\langle 0\rangle$ because over binary trees it was expressible using $\langle 0\rangle$ itself. The acceptance of such automata is defined using games almost the same way as before. Given a Σ labelled graph $\mathcal{M}$ we have the game $G_{\mathcal{A},\mathcal{M}}$:

- the set V_0 of vertices for player 0 is $\{0,1\}^* \times (Q_\exists \cup Q \times \{\langle\cdot\rangle, \varepsilon\})$,
- the set V_1 of vertices for player 1 is $\{0,1\}^* \times (Q_\forall \cup Q \times \{[\cdot]\})$,
- from a vertex (v,q), for each $(q',d') \in \delta(q,\lambda(v))$ there is an edge to $(v,(q',d'))$
- form a vertex $(v,(q',\varepsilon))$ there is an edge to (v,q'),
- form a vertex $(v,(q',\langle\cdot\rangle))$ or $(v,(q',[\cdot]))$ there is an edge to (v',q') for every successor v' of v.
- the acceptance condition Acc_G consists of the sequences

  $$(v_0,q_0),(v_0,(q_1,d_1)),(v_1,q_1),(v_1,(q_2,d_2)),(v_2,q_2),\ldots$$

  such that the sequence $q_0q_1\ldots$ is in Acc, i.e., it belongs to the acceptance condition of the automaton.

The difference with the game for binary trees is that now we have an additional round. If a game reaches a position $(v,(q',\langle\cdot\rangle))$ then player 0 chooses the successor of v. In a position $(v,(q',[\cdot]))$ the choice is made by player 1.

Very similar translations to the previous ones show

Theorem 35
The μ-calculus over Σ-labelled graphs is equivalent to the alternating automata. The translations in both directions are effective.

5.6 Relation to MSOL: binary trees

An interesting question is to compare the μ-calculus and alternating automata with MSOL. Given all the facts on automata and MSOL that we have seen till now, it is not difficult to show that over words and trees the formalisms are the same.

Consider the following translation of the μ-calculus over binary trees into MSOL:

$$\begin{aligned} P_a &\rightsquigarrow P_a(x) \\ X &\rightsquigarrow X(x) \\ \alpha \vee \beta &\rightsquigarrow \varphi_\alpha(x) \vee \varphi_\beta(x) \\ \neg\alpha &\rightsquigarrow \neg\varphi_\alpha(x) \\ \langle 0\rangle\alpha &\rightsquigarrow \exists y.s_0(x,y) \wedge \varphi_\alpha(y) \\ \mu X.\alpha(X) &\rightsquigarrow \forall Z.(\forall y.\ \varphi_\alpha(Z,y) \Rightarrow Z(y)) \Rightarrow Z(x) \end{aligned}$$

This translation produces a MSOL formula with one free first order variable x and the same free second-order variables as in the starting formula. It is not difficult to prove by induction on a formula α that for every tree t, vertex v of t and valuation $V : Var_2 \to \mathcal{P}(\{0,1\}^*)$ of second order variables we have:

$$v \in [\![\alpha]\!]_V^{\mathcal{M}_t} \quad \text{iff} \quad \mathcal{M}_t, V \vDash \varphi_\alpha(v)$$

This shows that whenever a set of trees is definable by a μ-calculus formula then it is definable by a MSOL formula.

The translation into the other direction goes through nondeterministic automata. We know that for every MSOL formula over binary trees there is an equivalent nondeterministic automaton. A nondeterministic automaton is a special case of alternating automaton. For every alternating automaton there is an equivalent μ-calculus formula. Summarizing we get:

Corollary 36 Over binary trees MSOL, the μ-calculus, alternating automata and nondeterministic automata have the same expressive power. This equivalence holds also over finite and infinite words.

From the presented translations it follows that MSOL is the most succinct of the formalism. For a formula of the μ-calculus or for an automaton there is an equivalent MSOL formula of a linear size. It can be shown that every translation the other way must produce results of nonelementary size. As we have seen, the translations between alternating automata and vectorial μ-calculus are linear. The known translations to scalar μ-calculus produce an exponential blowup. The translation from alternating to nondeterministic automata has an exponential lower and upper bound.

5.7 Relation to MSOL: graphs

An interesting question is whether we can relate the μ-calculus and MSOL over graphs. We cannot hope to have exact equivalence. As we will see μ-calculus sentences cannot distinguish between bisimilar models while sentences MSOL sometimes can. Still, we get a surprisingly strong connection.

A *bisimulation* between two Σ-labelled graphs $G_1 = \langle V_1, E_1, \lambda_1 \rangle$ and $G_2 = \langle V_2, E_2, \lambda_2 \rangle$ is a relation $R \subseteq V_1 \times V_2$ such that whenever $(v_1, v_2) \in R$ then:

- $\lambda(v_1) = \lambda(v_2)$,
- for each successor v_1' of v_1 there is a successor v_2' of v_2 with $(v_1', v_2') \in R$,
- the same with v_1 and v_2 interchanged.

Fact 37 Every μ-calculus sentence is invariant under bisimulation. That is if $\mathcal{M}, v \vDash \alpha$ and there is a bisimulation on $\mathcal{M} \times \mathcal{M}'$ relating v ad v' then $\mathcal{M}, v' \vDash \alpha$.

Proof
The proof is quite easy for alternating automata. We get the thesis using the correspondence from Theorem 35. □

Fact 38 There is an MSOL sentence which is not invariant under bisimulation.

Proof
Just consider a sentence saying that a model is a tree and its root has exactly two successors. □

So the most we can expect is that every MSOL sentence that is invariant under bisimulation is equivalent to a μ-calculus sentence. This is indeed the case. Before stating this theorem let us examine the power of MSOL on trees of arbitrary degree (i.e., not only binary).

A graph is a tree iff to every node there is a unique finite path from the distinguished vertex called the root. A counting alternating automaton is an extension of an alternating automaton with the ability to count the number of successors. It is of the form:

$$\mathcal{A} = \langle Q, \Sigma, Q_\exists, Q_\forall, q^0, \delta : Q \times \Sigma \to \mathcal{P}(Q \times \{\varepsilon, \langle n \rangle, [n] : n \in \mathbb{N}\}), Acc \rangle$$

The intuitive meaning of $(q, \langle n \rangle)$ move is that the automaton needs to find n distinct successors to which it should go with the state q. The meaning for $(q, [n])$ is that it should go to all but n successors with the state q. We have [53]

Theorem 39
Counting alternating automata characterize the expressive power of MSOL on trees of arbitrary degree.

Clearly counting alternating automata can distinguish between bisimilar structures. From every counting alternating automaton we can produce an ordinary one by changing each $\langle n \rangle$ and $[n]$ into $\langle \cdot \rangle$ and $[\cdot]$ respectively. An interesting thing is that if the counting automaton happen to accept a bisimulation invariant language then a slight improvement of this simple translation would produce an equivalent non-counting alternating automaton. It in turn can be translated in to a μ-calculus formula. This is roughly the way to show the following correspondence [30].

Theorem 40
A bisimulation invariant property of graphs is MSOL definable iff it is μ-calculus definable.

5.8 Complexity

We briefly summarize the complexity of some decision problems for the μ-calculus and alternating automata.

The satisfiability problem for the μ-calculus is to decide whether a given formula has a model. The problem is EXPTIME-complete [50, 18]. For the lower bound one can reduce the problem of universality of tree automata. For the upper bound one can use the translation to alternating automata.

The emptiness problem for alternating automata is EXPTIME-complete. The EXPTIME algorithm is to translate the automaton into a nondeterministic automaton and check the emptiness of the result. An important point here is that although the automaton grows exponentially in the translation, the acceptance conditions do not [39, 18].

Another important problem is the *model checking* problem: "For a finite graph G and a μ-calculus formula α decide if α holds in the given vertex of G". The problem is equivalent to the emptiness problem for Mostowski alternating tree automata over one letter alphabet [17]. The later problem is in turn equivalent to the emptiness problem for nondeterministic Mostowski tree automata. Hence, its complexity is in NP$\cap$CO-NP but no deterministic polynomial time algorithm is known.

6 Hierarchies

As we have seen, the connections between MSOL, the μ-calculus and alternating automata are quite strong. Here we will refine the connections by considering relations between the hierarchies. For each of the three formalisms there are some "obvious" ways of defining hierarchies. For MSOL we can consider the quantifier alternation hierarchy. For the μ-calculus this will be the fixpoint alternation hierarchy. For alternating automata we can measure the complexity of the acceptance condition.

6.1 Definitions

MSOL First, we define the hierarchy for MSOL. As it will turn out a notion of weak quantification is relevant here. A weak quantifier is a quantifier ranging over finite sets. We can write a formula $\exists^w X.\ \varphi(X)$ with the meaning that there is a finite set S for which $\varphi(S)$ holds. Over binary trees finiteness is definable so addition of weak quantification does not extend the power of the logic over this model. Let WMSOL be the fragment of MSOL using only first-order and weak second order quantification.

Consider MSOL sentences of the form $Q_1 X_1 \dots Q_n X_n.\ \varphi$ where φ is a WMSOL formula and $Q_1 \dots Q_n$ are second-order quantifiers. Define the classes $\Sigma_0^M = \Pi_0^M$ to be exactly WMSOL. Next, for each $i \in \mathbb{N}$ the level Σ_{i+1}^M is the set of formulas $\exists X_1 \dots . \exists X_n . \varphi$ with $\varphi \in \Pi_i^M$. Similarly Π_{i+1}^M consists of formulas $\forall X_1 \dots . \forall X_n . \varphi$ with $\varphi \in \Sigma_i^M$.

Definition 41 A language L of words, trees or graphs is in Σ_n^M if there is a Σ_n^M formula defining this language. Otherwise it is Σ_n^M-*unfeasible*. Similarly for Π_n^M classes.

μ-calculus The definition of the μ-calculus hierarchy is easier to formulate for the vectorial μ-calculus. The formulas of levels Σ_0^μ and Π_0^μ are of the form $\vec{\alpha}$ for some vector of formulas without a fixpoint operator. Σ_{i+1}^μ formulas are of the form $\mu \vec{X}.\vec{\alpha}$ for some Π_i^μ formula $\vec{\alpha}$. Π_{i+1}^μ formulas are of the form $\nu \vec{X}.\vec{\alpha}$ for some Σ_i^μ formula $\vec{\alpha}$.

Equivalently the definition of the hierarchy for the μ-calculus can be formulated using the scalar syntax. Then $\Sigma_0^\mu = \Pi_0^\mu$ is the set of formulas without fixpoints. Σ_{i+1}^μ is the closure of Π_0^μ under conjunction, disjunction, substitutions and application of the least fixpoint operator μ (i.e. $\mu X.\alpha \in \Sigma_{i+1}^\mu$ if $\alpha \in \Sigma_{i+1}^\mu$). Similarly for Π_{i+1}^μ but now the class is closed under applications of the greatest fixpoint operator.

An *alternation free* fragment of the μ-calculus is the closure of $\Sigma_1^\mu \cup \Pi_1^\mu$ under Boolean operations and substitutions.
Remark: We use the term alternation here with a different meaning than in the case of alternating automata.

Definition 42 A language L of words, trees or graphs is in Σ_n^μ if there is a Σ_n^μ formula defining this language (alternatively there is a Σ_n^μ vectorial formula α such that $(\alpha)\downarrow_1$ defines L). Otherwise it is Σ_n^μ-*unfeasible*. Similarly for Π_n^μ classes.

Automata Finally, we define the hierarchy for automata. The hierarchy is based on the size of acceptance conditions. It is the easiest to define it for Mostowski acceptance conditions although it was original formulated for Rabin conditions (cf. [41])

Automata of level Σ_n^a are such that the range of the acceptance function Ω is in the set $\{1, \ldots, n\}$. Automata of level Π_n^a have the range of the acceptance function in $\{0, \ldots, n-1\}$. In the case of $n = 1$ automata from Σ_1^a would accept nothing. To remedy this we assume that automata have a special state $\top$ from which they accept every tree. Such a state is definable in all but Σ_1^a automata.

Remark: The range of the function Ω defining a Mostowski acceptance condition can be always scaled down, so that the smallest number in the range is 0 or 1. Just observe that subtracting 2 from every value of Ω does not change the semantics of the automaton (provided we do not get negative values). Similarly we can cut out any gaps in the range of Ω. So we can always make the image of Ω to be an interval starting from 0 or 1.

A *weak automaton* is an automaton with weak acceptance conditions [38, 33]. These are like Büchi, Rabin, etc. conditions but on the set of all the states appearing in the run and not only on the set of states appearing infinitely often. So for example a weak Büchi condition $F \subseteq Q$ defines a set of runs going through some state from F at least once.

Fact 43 Every weak alternating automaton is a Σ_1^a and a Π_1^a automaton.

The above definitions did not depend on whether an automaton in question is deterministic, nondeterministic or alternating. Hence we have defined not one but actually three hierarchies.

Definition 44 A language L of words, trees or graphs is in deterministic, nondeterministic or alternating Σ_n^a if there is a Σ_n^a deterministic, nondeter-

ministic or alternating automaton recognizing this language. Otherwise it is Σ_n^a-*unfeasible*. Similarly for Π_n^a classes.

6.2 Connecting fixpoint alternation and index hierarchies

Looking closer at the translations between the μ-calculus and alternating automata from Sections 5.3 and 5.4 we can see that they preserve the levels of the hierarchy.

Theorem 45
For every n, $\Sigma_n^\mu = \Sigma_n^a$ *and* $\Pi_n^\mu = \Pi_n^a$.

The hierarchy of nondeterministic tree automata is different. There are Σ_2^a languages that are arbitrary high in the hierarchy of nondeterministic automata. The hierarchy of nondeterministic automata has a corresponding fixpoint hierarchy for formulas with a very limited use of conjunction. These relations are in depth discussed in [41].

We also cannot expect that the MSOL hierarchy coincides with the fixpoint alternation hierarchy. It should be clear that existence of a run of an automaton can be expressed by a formula quite low in the monadic hierarchy. So over words or trees the MSOL hierarchy collapses. The level on which it collapses depends on the model. We will examine it in more detail below.

6.3 The case of words

In the case of infinite words most of the hierarchies collapse on the first level. This is mainly due to the fact that there are deterministic devices capturing the power of MSOL on infinite words.

Fact 46 WMSOL=MSOL over infinite words.

Proof
Let φ be a MSOL formula and let $\mathcal{A}_\varphi$ be a deterministic parity automaton accepting exactly the models of φ. We write a WMSOL formula expressing the fact that there is an accepting run of $\mathcal{A}_\varphi$.

Let $w \in \Sigma^\omega$ be some word. Because $\mathcal{A}_\varphi$ is deterministic, if $\mathcal{A}_\varphi$ has arbitrary long finite prefixes of runs on w then it has a unique infinite run on w. Hence, to state an existence of a run of $\mathcal{A}_\varphi$ it is enough to say that for every position there is a run up to this position.

A run is accepting iff there is some even priority p which appears infinitely often on the run and every smaller priority appears only finitely often on the

run. This property can be expressed by saying that there is some position after which on the run there is no state of priority smaller than p and for every position there is a later position with a state of priority p.

All these facts can be formulated in WMSOL because they refer only to finite prefixes of the model. □

Fact 47 The hierarchy of nondeterministic word automata collapses on the level Π_1^a. Every word automaton is equivalent to a weak alternating automaton.

Proof
The first statement is just rephrasing of the fact that every automaton over words is equivalent to a Büchi nondeterministic automaton.

The second fact is the rephrasing of the above considerations for WMSOL. Let $\mathcal{A}$ be a deterministic parity automaton. We want to find an equivalent weak automaton $\mathcal{B}$. First part of $\mathcal{B}$ just simulates $\mathcal{A}$. All the states in this part have priority 1, so $\mathcal{B}$ cannot stay in this part if it is going to accept a word. It can go to another part when it decides that from that moment no priority smaller than p is going to appear and p is going to appear infinitely often. To check this $\mathcal{B}$ simulates $\mathcal{A}$ over states of priority $\geq p$. It also uses universal branching to check that after every position a state of priority p eventually occurs. □

Corollary 48 Every μ-calculus sentence is equivalent over words to a sentence from the alternation-free fragment.

The picture changes if we restrict ourselves to deterministic automata. We present here the analysis from [42]. Later we will see that very similar examples show up in the case of trees.

For deterministic automata we have a proper hierarchy. To see the examples consider for each $n \in \mathbb{N}$ an alphabet $\Sigma_n = \{0, \ldots, n\}$. Then we define the languages:

$$M_n = \{w \in \Sigma_n^* : \liminf_{n\to\infty} w(n) \text{ is even}\}$$
$$N_n = \{w \in \Sigma_n^* : \liminf_{n\to\infty} w(n) \text{ is odd}\}$$

So, M_n consists of words where the smallest number appearing infinitely often is even, and for words in N_n this number is odd.

It is easy to see that M_n can be recognized by a Σ_n^a deterministic automaton and N_n can be recognized by a Π_n^a deterministic automaton. The proof that there are no simpler automata follows from a more general lemma

presented below. It shows a connection between the Mostowski index of an ω-word language and the shape of a deterministic Mostowski automaton recognizing the language. Roughly speaking, it says that in the graph of an automaton recognizing a "hard" language there must be a subgraph, called a flower, "witnessing" this hardness.

Definition 49 Let $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \Omega \rangle$ be a deterministic Mostowski automaton on words. The *graph of* $\mathcal{A}$ is the graph obtained by taking Q as the set of vertices and adding an edge from q to q' whenever $\langle q, a, q' \rangle \in \delta$, for some letter a.

A *path* in a graph is a sequence of vertices $v_1, \ldots, v_j$, such that, for every $i = 1, \ldots, j-1$ there is an edge from v_i to v_{i+1} in the graph. A *maximal strongly connected component* of a graph is a maximal subset of vertices of the graph, such that, for every two vertices v_1, v_2 in the subset there is a path from v_1 to v_2 and from v_2 to v_1.

For an integer k, a *k-loop* in $\mathcal{A}$ is a path $v_1, \ldots, v_j$ in the graph of $\mathcal{A}$ with $v_1 = v_j$, $j > 1$ and $k = \min\{\Omega(v_i) : i = 1, \ldots, j\}$. Observe that a k-loop must necessarily go through at least one edge.

Given integers m and n, a state $q \in Q$ is a *m-n-flower* in $\mathcal{A}$ if for every $k \in \{m, \ldots, n\}$ there is, in the graph of $\mathcal{A}$, a k-loop containing q.

Definition 50 We say that a language $L \subseteq \Sigma^\omega$ *admits* an m-n-flower if there exists a deterministic Mostowski automaton $\mathcal{A}$, such that, $L = L(\mathcal{A})$ and $\mathcal{A}$ has an m-n-flower q for some q not a useless state in $\mathcal{A}$ (i.e. q occurring in some accepting run of $\mathcal{A}$).

Lemma 51 (Flower Lemma) For every $n \in \mathbb{N}$ and $L \subseteq \Sigma^\omega$: (1) if L is Σ^a_{n+1}-unfeasible then L admits a $2i$-$(2i+n)$-flower, for some i; (2) if L is Π^a_n-unfeasible then L admits a $(2i+1)$-$(2i+1+n)$-flower, for some i.

Corollary 52 The problem of establishing the index of the language accepted by an automaton $\mathcal{A}$ with a Mostowski condition can be solved in time $\mathcal{O}(|\mathcal{A}|^2)$.

6.4 The case of trees

The case of trees is even more interesting than the case of words. The fixpoint alternation and automata hierarchies are infinite over binary trees. The MSOL hierarchy collapses on Σ^M_2 level. But even here we have an intriguing correspondence between lower levels of the MSOL hierarchy and the fixpoint alternation hierarchy.

The fixpoint alternation hierarchy in infinite

The main result of this section is the proof of strictness of the fixpoint alternation hierarchy. The strictness of the hierarchy over graphs was proved by Bradfield [10]. For binary trees the result was independently shown by Bradfield [11] and Arnold [5]. We present here the beautiful proof of Arnold.

The example languages showing the strictness of the hierarchy reflect closely the acceptance games of alternating automata. For a number $n \in \mathbb{N}$ consider the alphabet $\Sigma_n = \{c_i, d_i : i = 1, \ldots, n\}$. A tree over Σ_n represents a game. In vertices labelled by d_i player 0 chooses a successor, in vertices labelled by c_i player 1 makes a choice. The result of a play in such a game is a path in the tree. We look at the subscripts of the letters c and d and consider those subscripts which appear infinitely often on the path. Player 0 wins if the minimal subscript is even. Hence, a tree over Σ_n defines a game with Mostowski winning conditions given by subscripts of the letters from the alphabet.

With the interpretation of trees over Σ_n as games we can define two families of languages:

$$\begin{aligned} M_n &= \{t \in \mathrm{Trees}(\Sigma_n) : \text{player 0 has a winning strategy on } t\} \\ N_n &= \{t \in \mathrm{Trees}(\Sigma_n) : \text{player 1 has a winning strategy on } t\} \end{aligned}$$

The following easy lemma gives an upper bound on the complexity of these languages.

Lemma 53 Languages M_n and N_n can be recognized by a nondeterministic Σ^a_n and Π^a_n automata respectively.

The theorem we want to prove now says that there are no simpler automata, even alternating ones, recognizing these languages.

Theorem 54 (Bradfield, Arnold)
Language N_n cannot be recognized by an alternating Σ^a_n automaton. Similarly, M_n cannot be recognized by an alternating Π^a_n automaton.

For the proof we need a way of coding runs of automata as game trees. Let $\mathcal{A} = \langle Q, Q_\exists, Q_\forall, \Sigma_n, q^0, \delta, Acc\rangle$ be an alternating automaton. We can assume without a loss of generality that for every $q \in Q$ and $a \in \Sigma$ the set $\delta(q, a)$ has precisely two elements. If not then we can split bigger sets and add some auxiliary states connecting them. Sets having zero or one element can be extended with dummy states.

Acceptance of a tree t by $\mathcal{A}$ was defined using the game $G_{\mathcal{A},t}$. By our assumption on the values of δ, every vertex of this game has a degree 2. Still

$G_{\mathcal{A},t}$ is usually not a tree. What we are after is an unwinding of this game into a tree such that each node is labelled with c_i or d_i depending on the player this node belongs to and the priority of the node. Formally, we define a function $run_{\mathcal{A}} : Q \times \mathrm{Trees}(\Sigma_n) \to \mathrm{Trees}(\Sigma_n)$ to be the unique function satisfying that for every $q \in Q$, whenever $\delta(q, t(a)) = \{(q_0, d_0), (q_1, d_1)\}$ then:

$$run_{\mathcal{A}}(q,t) = \begin{cases} c_i(run_{\mathcal{A}}(q_0, t|_{d_0}), run_{\mathcal{A}}(q_1, t|_{d_1})) & \text{for } q \in Q_\forall \\ d_i(run_{\mathcal{A}}(q_0, t|_{d_0}), run_{\mathcal{A}}(q_1, t|_{d_1})) & \text{for } q \in Q_\exists \end{cases}$$

We write $c_i(t_0, t_1)$ to denote the tree with root labelled c_i and the trees t_0, t_1 as the left and right subtrees respectively. We use $t|_d$ to denote the subtree rooted in the vertex d. So, $t|_0$ denotes the left subtree of the root and $t|_\varepsilon = t$. In the above definition a small catch is that the value of the function δ is an unordered pair and the definition of $run_{\mathcal{A}}$ depends on the order in this pair. We assume that we have some unambiguous way of ordering such pairs.

The function we are interested in is $run^0_{\mathcal{A}} : \mathrm{Trees}(\Sigma_n) \to \mathrm{Trees}(\Sigma_n)$ defined by $run^0_{\mathcal{A}}(t) = run_{\mathcal{A}}(q^0, t)$ where q^0 is the initial state of $\mathcal{A}$. The following lemma is just a reformulation of the definition of acceptance.

Lemma 55 For every tree $t \in \mathrm{Trees}(\Sigma_n)$, $t \in L(\mathcal{A})$ iff $run^0_{\mathcal{A}}(t) \in M_n$.

The proof of the theorem uses a tree which is a fixpoint of the function $run^0_{\mathcal{A}}$. The existence of such a fixpoint is guaranteed by Banach theorem which we now recall. We begin by with the usual ultrametric distance on $\mathrm{Trees}(\Sigma_n)$.

Definition 56 The distance between two trees $t, t' \in \mathrm{Trees}(\Sigma_n)$ is defined inductively as follows. If $t = t'$ then $dist(t, t') = 0$. If the labels of the roots of t and t' are different then $dist(t, t') = 1$, otherwise,

$$dist(t, t') = 1/2 \max(dist(t|_0, t'_0), dist(t_1, t'_1))$$

A mapping $f : \mathrm{Trees}(\Sigma_n) \to \mathrm{Trees}(\Sigma_n)$ is called *contracting* if there is a constant $c < 1$ such that for every $t, t' \in \mathrm{Trees}(\Sigma_n)$ we have:

$$dist(f(t), f(t')) \leq c\, dist(t, t')$$

Lemma 57 The mapping $run^0_{\mathcal{A}} : \mathrm{Trees}(\Sigma_n) \to \mathrm{Trees}(\Sigma_n)$ is contracting.

A metric space is *complete* if every Cauchy sequence of elements of the space has a limit. It is not difficult to check that Trees_Σ with *dist* metric is compact. By Banach theorem the function $run^0_{\mathcal{A}}$ has a unique fixpoint. We use this fixpoint to prove the theorem.

Proof (of Theorem 54)
For the first statement, suppose that N_n is recognized by an alternating Σ_n^a automaton $\mathcal{A}$. Take the fixpoint t of the $run_{\mathcal{A}}^0$ mapping. We have:

$$t \in N_n \quad \text{iff} \quad t \in L(\mathcal{A}) \quad \text{iff} \quad run_{\mathcal{A}}^0(t) \in M_n \quad \text{iff} \quad t \in M_n$$

The second equivalence follows from Lemma 55. The third holds just because $run_{\mathcal{A}}^0(t) = t$.

For the second statement, suppose that M_n is recognized by an alternating Π_n^a automaton $\mathcal{A}$. Language N_n is the complement of M_n. Hence, it is recognized by $\overline{\mathcal{A}}$, the dual of $\mathcal{A}$ (see 26). But, $\overline{\mathcal{A}}$ is a Σ_n^a automaton. Contradiction with the previous paragraph. □

The languages M_n, N_n showing strictness of the hierarchy use alphabets depending on n. It is possible to show the strictness for the languages over a two element alphabet. Big alphabets can be coded by sequences over two letters. Then, one can show that if the coding of N_n were accepted by a Σ_n^a alternating automaton then N_n also would be accepted by an automaton of this kind.

Relation to the MSOL hierarchy

The MSOL hierarchy over binary trees also collapses but on a higher level than over words.

Lemma 58 Every MSOL formula over trees is equivalent to a Σ_2^M formula.

Proof
It is enough to show that for every nondeterministic tree automaton $\mathcal{A}$ there is a Σ_2^M formula $\varphi_{\mathcal{A}}$ expressing the fact that $\mathcal{A}$ has an accepting run. The formula is of the form

$$\exists X_1 \ldots X_n . \forall P.\ Run(X_1, \ldots, X_n) \wedge (Path(P) \Rightarrow Accepting(X_1, \ldots, X_n, P))$$

So we use existential quantification to guess a run and then universal quantification to quantify over paths of the tree. The facts that $X_1, \ldots, X_n$ define a run and that that the run is accepting on a path P can be expressed in first-order logic. □

There is a nice correspondence between lower levels of the MSOL hierarchy and the fixpoint alternation hierarchy.

Theorem 59
Σ_1^M properties are exactly Π_2^μ properties. $\Sigma_0^M = \Pi_0^M$ properties are exactly the properties expressible in the alternation free fragment of the μ-calculus. Moreover, $\Sigma_0^M = \Sigma_1^M \cap \Pi_1^M$.

Recall that Π_2^μ tree languages are precisely the languages definable by Büchi tree automata. By Fact 17 we know that Σ_1^M is strictly smaller than Σ_2^M. As the set of Σ_2^M tree languages is closed under complement, also Π_1^M is strictly smaller that Σ_2^M. For the similar reasons $\Sigma_0^M = \Pi_0^M$ is strictly included in Σ_1^M and Π_1^M.

6.5 The case of graphs

The hierarchies over graphs are also very interesting. There are still many open questions in this setting. The first important difference is that finiteness is not definable is MSOL over graphs. So the standard definition of the hierarchy changes. Now the $\Sigma_0^M = \Pi_0^M$ level consists of first-order formulas. The rest of the monadic hierarchy is defined the same way as before. This is the way we will understand the monadic hierarchy in this section.

The important difference with the previous cases is that here the monadic hierarchy is infinite [34] over graphs.

Theorem 60 (Matz, Schweikardt, Thomas)
The MSOL hierarchy over finite graphs is strict.

The strictness of the fixpoint alternation hierarchy over binary trees implies the strictness of this hierarchy over (finite) graphs. A natural question to ask is what are the relations between the hierarchies. It turns out that we cannot hope for a complete correspondence [29].

Fact 61 There are bisimulation invariant graph properties that are arbitrary high in the fixpoint alternation hierarchy but on Σ_2^M level of the monadic hierarchy.

It is an interesting open question whether the monadic hierarchy is strict for bisimulation closed properties. In other words, whether one can translate the μ-calculus into some fixed level of the monadic hierarchy.

A whole new spectrum of problems opens when one considers a modification of the monadic hierarchy called *closed monadic hierarchy*. Roughly, in the closed monadic hierarchy we can use first-order quantifiers for free. See [1] for the introduction to this hierarchy. In [6] the closed monadic hierarchy over trees is discussed.

7 Guarded logic

The goal of this section is to extend the results on MSOL and the μ-calculus to a more general relational setting. Consider a translation of modal logic

(i.e. the μ-calculus without fixpoints) to first-order logic:

$$\begin{aligned} Z &\rightsquigarrow Z(x) \\ P_a &\rightsquigarrow P_a(x) \\ \langle\cdot\rangle\alpha &\rightsquigarrow \exists y.\, E(x,y) \wedge \varphi_\alpha(y) \\ \alpha \vee \beta &\rightsquigarrow \varphi_\alpha(x) \vee \varphi_\beta(x) \end{aligned}$$

The translation gives for a modal formula α a formula $\varphi_\alpha(x)$ with one free variable x, s.t, for every labelled graph $\mathcal{M} = \langle V, E, (P_a)_{a\in\Sigma}\rangle$ we have $\| \alpha \|^M_V = \{s : M, V \vDash \varphi_\alpha(s)\}$

The set of formulas obtained from the translation is called the *modal fragment*. These formulas have several special properties. They use only monadic relations except for the edge relation. They use only two variables. The quantification pattern is very specific. First-order logic is undecidable, but the modal fragment is decidable in exponential time (because the μ-calculus is). The question we ask is "what makes the modal fragment so special?"

Here we show that it is the quantification patterns that are important. The idea of having the same quantification pattern as in the modal fragment is captured by the definition below. The main extension is that we put no restrictions on arity of relations. In the definition we use bold letters for vectors of variables.

Definition 62 The *guarded fragment* GF [4] of first-order logic is defined inductively as follows:

1. Every relational atomic formula belongs to GF.
2. GF is closed under propositional connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$.
3. If $\boldsymbol{x}, \boldsymbol{y}$ are tuples of variables, $\alpha(\boldsymbol{x}, \boldsymbol{y})$ is a positive atomic formula and $\psi(\boldsymbol{x}, \boldsymbol{y})$ is a formula in GF such that $\text{free}(\psi) \subseteq \text{free}(\alpha) = \boldsymbol{x} \cup \boldsymbol{y}$, then the formulas

$$\begin{aligned} \exists \boldsymbol{y}(\alpha(\boldsymbol{x}, \boldsymbol{y}) \wedge \psi(\boldsymbol{x}, \boldsymbol{y})) \\ \forall \boldsymbol{y}(\alpha(\boldsymbol{x}, \boldsymbol{y}) \rightarrow \psi(\boldsymbol{x}, \boldsymbol{y})) \end{aligned}$$

   belong to GF.

Here $\text{free}(\psi)$ denotes the set of free variables of ψ. An atom $\alpha(\boldsymbol{x}, \boldsymbol{y})$ that relativizes a quantifier as in rule (3) is the *guard* of the quantifier. Notice that the guard must contain *all* the free variables of the formula in the scope of the quantifier.

Note that first-order quantification over individual free variable is always admissible in GF, since singletons are guarded:

$$\exists x.\ \varphi(x) \ \equiv\ \exists x.\ x = x \wedge \varphi(x)$$

Clearly all modal formulas are translateable to GF. But there are other formulas in GF too. For example backwards modalities are expressible in GF:

$$\langle\ \rangle^{-1}\alpha \ \rightsquigarrow\ \exists y.\ R(y,x) \wedge \varphi_\alpha(y)$$

We can also have very strange modalities like

$$\langle\circlearrowleft\rangle\alpha \ \rightsquigarrow\ \exists y.\ R(x,y) \wedge R(y,x) \wedge \varphi_\alpha(y)$$

GF seems not to be able to express all of temporal logic over $(\mathbb{N}, \leq)$. Indeed, the straightforward translation of (ψ **until** ϕ) into first-order logic

$$\exists y(x \leq y \wedge \phi(y) \wedge \forall z((x \leq z \wedge z < y) \rightarrow \psi(z))$$

is not guarded in the sense of Definition 62. However, the quantifier $\forall z$ in this formula is guarded in a weaker sense, which lead van Benthem [8] to the following generalization of GF.

Definition 63 The *loosely guarded fragment* LGF is defined similarly to GF, but the quantifier-rule is relaxed as follows:

(3)' If $\psi(\boldsymbol{x}, \boldsymbol{y})$ is in LGF, and $\alpha(\boldsymbol{x}, \boldsymbol{y}) = \alpha_1 \wedge \cdots \wedge \alpha_m$ is a conjunction of atoms, then

$$\exists \boldsymbol{y}((\alpha_1 \wedge \cdots \wedge \alpha_m) \wedge \psi(\boldsymbol{x}, \boldsymbol{y}))$$
$$\forall \boldsymbol{y}((\alpha_1 \wedge \cdots \wedge \alpha_m) \rightarrow \psi(\boldsymbol{x}, \boldsymbol{y}))$$

belong to LGF, provided that $\mathrm{free}(\psi) \subseteq \mathrm{free}(\alpha) = \boldsymbol{x} \cup \boldsymbol{y}$ and for every quantified variable $y \in \boldsymbol{y}$ and every variable $z \in \boldsymbol{x} \cup \boldsymbol{y}$ there is at least one atom α_j that contains both y and z.

In the translation of (ψ **until** ϕ) described above, the quantifier $\forall z$ is loosely guarded by $(x \leq z \wedge z < y)$ since z coexists with both x and y in some conjunct of the guard. On the other side, the transitivity axiom

$$\forall xyz(Exy \wedge Eyz \rightarrow Exz)$$

is not in LGF. The conjunction $Exy \wedge Eyz$ is not a proper guard of $\forall xyz$ since x and z do not coexist in any conjunct. Indeed, adding transitivity statement to GF makes the fragment undecidable [21].

Notation. We will use the notation $(\exists \boldsymbol{y} \,.\, \alpha)$ and $(\forall \boldsymbol{y} \,.\, \alpha)$ for relativized quantifiers, i.e., we write guarded formulas in the form $(\exists \boldsymbol{y} \,.\, \alpha)\psi(\boldsymbol{x}, \boldsymbol{y})$ and $(\forall \boldsymbol{y} \,.\, \alpha)\psi(\boldsymbol{x}, \boldsymbol{y})$. When this notation is used, then it is always understood that α is indeed a proper guard as specified by condition (3) or (3)'.

The following theorem says that LGF is not much more difficult than modal logic. The theorem refers to the width of a formula. This is a maximal number of free variables in any subformula of the formula.

Theorem 64 (Grädel [21])
The satisfiability problem for LGF is 2EXPTIME*-complete. It is* EXPTIME*-complete for formulas of bounded width.*

The reason for this doubly exponential complexity is just the fact that the formulas have unbounded width. Given that even a single predicate of arity n over a domain of just two elements leads to 2^{2^n} possible types already on the atomic level, the double exponential lower complexity bound is hardly a surprise. When the width is bounded the complexity of LGF is just slightly bigger than that of the modal logic (which is PSPACE-complete).

The next step is to add fixpoints to LGF without loosing decidability. We follow [23].

Definition 65 The guarded fixpoint logics μGF and μLGF are obtained by adding to GF and LGF, respectively, the following rules for constructing fixed-point formulas:

> Let W be a k-ary relation variable and let $\boldsymbol{x}$ be a k-tuple of distinct variables. Further, let $\psi(W, \boldsymbol{x})$ be a guarded formula where W appears only positively and not in guards. Moreover we require that all the free variables of $\psi(W, \boldsymbol{x})$ are contained in $\boldsymbol{x}$. For such a formula $\psi(W, \boldsymbol{x})$ we can build a formula
>
> $$[\mathsf{LFP}\, W\boldsymbol{x} \,.\, \psi](\boldsymbol{x})$$

The part in square brackets, i.e. $[\mathsf{LFP}\, W\boldsymbol{x} \,.\, \psi]$ is called *fixed point predicate.*

The semantics of fixpoint formulas is the usual one: Given a structure $\mathcal{M}$ and a valuation V for the free second-order variables in ψ, other than W, the formula $\psi(W, \boldsymbol{x})$ defines an operator on k-ary relations $W \subseteq M^k$, namely

$$\psi^{\mathcal{M},V}(W) := \{\boldsymbol{a} \in M^k : \mathcal{M}, V \models \psi(W, \boldsymbol{a})\}.$$

Since W occurs only positively in ψ, this operator is monotone (i.e., $W \subseteq W'$ implies $\psi^{\mathcal{M},V}(W) \subseteq \psi^{\mathcal{M},V}(W')$) and therefore has a least fixed

point $\mathrm{LFP}(\psi^{\mathcal{M},V})$. Now, the semantics of least fixed point formulas is defined by

$$\mathcal{M}, V \models [\mathsf{LFP}\, W\boldsymbol{x}\,.\,\psi(W,\boldsymbol{x})](\boldsymbol{a}) \quad \text{iff} \quad \boldsymbol{a} \in \mathrm{LFP}(\psi^{\mathcal{M},V})$$

Similarly as in the μ-calculus the greatest fixpoint is definable by:

$$[\mathsf{GFP}\, W\boldsymbol{x}\,.\,\psi(W,\boldsymbol{x})](\boldsymbol{a}) \;\equiv\; \neg[\mathsf{LFP}\, W\boldsymbol{x}\,.\,\neg\psi(\neg W,\boldsymbol{x})](\boldsymbol{a})$$

Observe that we do not allow to use fixed point predicates in guards. Otherwise guarded quantification would be as powerful as unrestricted quantification. Indeed, for every k, we can define the universally true k-ary relation by the fixed point predicate $[\mathsf{GFP}\, U^k x_1 \cdots x_k\,.\,\mathit{true}]$ (where *true* stands for any tautology). Using these predicates as guards one could obtain unrestricted quantification. Also the use of the fixed point variable W as a guard inside the formula defining it as a least or greatest fixed point, or the use of additional first-order variables as parameters in fixed point formulas would lead to an undecidable logic.

Despite all of these restrictions on constructing fixpoints, we can still translate the μ-calculus to GF. We extend the translation of modal logic to GF given at the beginning of the section:

$$\mu Z.\alpha(Z) \;\rightsquigarrow\; [\mathsf{LFP}\, Z(y).\,\varphi_\alpha(y)](x)$$

Contrary to both GF and the μ-calculus, guarded fixed-point logic does not have the finite model property. An infinity axiom is a satisfiable sentence that does not have a finite model.

Proposition 66 Guarded least fixpoint logic (even with only two variables, without nested fixed points and without equality) contains infinity axioms.

Proof
Consider the formulas

$$\exists xy.\, Fxy$$
$$(\forall xy\,.\,Fxy)\exists xFyx$$
$$(\forall xy\,.\,Fxy)[\mathsf{LFP}\, Wx\,.\,(\forall y\,.\,Fyx)Wy](x)$$

The first two formulas say that a model should contain an infinite F-path and the third formula says that F is well-founded, thus, in particular, acyclic. Therefore every model of these formulas is infinite. On the other side, the formulas are clearly satisfiable, for instance by $(\mathbb{N}, <)$. □

Even though we can express more than in the μ-calculus, the complexity of μLGF stays essentially the same.

Theorem 67
The satisfiability problem for μLGF is 2EXPTIME*-complete. It is* EXPTIME*-complete for formulas of bounded width.*

Note that this is the same complexity as for guarded first-order sentences, so we essentially do not pay any penalty for fixpoints. Fortunately, in most practical applications, formulas have only bounded width. In particular, for a fixed finite vocabulary all guarded formulas have bounded width. For example, the translation of the μ-calculus into μGF uses at most binary relations and leads to formulas of width two.

Knowing the complexity of guarded fragments it would be nice to understand the expressive power of the logic. Theorem 40 characterizes the expressive power of the μ-calculus by MSOL properties invariant under bisimulation. Of course we cannot directly compare μGF with MSOL as the signatures of the logics are different (μGF contains relations of higher arity). We rather define a fragment of second-order logic which we call guarded fragment or GSO for short. Then we define guarded bisimulation which will relate tuples of elements of two structures and not just single elements as bisimulation did. Finally, we show that GSO sentences that are guarded bisimulation invariant are exactly μGF sentences. The results sumarized below come from [22].

Definition 68 Let $\mathcal{M}$ be a structure over a signature Sig and with the universe M. A tuple $(m_1, \ldots, m_k)$ is *guarded* iff there is a relation R and elements $m'_1, \ldots, m'_l$ such that $R(m'_1, \ldots, m'_l)$ holds in $\mathcal{M}$ and $\{m_1, \ldots, m_k\} \subseteq \{m'_1, \ldots, m'_l\}$. A relation $S \subseteq M^n$ is guarded if it consists of guarded tuples.

Definition 69 *Guarded second-order logic* (GSO), is an extension of first-order logic with second-order quantifiers ranging over guarded relations.

Lemma 70 SO is strictly more expressive than GSO. In particular GSO collapses to MSOL over words in case words are represented as structures with a successor relation instead of linear ordering.

In order to define guarded bisimulation it will be useful to have a notion of *partial isomorphism*. The bisimulation relation relates single elements. These elements are required to have the same labeling. Now we want to relate tuples of elements and we want to say that the tuples satisfy the same relations. This is precisely what partial isomorphism is saying. Formally, a partial isomorphism between structures $\mathcal{M}_1$ and $\mathcal{M}_2$ is a bijective function $f : X \to Y$ for some $X \subseteq \mathcal{M}_1$ and $Y \subseteq \mathcal{M}_2$. It must satisfy the condition that for every relation symbol R and a tuple of elements $a_1, \ldots, a_k \in X$: relation $R^{\mathcal{M}_1}(a_1, \ldots, a_k)$ holds iff $R^{\mathcal{M}_2}(f(a_1), \ldots, f(a_k))$ holds.

Definition 71 *Guarded bisimulation* between two structures $\mathcal{M}_1$, $\mathcal{M}_2$ of signature Sig is a non-empty set I of partial isomorphisms from $\mathcal{M}_1$ to $\mathcal{M}_2$ such that for every $f : X \to Y$ in I the following conditions hold:

- for every guarded set $X' \subseteq \mathcal{M}_1$ there is a partial isomorphism $g : X' \to Y'$ such that f and g agree on $X \cap X'$.
- for every guarded set $Y' \subseteq \mathcal{M}_2$ there is a partial isomorphism $g : X' \to Y'$ such that f^{-1} and g^{-1} agree on $Y \cap Y'$.

Two tuples of elements $(a_1, \ldots, a_n) \in \mathcal{M}_1$ and $(b_1, \ldots, b_n) \in \mathcal{M}_2$ are *guarded bisimilar* if there is $f \in I$ mapping a_i to b_i for all $i = 1, \ldots, n$.

Definition 72 A formula $\varphi(x_1, \ldots, x_n)$ is *invariant under bisimulation* if it cannot distinguish between guarded bisimilar tuples, i.e., if $\mathcal{M}_1 \models \varphi(a_1, \ldots, a_n)$ and $(a_1, \ldots, a_n)$ is guarded bisimilar to a tuple $(b_1, \ldots, b_n) \in \mathcal{M}_2$ then $\mathcal{M}_2 \models \varphi(b_1, \ldots, b_n)$.

The following theorem from [22] ties together the expressive power of GSO and μGF.

Theorem 73 (Grädel, Hirsch, Otto)
Every formula of GSO invariant under guarded bisimulation is equivalent to a μGF formula.

8 Traces

Infinite words, which are linear orders on *events*, are often used to model executions of systems. Infinite *traces*, which are partial orders on events, can be used to model concurrent systems when we do not want to put some arbitrary ordering on actions occurring concurrently. The idea is that if we have two actions, say a and b, occurring concurrently then we do not want to model this neither as a word ab nor as ba. A more faithful representation of what happened is a partial order with two events a, b and no ordering between them.

A *trace alphabet* is a pair (Σ, D) where Σ is a finite set of *actions* (i.e. letters) and $D \subseteq \Sigma \times \Sigma$ is a reflexive and symmetric *dependence relation*. Intuitively if $(a, b) \in D$ then a and b share some resource, so their occurrences should be ordered. On the other hand, if $(a, b) \notin D$ then there is no reason to order occurrences of these actions.

A *trace* or *dependence graph* is a labelled graph

$$G = \langle E, R \subseteq E \times E, \lambda : E \to \Sigma \rangle$$

such that R is a partial order and the following conditions are satisfied:

(T1) $\forall e \in E.$ $\quad \{e' : R(e', e)\}$ is a finite set.
(T2) $\forall e, e' \in E.$ $\quad (\lambda(e), \lambda(e')) \in D \Rightarrow R(e, e') \vee R(e', e).$
(T3) $\forall e, e' \in E.$ $\quad R(e, e') \Rightarrow (\lambda(e), \lambda(e')) \in D \vee$
$\exists e''.\ R(e, e'') \wedge R(e'', e') \wedge e \neq e'' \neq e'.$

The nodes of a dependence graph are called *events*. An *a-event* is an event $e \in E$ which is labelled by a, i.e., $\lambda(e) = a$. We say that e is *before* e' iff $R(e, e')$ holds. In this case we also say that e' is *after* e.

The first condition of the definition of dependence graphs says that the past of each event (the set of the events before the event) is finite. The second one postulates that events labelled by dependent letters are ordered. The third, says that the order is induced by the order between dependent letters.

Below we describe a variation on the representation of dependence graphs. This variation will be important when defining the μ-calculus over traces.

Definition 74 A *Hasse diagram* of a trace $G = \langle E, R, \lambda \rangle$ is a labelled graph $\langle E, R_H, \lambda \rangle$ where R_H is the smallest relation needed to determine R, i.e., the reflexive and transitive closure of R_H is R, and if $R_H(e, e')$ holds then there is no e'' different from e and e' such that $R_H(e, e'')$ and $R_H(e'', e')$ hold.

Büchi theorem tells us that for the class of finite or infinite words (dependence graphs for alphabets where all the letters are mutually dependent) the properties definable by MSOL are exactly the languages recognizable by automata. This characterization carries through to traces with an appropriate modification of the notion of automata.

MSOL logic over traces is just MSOL logic over dependence graphs considered as labelled graphs. Observe that a Hasse diagram of a trace is MSOL definable in a dependence graph. Also dependence graph is MSOL definable in a Hasse diagram of a trace. Hence, MSOL definability over dependence graphs and over Hasse diagrams are the same thing.

Another way of defining traces is to consider linearizations of traces. A *linearization of a trace* $\langle E, R, \lambda \rangle$ is an injective function $f : E \to \mathbb{N}$ such that if $R(e, e')$ holds then $f(e) \leq f(e')$. We can identify a linearization f with an ω-word $\lambda(f^{-1}(0))\lambda(f^{-1}(1))\ldots$. It is easy to see that an infinite word over Σ defines the unique trace of which it is a linearization. This defines a trace equivalence over ω-words: $w \sim w'$ if they define the same trace. A language L is *trace consistent* if whenever $w \in L$ and $w \sim w'$ then $w' \in L$. Hence, a way to define a trace language is to define the set of its linearizations, i.e., a trace consistent language.

There are also automata working directly on traces, although it will be easier here to describe their behaviour on linearizations of traces. From their definition it will be clear that they only accept trace consistent sets of ω-words.

Suppose we have some number k off *processes* and consider function $loc : \Sigma \to \mathcal{P}(\{1, \ldots, k\})$ assigning to each letter a set of processes. Intuitively these are the processes that are needed to read input a. The distribution of letters should reflect our dependence alphabet (Σ, D) in a sense that $(a, b) \in D$ iff $loc(a) \cap loc(b) \neq \emptyset$. Intuitively, the two letters are dependent if for reading them some common process is needed. An *asynchronous automaton* is a tuple:

$$\mathcal{A} = \langle \prod_{i=1}^{k} Q_i, \Sigma, q^0, (\delta_a)_{a \in \Sigma}, \mathcal{F} \rangle$$

satisfying the following conditions

- the global set space $Q = \prod_{i=1}^{k} Q_i$ is the product of local finite states spaces Q_i,
- $q^0 \in Q$,
- for each $a \in \Sigma$, relation $\delta_a \subseteq Q \times Q$ such that if

  $$((q_1, \ldots, q_k), a, (q'_1, \ldots, q'_k)) \in \delta_a$$

  and $i \notin loc(a)$ then $q_i = q'_i$ and for every $\widehat{q} \in Q_i$:

  $$((q_1, \ldots, q_{i-1}, \widehat{q}, \ldots, q_k), a, (q'_1, \ldots, \widehat{q}, \ldots, q'_k)) \in \delta_a$$

- $\mathcal{F} = \{(F_1^\omega, F_1), \ldots, (F_k^\omega, F_k)\} \subseteq \mathcal{P}(Q) \times \mathcal{P}(Q)$ defines the acceptance condition.

Hence the transition relation for a letter a is allowed to examine and change only the components of the automaton that are in $loc(a)$. A run of $\mathcal{A}$ on a ω-word $w \in \Sigma^\omega$ is defined as for ordinary finite automata over states Q. For a run $r : \mathbb{N} \to Q$ and $p \in \{1, \ldots, k\}$ we define $\mathrm{Inf}_p(r) = \mathrm{Inf}(r) \cap Q_p$ to be the set of sates form Q_p that appear infinitely often in the run. A run r is *accepting* if:

- $\mathrm{inf}_p(r) \cap F_p^\omega \neq \emptyset$ for every p such that a letter b with $p \in loc(b)$ appears infinitely often in w, and
- $\mathrm{inf}_p(r) \cap F_p \neq \emptyset$ for every other p.

So the acceptance condition is a Büchi condition but it can also tell whether a process p was active infinitely often or not.

A trace is *accepted* by $\mathcal{A}$ if one of its linearizations is accepted by $\mathcal{A}$. The *language recognized by* $\mathcal{A}$ is the set of traces accepted by $\mathcal{A}$.

The following theorem summarizing many results on traces can be found in [16, 20, 40].

Theorem 75
Fix a trace alphabet. For a set L of traces the following are equivalent:

- *L is definable by a MSOL formula.*
- *L is recognizable by an asynchronous automaton.*
- *The set of linearizations of traces in L is a recognizable language of infinite words.*

In the case of words we had also a characterization of regular languages by the μ-calculus. As traces are just labelled graphs we can evaluate the μ-calculus directly on them, but we have a small problem when we want to make it precise what it means that a set of traces is definable by a μ-calculus formula. In the case of words we said that these are the words where the formula holds on the first position. In the case of traces we may have several minimal events. To overcome this problem we assume that in our traces we have always the least element $\bot$ labelled with a special letter also denoted by $\bot$. This letter is dependent on every other letter in Σ.

If G is a trace that has the least event $\bot$ and α is a μ-calculus sentence then we write $G \vDash \alpha$ to mean that $G, \bot \vDash \alpha$. Sentence α defines the set of traces $\{G : G \vDash \alpha\}$.

The μ-calculus over traces does not have sufficient expressive power. Let us see an example showing that there are even first-order definable properties of traces which are not expressible in the μ-calculus.

We claim that no μ-calculus sentence can distinguish between the following two Hasse diagrams of traces presented in Figure 2. In the left graph the dots stand for the sequence $(dc)^\omega$ and in the right graph for $(cd)^\omega$. In this example the trace alphabet $(\{\bot, a, b, c, d\}, D)$ where D is the smallest symmetric and reflexive relation containing the pairs $\{(a, c), (b, d), (c, d)\} \cup \{\bot\} \times \{a, b, c, d\}$. The two Hasse diagrams are bisimilar, hence indistinguishable by μ-calculus formulas. Still, a first order formula saying that the first d is before the first c is satisfied in the left graph but not in the right. The figure shows Hasse diagrams, but also dependence graphs of these two traces are bisimilar.

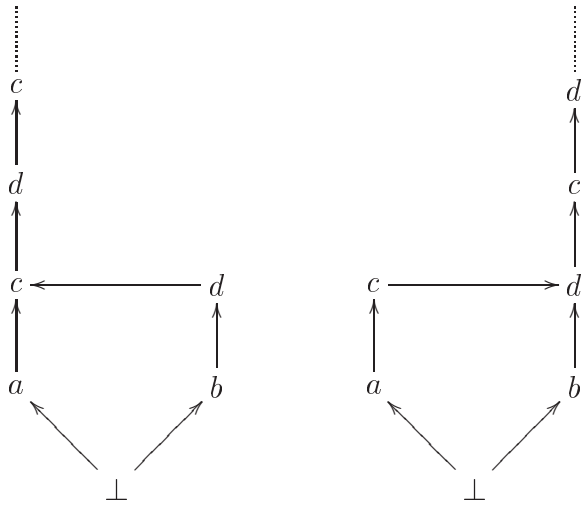

Figure 2: Indistinguishable traces

The above example indicates that some extension of the μ-calculus is needed. To see such an extension consider a *concurrency relation* in a trace G defined by $co(e, e')$ if neither $R(e, e')$ nor $R(e', e)$ holds. Then we can extend the μ-calculus with a proposition $co(a)$ for every $a \in \Sigma$. The semantics is

- $G, e \vDash co(a)$ if there is e' such that $co(e, e')$ and $\lambda(e') = a$.

Let μ^{co} be the extension of the μ-calculus with $co(a)$ propositions.

In the μ^{co}-calculus we can distinguish the two traces from Figure 2. The formula $\langle\cdot\rangle(P_a \wedge co(d))$ says that there is a successor of the least event, this successor is labelled by a and has a concurrent event labelled by d. The formula is true in the left graph but not in the right. The theorem below [54] says that this is not a coincident.

Theorem 76 (Expressive completeness)
The μ^{co} calculus is equivalent in expressive power to MSOL over traces.

Moreover the satisfiability problem for the μ^{co}-calculus is relatively easy.

Theorem 77
The satisfiability problem for the μ^{co}-calculus is PSPACE-complete.

9 Real-time

In real-time systems we have an interaction between continuous behaviour of some physical components and finite, discrete control. A standard example of

such a system is a steam boiler, that needs to keep water warm by switching a heater on and off. To describe behaviour of such a device it seems natural to model time flow by nonnegative real numbers. So, now we will be interested in properties of *signals* which are functions from reals to $\{0,1\}$. Equivalently a signal is a monadic predicate $P \subseteq \mathbb{R}^+$.

Intuitively, not all predicates can model a behaviour of a physical device. It is implausible to expect a device which is on when the time is a rational number and which is off when the time is an irrational number. To exclude such predicates one sometimes postulates *non-Zeno* assumption. A predicate is non-Zeno if on every bounded interval it changes the value only finitely many times. An equivalent convenient definition is the following:

Definition 78 A predicate $P \subseteq \mathbb{R}^+$ is *non-Zeno* if there is an unbounded sequence $0 = \tau_0 < \tau_1 < \tau_2 < \cdots$ of reals such that for every $i \in \mathbb{N}$: either $(\tau_i, \tau_{i+1}) \subseteq P$ or $(\tau_i, \tau_{i+1}) \cap P = \emptyset$. We write $\mathcal{P}_{NZ}(\mathbb{R}^+)$ for the set of non-Zeno predicates on $\mathbb{R}^+$.

In this section we will describe logics and automata for real-time properties. As we will see the situation here is much less satisfactory than in the cases we have discussed till now.

9.1 FOL and MSOL over reals

The signature of all the logics will be the same. It consists of a binary predicate symbol $\leq$ and unary predicate symbols $P_1, P_2, \ldots$ We will consider three classes of models

- $\mathcal{N}$ is the class of models of the form $\mathcal{M} = \langle \mathbb{N}, \mathcal{P}(\mathbb{N}), \leq, P_1^{\mathcal{M}}, \ldots \rangle$
- $\mathcal{R}$ is the class of models of the form $\mathcal{M} = \langle \mathbb{R}^+, \mathcal{P}(\mathbb{R}^+), \leq, P_1^{\mathcal{M}}, \ldots \rangle$
- $\mathcal{R}_{NZ}$ is the class of models of the form $\mathcal{M} = \langle \mathbb{R}^+, \mathcal{P}_{NZ}(\mathbb{R}^+), \leq, P_1^{\mathcal{M}}, \ldots \rangle$

The second element of the structure defines the range of second order variables and the interpretation of predicates. In particular in the case of $\mathcal{R}_{NZ}$ all predicates must be non-Zeno. In all three cases $\leq$ is interpreted as the standard relation on numbers.

The semantics of first-order and monadic second-order logics over these classes of structures is standard. In particular the second component of the structures plays no role in the semantics of first-order logic. In case of MSOL the range of second order variables is restricted to the elements of the second component of the structures. So, in case of $\mathcal{R}_{NZ}$ it means that we can quantify only over non-Zeno subsets of $\mathbb{R}^+$.

As we have seen in Section 3, MSOL over $\mathcal{N}$ is decidable and has characterizations in terms of automata and fixpoint calculi. Interestingly, the decidability of $\mathcal{R}_{NZ}$ can be reduced to the decidability of MSOL theory of the binary tree [44]. This gives

Theorem 79 (Rabin)
MSOL theory of $\mathcal{R}_{NZ}$ is decidable.

The picture for the class $\mathcal{R}$ is different (cf. [47, 13])

Theorem 80 (Shelah)
MSOL theory of $\mathcal{R}$ is undecidable. FOL theory of $\mathcal{R}$ is decidable.

Even in MSOL over reals we cannot express properties like "after no more than 5 units of time the heater switches off". To express such a property it seems to be a good idea to add $+1$ predicate to the classes of models considered above. So $\mathcal{N}^{+1}$, $\mathcal{R}^{+1}$ and $\mathcal{R}_{NZ}^{+1}$ stand for the classes of models extended with a binary predicate $+1(x, y)$ saying that $x + 1 = y$.

Unfortunately, it is not difficult to see that an unrestricted use of $+1$ predicate makes all the considered logics over reals undecidable.

Fact 81 FOL over $\mathcal{R}^{+1}$ or $\mathcal{R}_{NZ}^{+1}$ is undecidable.

We finish this subsection with a proposal of limiting the use of $+1$ predicate so that the decidability is regained. The logic L_1 [28] is an extension of FOL without $+1$ predicates by the following construction:

> If $\varphi(x)$ is a L_1 formula and x is the only free variable in $\varphi(x)$ then
>
> $$(\exists x)^{\leq x_0+1}_{>x_0}\ \varphi(x) \qquad (\exists x)^{\leq x_0}_{>x_0-1}\ \varphi(x)$$
>
> are formulas of L_1. The variable x_0 is the only free variable in these formulas

The semantics is as the syntax suggests:

> $\mathcal{M}, V \vDash (\exists x)^{\leq x_0+1}_{>x_0}\ \varphi(x)$ iff there is t such that $\mathcal{M}, V[t/x] \vDash \varphi(x)$ and $V(x_0) < t < V(x_0) + 1$.

We can also define ML_1 as an extension of L_1 with monadic quantification.

Theorem 82 (Hirshfeld, Rabinovich)
Both L_1 and ML_1 are decidable over $\mathcal{R}_{NZ}^{+1}$.

The complexity of FOL over $\mathcal{R}$, and hence also of L_1, is nonelementary. The bound follows, as usual, from the complexity of FOL over finite words. In [27] a decidable extension of L_1 is presented.

9.2 Real-time automata

Here we want to present an automata model for specifying real-time properties. Unfortunately the model is not closed under complement. We will also discuss some restrictions of this model.

Models over $\mathbb{N}$ are represented by ω-words. Models over $\mathbb{R}^+$ are represented by *timed words*. Unfortunately the representation is not as good as in the case of $\mathbb{N}$.

A *timed word* over an alphabet Σ is an infinite sequence $(a_0, \tau_0), (a_1, \tau_1), \ldots$ over $\Sigma \times \mathbb{R}^+$ such that , $\tau_i \leq \tau_{i+1}$ for every $i \in \mathbb{N}$. The idea is that the first components describe the events that occur and the second the times at which they occur.

There is no hope to have one-to-one correspondence between timed words and models from $\mathcal{R}$. We have a better chance with non-Zeno models from $\mathcal{R}_{NZ}$. There are several ways of coding a model $\mathcal{M} \in \mathcal{R}_{NZ}$ as a timed word. To have one-to-one correspondence between non-Zeno models and timed words it is necessary to put some restrictions on timed words. One of them is a progress requirement which says that for every $t \in \mathbb{R}^+$ there should be i with $\tau_i > t$. We will not discuss such a coding in detail because there is no standard coding and a coding is not important for the results on timed automata that we are going to present.

Let $\mathcal{Z}$ be a set of clocks (variables ranging over $\mathbb{R}^+$). Consider clock constraints given by the grammar:

$$CC(\mathcal{Z}) := x \leq c \mid c \leq x \mid c < x \mid x < c \mid CC(\mathcal{Z}) \wedge CC(\mathcal{Z})$$

where $x \in \mathcal{Z}$ is a clock and $c \in \mathbb{N}$ is a constant. A *clock interpretation* is a function $V : \mathcal{Z} \to \mathbb{R}^+$. The satisfaction relation $V \vDash \alpha$ for a clock constraint α is defined in a expected way. For a set of clocks $\mathcal{Y} \subseteq \mathcal{Z}$, let $V[\mathcal{Y} := 0]$ be the clock interpretation which is identical to V on clocks not in $\mathcal{Y}$ and equal to 0 on clocks in $\mathcal{Y}$. For every $t \in \mathbb{R}^+$, the interpretation $V + t$ gives the value $V(x) + t$ for every clock x.

A *timed automaton* is a tuple:

$$\mathcal{A} = \langle Q, \Sigma, \mathcal{Z}, q^0, \delta \subseteq Q \times \Sigma \times CC(\mathcal{Z}) \times \mathcal{P}(\mathcal{Z}) \times Q, Acc \subseteq Q^\omega \rangle$$

where Q is a finite set of states, Σ is a finite alphabet, $\mathcal{Z}$ is a finite set of clocks, q^0 is the initial state, and Acc is the set of accepting runs as in the case of ordinary ω-word automata. The transition relation is slightly complicated. Additionally to the usual components it has a clock constraint to be satisfied and a set of clocks to be reset.

A *configuration* of a timed automaton is a pair (q, V) consisting of a state of $\mathcal{A}$ and a valuation of the clocks. For a fixed automaton we define three relations on configurations:

- $(q, V) \xrightarrow{t} (q, V')$ for $t \in \mathbb{R}^+$ and $V' = V + t$,
- $(q, V) \xrightarrow{a} (q', V')$ if there is $(q, a, \alpha, \mathcal{Y}, q') \in \delta$ with $V \vDash \alpha$ and $V' = V[\mathcal{Y} := 0]$.
- $(q, V) \xrightarrow[t]{a} (q', V')$ if $(q, V) \xrightarrow{t} (q, V'')$ and $(q, V'') \xrightarrow{a} (q', V')$ for some V''.

A run of $\mathcal{A}$ on a timed word $(a_0, \tau_0), (a_1, \tau_1), \ldots$ is a sequence:

$$(q_0, V_0) \xrightarrow[t_0]{a_0} (q_1, V_1) \xrightarrow[t_1]{a_1} (q_2, V_2) \xrightarrow[t_2]{a_2} \cdots$$

such that: $q_0 = q^0$ is the initial state; V_0 is the initial valuation assigning 0 to every clock; $t_0 = \tau_0$ and $t_{i+1} = \tau_{i+1} - \tau_i$. A run is *accepting* if the sequence of states from the run is in Acc. The language recognized by $\mathcal{A}$ is the set of timed words accepted by $\mathcal{A}$.

Example: Consider a one letter alphabet $\Sigma = \{a\}$. Let L_1 be the language of timed words $(a, \tau_0), (a, \tau_1), \ldots$ such that $\tau_i + 1 = \tau_j$ for some i and j. So, we require that there are two occurrences of the letter with exactly one time unit difference. The language is recognized by the automaton on Figure 3. The initial state of the automaton is q_0. The acceptance condition Acc consists of all the sequences with infinitely many occurrences of q_3. $\square$

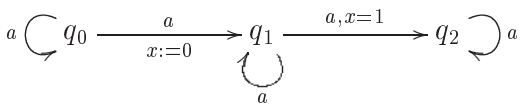

Figure 3: Timed automaton recognizing L_1.

The main result about timed automata is the decidability of the emptiness problem [2].

Theorem 83 (Alur, Dill)
The following problem is PSPACE complete: given $\mathcal{A}$ decide if $\mathcal{A}$ accepts some timed word.

Strangely enough the universality problem, i.e., whether an automaton accepts all timed words, is highly undecidable.

Theorem 84 (Alur, Dill)
The universality problem is Π_1^1-complete.

This suggests that there is a difficulty with complementing timed automata. Indeed we have:

Fact 85 There is no timed automaton recognizing the complement of the language L_1 from the example above.

One solution to the problem with complement is to restrict to deterministic timed automata. These are automata that from any state at any moment have at most one transition for each letter in the alphabet. Unfortunately the class of languages recognized by deterministic timed automata is not closed under projection. Deterministic automata are discussed in [2].

In [3] a notion of event-time automaton is proposed. In this model we have clocks associated with letters of the alphabet. With each letter a we have a clock x_a telling how much time elapsed since the last occurrence of a and a clock y_a telling in how much time the next a will occur. So y_a is a kind of prophecy clock. The important difference with the ordinary timed automata is that there are no explicit ways of resetting clocks. We have:

Theorem 86 (Alur, Fix, Henzinger)
Deterministic and nondeterministic versions of event-time automata have the same expressive power. Every event clock automaton is equivalent to some standard clock automaton. The emptiness problem for event clock automata is decidable in PSPACE.

Unfortunately, event-clock automata are not closed under projection. Still [26] shows a correspondence between a (hierarchical) extension of event-timed automata model and some monadic second-order logic over timed sequences.

There are numerous other extensions/modifications of timed automata model, for some recent papers see [12, 14, 9]. These variations give us better understanding of the situation for reals but they also show that we do not yet have the same set of canonical notions as in the case of words or trees.

References

[1] M. Ajtaj, R. Fagin, and L. Stockmeyer. The colsure of monadic NP. In *STOC'98*, pages 309–318, 1998.

[2] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] R. Alur, L. Fix, and T. Henzinger. A determinizable class of timed automata. *Theoretical Computer Science*, 204, 1997.

[4] H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. Technical report, ILLC Research Report ML-96-03, 1996. 59 pages.

[5] A. Arnold. The mu-calculus alternation-depth hierarchy is strict on binary trees. *RAIRO–Theoretical Informatics and Applications*, 33:329–339, 1999.

[6] A. Arnold, G. Lenzi, and J. Marcinkowski. The hierarchy inside closed monadic Σ_1 collapses on the iinfinite binary tree. In *LICS'01*, 2001. To appear.

[7] A. Arnold and D. Niwiski. *The Rudiments of the Mu-Calculus*, volume 146 of *Studies in Logic*. North-Holand, 2001.

[8] J. Benthem. Dynamic bits and pieces. Technical report, University of Amsterdam, 1997. IILC research report.

[9] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Are timed automata updatable? In *CAV'2000*, volume 1855 of *LNCS*, pages 464–479, 2000.

[10] J. Bradfield. The modal mu-calculus alternation hierarchy is strict. *Theoretical Computer Science*, 195:133–153, 1997.

[11] J. Bradfield. Fixpoint alternation: Arithmetic, transition systems, and the binary tree. *RAIRO–Theoretical Informatics and Applications*, 33:341–356, 1999.

[12] B. Brard, V. Diekert, P. Gastin, , and A. Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2):145–182, 1998.

[13] J. Burgess and Y. Gurevich. The decision problem for linear temporal logic. *Notre Dame Journal of Formal Logic*, 26:115–128, 1985.

[14] C. Choffrut and M. Goldwurm. Timed automata with periodic clock constraints. *Journal of Automata, Languages and Combinatorics*, 5:371–404, 2000.

[15] K. J. Compton and C. W. Henson. A uniform method for proving lower bounds on the computational complexity of logical theories. *Annals of Pure and Applied Logic*, 48:1–79, 1990.

[16] W. Ebinger. Logical definability of trace languages. In V. Diekert and G. Rozenberg, editors, *The Book of Traces*, pages 382–390. World Scientific, 1995.

[17] E. A. Emerson, C. Jutla, and A. Sistla. On model-checking for fragments of μ-calculus. In *CAV'93*, volume 697 of *LNCS*, pages 385–396, 1993.

[18] E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. In *29th FOCS*, 1988.

[19] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proc. FOCS 91*, pages 368–377, 1991.

[20] P. Gastin and A. Petit. Asynchronous cellurar automata for infinite traces. In *ICALP '92*, volume 623 of *LNCS*, pages 583–594, 1992.

[21] E. Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 1999. to appear.

[22] E. Grädel, C. Hirsch, and M. Otto. Back and forth between guarded and modal logics. In *LICS'00*, pages 217–228, 2000.

[23] E. Grädel and I. Walukiewicz. Guarded fixpoint logic. In *LICS'99*, pages 45–55, 1999.

[24] A. Grzegorczyk. Some classes of recursive functions. *Rozprawy Matematyczne*, 4:1–45, 1953.

[25] Y. Gurevich and L. Harrington. Trees, automata and games. In *14th ACM Symp. on Theory of Computations*, pages 60–65, 1982.

[26] T. Henzinger, J.-F. Raskin, and P.-Y. Schobbens. The regular real-time languages. In *ICALP 97: Automata, Languages, and Programming*, volume 1443 of *LNCS*, pages 580–591. 1998.

[27] Y. Hirshfeld and A. Rabinovich. A framework for decidable metrical logics. In *ICALP 99*, volume 1664 of *LNCS*, pages 422–432, 1999.

[28] Y. Hirshfeld and A. Rabinovich. Quantitative temporal logic. In *CSL'99*, volume 1683 of *LNCS*, pages 172–187, 1999.

[29] D. Janin and G. Lenzi. Relating levels of the mu-calculus hierarchy and levels of the monadic hierarchy. In *LICS'01*, 2001. To appear.

[30] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *CONCUR'96*, volume 1119 of *LNCS*, pages 263–277, 1996.

[31] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[32] C. Löding. Optimal bounds for the transformation of omega-automata. In *FSTTCS'99*, volume 1738 of *LNCS*, pages 97–109, 1999.

[33] C. Löding and W. Thomas. Alternating automata and logics over infinite words. In *IFIP TCS 2000*, volume 1872 of *LNCS*, pages 521–535, 2000.

[34] O. Matz, N. Schweikardt, and W. Thomas. The monadic quantifier alternation hierarchy over grids and graphs. Information and Computation (to appear), 2000.

[35] A. Meyer. Weak monadic second order theory of one successor is not elementary. In *Lecture Notes in Mathematics*, volume 453, pages 132–154. Springer-Verlag, 1975.

[36] A. W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In A. Skowron, editor, *Fifth Symposium on Computation Theory*, volume 208 of *LNCS*, pages 157–168, 1984.

[37] A. W. Mostowski. Games with forbidden positions. Technical Report 78, University of Gdansk, 1991.

[38] A. W. Mostowski. Hierarchies of weak automata and week monadic formulas. *Theoretical Computer Science*, 83:323–335, 1991.

[39] D. Muller and P. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987.

[40] P. Niebert. A ν-calculus with local views for sequential agents. In *MFCS '95*, volume 969 of *LNCS*, pages 563–573, 1995.

[41] D. Niwiński. Fixed point characterization of infinite behaviour of finite state systems. *Theoretical Computer Science*, 189:1–69, 1997.

[42] D. Niwiński and I. Walukiewicz. Relating hierarchies of word and tree automata. In *STACS'98*, volume 1373 of *LNCS*. Springer-Verlag, 1998.

[43] M. Rabin. Weakly definable relations and special automata. In Y.Bar-Hillel, editor, *Mathematical Logic in Foundations of Set Theory*, pages 1–23. 1970.

[44] M. Rabin. Decidable theories. In J. Barwise, editor, *Handbook of Mathematical Logic*. Elsevier, 1977.

[45] S. Safra. On the complexity of ω-automata. In *29th IEEE Symp. on Foundations of Computer Science*, 1988.

[46] H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal of Computing*, 19:424–437, 1990.

[47] S. Shelah. The monadic second order theory of order. *Annals of Mathematics*, 102:379–419, 1975.

[48] C. S. Stirling. Modal and temporal logics. In S.Abramsky, D.Gabbay, and T.Maibaum, editors, *Handbook of Logic in Comuter Science*, pages 477–563. Oxford University Press, 1991.

[49] H. Straubing. *Finite Automata, Formal Logic and Circuit Complexity*. Birkhäuser, 1994.

[50] R. S. Streett and E. A. Emerson. An automata theoretic procedure for the propositional mu-calculus. *Information and Computation*, 81:249–264, 1989.

[51] W. Thomas. Automata on infinite objects. In J. van Leeuven, editor, *Handbook of Theoretical Computer Science Vol.B*, pages 133–192. Elsevier, 1990.

[52] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3. Springer-Verlag, 1997.

[53] I. Walukiewicz. Monadic second order logic on tree-like structures. In *STACS '96*, volume 1046 of *LNCS*, pages 401–414, 1996. Full version to appear in Theoretical Computer Science.

[54] I. Walukiewicz. Local logics for traces. Technical Report RS-00-2, BRICS, Aarhus University, 2000. Extended version to appear in Journal of Automata, Languages and Combinatorics.