Model Checking the Higher-Dimensional Modal μ-calculus

Martin Lange and Etienne Lozes

School of Electrical Engineering and Computer Science, University of Kassel, Germany*

Abstract. The higher-dimensional modal μ-calculus is an extension of the μ-calculus that has been introduced by Otto. The fascinating feature of this extension is that it precisely characterizes the bisimulation-invariant polynomial-time properties over finite structures. In this paper we investigate how the model checking problem of the higher-dimensional modal μ-calculus can be efficiently implemented. We propose two algorithms, based on extensions of local model checking and symbolic model checking algorithms respectively. We then illustrate the potential strength of these generic algorithms for deciding specific problems of various fields, as process equivalences, automata theory, parsing, string problems, or games.

1 Introduction

The Modal μ-Calculus $\mathcal{L}_\mu$ [Koz83] is an important modal fixpoint logic for defining temporal properties of transition systems, i.e. properties of reactive programs, concurrent programs, etc. [Sti01]. It can be seen as an extension of multi-modal logic with operators for recursion in the form of least and greatest fixpoint quantifiers. Alternatively, it can be seen as a fragment of Monadic Second-Order Logic since such fixpoint quantification can be expressed using second-order quantification. In fact, it is the largest fragment of Monadic Second-Order Logic that is bisimulation-invariant [JW96], i.e. that cannot distinguish between two bisimilar states of a transition system.

$\mathcal{L}_\mu$ is mostly known as a backbone for temporal logics used in program specification and verification which provide a more intuitive syntax and are therefore used more often like CTL [CE81] and CTL* [EH86]. The most intriguing logical problem in this domain is of course the model checking problem which is used to automatically prove correctness of programs. From a theoretical point of view, it is not answered entirely yet: it is known that the model checking problem for $\mathcal{L}_\mu$ and finite models is in NP∩coNP [EJS93], even in UP∩coUP [Jur98], and that it is equivalent under linear-time reductions to the problem of solving a parity game [Sti95].

* The European Research Council has provided financial support under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no 259267.

A question of reasonable relevance in this domain is also the expressive power of such a logic. It is basically answered by the result stated above: since bisimulation is considered to be *the* notion of program equivalence, bisimulation-invariance is a feature of a program specification logic, and $\mathcal{L}_\mu$ is therefore capable of expressing all regular properties of bisimulation-invariant tree models.

The expressive power of logics is the main concern in descriptive complexity theory [Imm99]. There, one considers formulas as a computational model which *describes* classes of structures rather than *accepts* them like a Turing machine for instance. A famous theorem in this field is Fagin's stating that NP is the class of all problems which can be described by a sentence in ESO—Second-Order Logic using existential second-order quantification only [Fag74]. A corresponding characterization of the complexity class P has not been found yet. Such a characterization by a logic $\mathcal{L}$ would of course reduce the famous problem of P $\stackrel{?}{=}$ NP to a question of whether $\mathcal{L}$ can be separated from ESO using tools like Ehrenfeucht-Fraïssé-games for instance [Ehr61,Fra54].

Some advancement has been made in this direction. It is known for instance that P is characterized by the extension of First-Order Logic with Least Fixpoint Quantifiers over the restricted class of all finite and ordered structures [Var82,Imm82]. This immediately raises the question of the expressive power of $\mathcal{L}_\mu$ in the context of descriptive complexity theory because it features least fixpoint quantification on top of Modal Logic rather than First-Order Logic. Furthermore, it is known that Modal Logic has the same expressive power as the bisimulation-invariant fragment of First-Order Logic [AvBN98]. It is therefore reasonable to ask whether $\mathcal{L}_\mu$ captures the bisimulation-invariant fragment of P—over finite and ordered structures at least. This is not the case, though. It is strictly weaker [Ott99]. Otto has defined an extension of the Modal μ-Calculus, the Higher-Dimensional μ-Calculus $\mathcal{L}_\mu^\omega$, and shown that this extension does have this feature.

Computational complexity theory still features many open questions regarding the separation of different complexity classes. The main purpose of descriptive complexity theory is to provide alternative and machine-independent characterizations of complexity classes which may lead to further separation or collapse results. In this paper we make use of descriptive complexity theory and the computational content of $\mathcal{L}_\mu^\omega$ in particular. In general, suppose there is a complexity class $\mathcal{C}$ and a logic $\mathcal{L}$ such that for every problem $L \in \mathcal{C}$ there is a formula $\varphi_L \in \mathcal{L}$ whose models are exactly (encodings of) the positive instances of L, i.e. $w \in L$ iff $enc(w) \models \varphi_L$ for some encoding of the instances of L as structures over which $\mathcal{L}$ can be interpreted. Suppose furthermore that there is a model checking algorithm $\mathrm{MC}_\mathcal{L}$ for $\mathcal{L}$. Note that a model checking algorithm takes two inputs: a structure and a formula. Then this yields algorithms for every problem in $\mathcal{C}$ simply by instantiating it with the corresponding formula and possibly using partial evaluation.

As stated before, this paper deals with the construction of algorithms for various problems using such logical characterizations; here we consider $\mathcal{L}_\mu^\omega$ and, clearly, problems in P which have bisimulation-invariant encodings via labeled

transition systems. Since—basically by definition—there are efficient algorithms for all problems known to be in P the merit of this research is the provision of a uniform treatment of such problems as model checking problems. It makes a big toolbox of methods and optimizations discovered in the area of program verification available to a much broader area, namely all problems definable in $\mathcal{L}_\mu^\omega$.

This paper is organized as follows. Section 2 introduces $\mathcal{L}_\mu^\omega$. Section 3 contains expositions of two methods for doing model checking for $\mathcal{L}_\mu$ which have proven to be of great value for the area or program verification: on-the-fly [SW91] and symbolic model checking [BCM+92]. In Section 4 we give examples of problems from various domains and show how they can be defined in $\mathcal{L}_\mu^\omega$ thus enabling on-the-fly as well as BDD-based symbolic algorithms for all such problems. Section 5 contains a theoretical development on the reduction to ordinary modal μ-calculus that underlies the two model checking algorithms of Section 3, and digressions concerning the satisfiability problem.

2 The Higher-Dimensional Modal μ-Calculus

2.1 Syntax

We assume infinite sets $\mathsf{Var} = \{x, y, \dots\}$ and $\mathsf{Var}_2 = \{X, Y, \dots\}$, of respectively first-order and second-order variables, and finite sets $\mathsf{P} = \{p, q, \dots\}$ and $\Sigma = \{a, b, \dots\}$ of respectively atomic propositions and labels. For tuples of first-order variables $\bar{x} = (x_1, \dots, x_n)$ and $\bar{y} = (y_1, \dots, y_n)$, with all x_i distinct, $\bar{x}{\leftarrow}\bar{y}$, denote the function $\kappa : \mathsf{Var}{\to}\mathsf{Var}$ such that $\kappa(x_i) = y_i$, and $\kappa(z) = z$ otherwise.

The syntax of the higher-dimensional modal μ-calculus $\mathcal{L}_\mu^\omega$ is reminiscent of that of the ordinary modal μ-calculus. However, modalities and propositions are relativized to a first-order variable, and it also features the *replacement* modality $\{\kappa\}$. Formulas of $\mathcal{L}_\mu^\omega$ are defined by the grammar

$$\varphi, \psi \;:=\; p(x) \mid X \mid \neg\varphi \mid \varphi \wedge \psi \mid \langle a \rangle_x \varphi \mid \mu X.\varphi \mid \{\kappa\}\varphi$$

where $x, y \in \mathsf{Var}$, $\kappa : \mathsf{Var}{\to}\mathsf{Var}$, $a \in \Sigma$, and $X \in \mathsf{Var}_2$.

A formula is of dimension n if it contains at most n distinct first-order variables; we write $\mathcal{L}_\mu^n$ to denote the set of formulas of dimension n. Note that $\mathcal{L}_\mu^1$ is equivalent to the standard modal μ-calulus: with a single first-order variable x, we have $p(x) \equiv p$, $\{x{\leftarrow}x\}\psi \equiv \psi$ and $\langle a \rangle_x \psi \equiv \langle a \rangle \psi$ for any ψ.

We actually restrict the syntax of formulas, like for $\mathcal{L}_\mu^1$, and require that every second-order variable gets bound by a fixpoint quantifier μ at most once in a formula. Then for every formula ϕ there is a function fp_ϕ which maps each second-order variable X occurring in ϕ to its unique binding formula $fp_\phi(X) = \mu X.\psi$. Finally, we allow occurrences of a second-order variable X only under the scope of an even number of negation symbols underneath $fp_\phi(X)$.

As usual, we write $\varphi \vee \psi$, $[a]_x\varphi$, and $\nu X.\phi$ to denote $\neg(\neg\varphi \wedge \neg\psi)$, $\neg\langle a \rangle_x \neg\varphi$, $\neg\mu X.\neg\varphi'$ respectively where φ' is obtained from φ by replacing every occurrence of X with $\neg X$.

Note that $\{\kappa\}$ is an operator in the syntax of the logic; it does not describe syntactic replacement of variables. Consider for instance the formula

$$\nu X. \bigwedge_{p \in \mathsf{P}} p(x) \Rightarrow p(y) \quad \wedge \quad \bigwedge_{a \in \Sigma} [a]_x \langle a \rangle_y X \quad \wedge \quad \{x, y \leftarrow y, x\} X.$$

As we will later see, this formula characterizes bisimilar states x and y. In this formula, the operational meaning of $\{x, y \leftarrow y, x\}X$ can be thought as "swapping the player's sides" in the bisimulation game.

We will sometimes require formulas to be in *positive normal form*. Such formulas are built from literals $p(x)$, $\neg p(x)$ and second-order variables X using the operators $\wedge$, $\vee$, $\langle a \rangle_x$, $[a]_x$, μ, ν, and $\{\kappa\}$. A formula is *closed* if all second-order variables are bound by some μ.

With $Sub(\varphi)$ we denote that set of all *subformulas* of φ. It also serves as a good measure for the *size* of a formula: $|\varphi| := |Sub(\varphi)|$. Another good measure of the complexity of the formula ϕ is its *alternation depth* ad_ϕ, *i.e* the maximal alternation of μ and ν quantifiers along any path in the syntactic tree of its positive normal form.

2.2 Labeled Transition Systems

A labeled transition system (LTS) is a graph whose vertices and edges are labeled with sets of propositional variables and labels respectively. Formally, a LTS is a tuple $\mathfrak{M} = (S, s_0, \Delta, \rho)$ such that $\Delta \subseteq S \times \Sigma \times S$ and $\rho : S \to \mathcal{P}(\mathsf{P})$. Elements of S are called states, and we write $s \xrightarrow{a} s'$ when $(s, a, s') \in \Delta$. The state $s_0 \in S$ is called the initial state of $\mathfrak{M}$.

We will mainly consider *finite* transition systems, *i.e.* transition systems (S, s_0, Δ, ρ) such that S is a finite set. Infinite-state transition systems arising from program verification are also of interest, but their model checking techniques differ from the ones of finite LTS and cannot be handled by our approach (see more comments on that point in the conclusion).

2.3 Semantics

A first-order valuation v over a LTS $\mathfrak{M}$ is a mapping from first-order variables to states, and a second order valuation is a mapping from second order variables to sets of first-order valuations:

$$\begin{aligned} \mathsf{Val} &\triangleq \mathsf{Var} \ \to S \\ \mathsf{Val}_2 &\triangleq \mathsf{Var}_2 \to \mathcal{P}(\mathsf{Val}) \end{aligned}$$

We write $v[\bar{x} \mapsto \bar{s}]$ to denote the first-order valuation that coincides with v, except that $x_i \in \bar{x}$ is mapped to the corresponding $s_i \in \bar{s}$. We use the same notation $\mathcal{V}[\bar{X} \mapsto \bar{P}]$ for second-order valuations. The semantics of a formula φ of $\mathcal{L}_\mu^\omega$ for a LTS $\mathfrak{M}$ and a second-order valuation $\mathcal{V}$ is defined as a set of first-order

valuations by induction on the formula:

$$\begin{array}{lcl}
[\![p(x)]\!]^{\mathscr{V}}_{\mathfrak{M}} & \triangleq & \{v : p \in \rho(v(x))\} \\
[\![\neg\varphi]\!]^{\mathscr{V}}_{\mathfrak{M}} & \triangleq & \mathsf{Val} - [\![\varphi]\!]^{\mathscr{V}}_{\mathfrak{M}} \\
[\![\varphi \wedge \psi]\!]^{\mathscr{V}}_{\mathfrak{M}} & \triangleq & [\![\varphi]\!]^{\mathscr{V}}_{\mathfrak{M}} \cap [\![\psi]\!]^{\mathscr{V}}_{\mathfrak{M}} \\
[\![\langle a\rangle_x \varphi]\!]^{\mathscr{V}}_{\mathfrak{M}} & \triangleq & \{v : \exists s.\ v(x) \xrightarrow{a} s \text{ and } v[x \mapsto s] \in [\![\varphi]\!]^{\mathscr{V}}_{\mathfrak{M}}\} \\
[\![X]\!]^{\mathscr{V}}_{\mathfrak{M}} & \triangleq & \mathscr{V}(X) \\
[\![\mu X.\varphi]\!]^{\mathscr{V}}_{\mathfrak{M}} & \triangleq & LFP\ \ \lambda P \in \mathcal{P}(\mathsf{Val}).\ [\![\varphi]\!]^{\mathscr{V}[X\mapsto P]}_{\mathfrak{M}} \\
[\![\{\bar{x}\leftarrow\bar{y}\}\varphi]\!]^{\mathscr{V}}_{\mathfrak{M}} & \triangleq & \{v : v[\bar{x} \mapsto v(\bar{y})] \in [\![\varphi]\!]^{\mathscr{V}}_{\mathfrak{M}}\}
\end{array}$$

We simply write $[\![\varphi]\!]_{\mathfrak{M}}$ to denote the semantics of a closed formula. We write $\mathfrak{M}, v \vDash \varphi$ if $v \in [\![\varphi]\!]_{\mathfrak{M}}$, and $\mathfrak{M} \vDash \varphi$ if $\mathfrak{M}, v_0 \vDash \varphi$, where v_0 is the constant function to s_0. Two formulas are equivalent, written $\varphi \equiv \psi$, if $[\![\varphi]\!]_{\mathfrak{M}} = [\![\psi]\!]_{\mathfrak{M}}$ for any LTS $\mathfrak{M}$. As with the normal modal μ-calculus, it is a simple exercise to prove that every formula is equivalent to one in positive normal form.

Proposition 1. *For every $\varphi \in \mathcal{L}^{\omega}_{\mu}$ there is a ψ in positive normal form such that $\varphi \equiv \psi$ and $|\psi| \leq 2 \cdot |\varphi|$.*

3 Model Checking

We describe two different ways of model checking a closed $\mathcal{L}^{\omega}_{\mu}$ formula in an LTS: a reduction to parity games, and a symbolic method using fixpoint iteration.

3.1 Local Model Checking

A *parity game* is a tuple $G = (Q, Q_0, Q_1, q_0, E, \Omega)$ where (Q, E) is a total, directed graph, $Q_0 \cup Q_1$ is a partition of Q into nodes owned by player 0, respectively 1, $q_0 \in Q$ is a designated starting node, and $\Omega : Q \to \mathbb{N}$ assigns priorities to the nodes. There are two important complexity measures for parity games: its size is $|G| = |Q|$, and its index is the number of different priorities occurring in it: $idx(G) = |\{\Omega(q)\ :\ q \in Q\}|$. As with LTS we write $q \longrightarrow q'$ rather than $(q, q') \in E$.

A play starts in q_0, and the owner of the current node q_i chooses a successor q_{i+1} such that $q_i \longrightarrow q_{i+1}$. The winner of an infinite play $q_0, q_1, \ldots$ is $\lim_{n\to\infty} \max_{i\geq n} \Omega(q_i)$, modulo 2. A (positional) winning strategy for player i is a function $\zeta : Q_i \to Q$ such that player i wins every play in which he/she makes always chooses $\zeta(q)$ whenever the play reaches a node $q \in Q_i$. The problem of *solving a parity game* is: given such a G, decide whether or not player 0 has a winning strategy for G.

Theorem 1 ([Zie98]). *A parity game of size n and index k can be solved in time $\mathcal{O}(n^k)$.*

It is well-known that the model checking problem for the Modal μ-Calculus can be reduced to solving a parity game [Sti95]. We now extend this construction to the higher-dimensional modal mu-calculus. Assume an $n \in \mathbb{N}$, an $\mathcal{L}_\mu^n$ formula ϑ in positive normal form and an LTS $\mathfrak{M} = (S, s_0, \Delta, \rho)$ to be fixed for this part, and let V be $fv(\vartheta)$. The parity game $G^n_{\mathfrak{M},\vartheta}$ is $(Q, Q_0, Q_1, q_0, E, \Omega)$ where

- $Q \triangleq (V \to S) \times Sub(\vartheta)$,
- $Q_0 \triangleq \{(v, \varphi) \mid \varphi \text{ is of the form } \psi_1 \vee \psi_2 \text{ or } \langle a \rangle_x \psi\}$,
- $Q_1 \triangleq Q - Q_0$,
- $q_0 \triangleq (v_0, \vartheta)$, with v_0 the constant valuation to s_0,
- the edge relation in the game is defined by

$$(v,\varphi) \longrightarrow \begin{cases} (v, p(x)) & , \text{ if } \varphi = p(x) \\ (v, \neg p(x)) & , \text{ if } \varphi = \neg p(x) \\ (v, \psi_i) & , \text{ if } \varphi = \psi_1 \otimes \psi_2 \text{ and } \otimes \in \{\wedge, \vee\} \text{ and } i \in \{1,2\} \\ (v', \psi) & , \text{ if } \varphi = \langle a \rangle_x \psi \text{ or } \varphi = [a]_x \psi, \text{ and} \\ & \quad \exists s \in S. v' = v[x \mapsto s] \text{ and } v(x) \xrightarrow{a} s \\ (v, \psi) & , \text{ if } \varphi = \eta X.\psi, \text{ or } \varphi = X \text{ and } fp_\vartheta(X) = \eta X.\psi \\ & \quad \text{for } \eta \in \{\mu, \nu\} \\ (v', \psi) & , \text{ if } \varphi = \{\kappa\}\psi \text{ and } v'(x) = v(\kappa(x)) \text{ for all } x \in V \end{cases}$$

- and the priorities solely depend on the formula component of a state:

$$\Omega(v,\varphi) \triangleq \begin{cases} ad_\vartheta(X) & , \text{ if } \varphi = X \\ 1 & , \text{ if } \varphi = p(x) \text{ and } p \notin \rho(v(x)), \\ & \quad \text{or } \varphi = \neg p(x) \text{ and } p \in \rho(v(x)) \\ 0 & , \text{ in all other cases.} \end{cases}$$

Theorem 2. *For every $n \in \mathbb{N}$, every $\mathcal{L}_\mu^n$ sentence φ in positive normal form, every LTS $\mathfrak{M}$, we have $\mathfrak{M} \models \varphi$ if and only if player 0 has a winning strategy for the parity game $G^n_{\mathfrak{M},\varphi}$.*

The proof of this is a consequence of the reduction of $\mathcal{L}_\mu^\omega$ to $\mathcal{L}_\mu^1$ that we present at the end of the paper (see the Reduction Theorem 4, Section 5): the parity game $G^n_{\mathfrak{M},v,\varphi}$ is nothing but the ordinary parity game on a "product" LTS $\mathsf{clone}_n(\mathfrak{M})$, and for a $\mathcal{L}_\mu^1$ formula $\widehat{\varphi}$ (see Section 5).

Theorem 2 then provides a multitude of model checking algorithms for $\mathcal{L}_\mu^\omega$ since there are many parity game solving algorithms. In particular, on-the-fly model checking for $\mathcal{L}_\mu^\omega$ is possible using local strategy improvement algorithms [FL11] and the alike [SS98]. There is also a tool that solves parity games well in practice and has implemented virtually all algorithms that are available in the literature [FL09].

3.2 Symbolic Model Checking

Symbolic model checking refers to the use of certain data structures for storing the results of intermediate computations when determining whether a given interpretation models a formula. Possibly the most prominent symbolic technique uses BDDs [Bry86] to represent sets of states in an LTS as well as the transition relations in it [BCM+92]. Success is based on the fact that necessary operations on state sets and these relations can efficiently be computed on such presentations. Here we describe how to extend BDD-based model checking to $\mathcal{L}_\mu^\infty$. We refer to the literature on symbolic model checking for a thorough introduction to the use of BDDs in verification. Here we consider a BDD simply as a Boolean function over some n, linearly ordered variables.

The key to BDD-based model checking for $\mathcal{L}_\mu^m$ is the fact that $[\![\varphi]\!]^{\mathscr{V}}_{\mathfrak{M}}$ can be represented as a Boolean function in $n \cdot m$ variables for any second-order valuation $\mathscr{V}$ and LTS $\mathfrak{M}$ with no more than 2^n many states. Suppose such an LTS $\mathfrak{M} = (S, s_0, \Delta, \rho)$ is given. W.l.o.g. we assume $S = \{0, \ldots, 2^n - 1\}$ for some $n \in \mathbb{N}$. It can be represented using $|\mathsf{P}| + |\Sigma|$ many Boolean functions in variables $x_0, \ldots, x_{n-1}$ as follows. We identify a vector $\bar{b}$ of n Boolean values with the natural number it represents in binary encoding. I.e. say $n = 3$ for instance, and consider the vector $(0, 1, 1)$. It represents a mapping of some variables x_0, x_1, x_2 to 0, 1, 1 respectively, and at the same time it denotes the state 6.

- For each $p \in \mathsf{P}$ there is a function $f_p(\bar{x})$ in n variables, defined by $f_q(\bar{s}) = 1$ iff $q \in \rho(\bar{s})$.
- For each $a \in \Sigma$ there is a function $f_a(\bar{x}, \bar{y})$ in $2n$ variables, defined by $f_a(\bar{s}, \bar{t}) = 1$ iff $\bar{s} \xrightarrow{a} \bar{t}$.

Furthermore, a set of first-order valuations $\mathcal{V} \subseteq \mathsf{Var} \to S$ for the m free variables $x_0, \ldots, x_{m-1}$ of an $\mathcal{L}_\mu^m$ formula can be represented by a Boolean function f in $m \cdot n$ variables as follows. $f_{\mathcal{V}}(\bar{s}_0, \ldots, \bar{s}_{m-1}) = 1$ iff $v \in \mathcal{V}$ where $v(x_i) = \bar{s}_i$ for all $i = 0, \ldots, m-1$.

Equally, a symbolic representation of a second-order valuation $\mathscr{V}$ is then simply a map from the second-order variables to such sets of first-order valuations.

We can then present an algorithm that computes for a given $\varphi \in \mathcal{L}_\mu^m$ and a symbolically represented second-order valuation $\mathscr{V}$, a symbolic representation of $[\![\varphi]\!]^{\mathscr{V}}_{\mathfrak{M}}$. The algorithm is presented on Figure 1. It manipulates Boolean functions as introduced above using for instance conjunction and negation. We write $f[\bar{y}/\bar{x}]$ for some vectors of variables $\bar{x}$ and $\bar{y}$ of the same arity, to denote the simultaneous replacement of the variables in $\bar{x}$ by those in $\bar{y}$.

Theorem 3. *For every $m \in \mathbb{N}$, every $\mathcal{L}_\mu^m$ formula φ, every second-order valuation $\mathscr{V}$ and every LTS $\mathfrak{M}$ we have $[\![\varphi]\!]^{\mathscr{V}}_{\mathfrak{M}} = \mathrm{MC}(\varphi, \mathscr{V})$ over this LTS $\mathfrak{M}$.*

The proof is by straightforward induction on φ, and consists in checking that the algorithmic operations tightly follow the definition of $[\![\varphi]\!]^{\mathscr{V}}_{\mathfrak{M}}$.

$\mathrm{MC}(\varphi,\mathscr{V}) =$
`case` φ `of`
- $p(x_i)$: `return` $f_p[\bar{x}_i/\bar{x}]$
- $\neg\psi$: `return` $\neg\mathrm{MC}(\psi,\mathscr{V})$
- $\psi_1 \wedge \psi_2$: `return` $\mathrm{MC}(\psi_1,\mathscr{V}) \wedge \mathrm{MC}(\psi_2,\mathscr{V})$
- $\langle a \rangle_{x_i}\psi$: $f := \mathrm{MC}(\psi,\mathscr{V})[\bar{y}/\bar{x}_i]$; `return` $\exists\bar{y}.f_a[\bar{x}_i/\bar{x}] \wedge f$
- X : `return` $\mathscr{V}(X)$
- $\mu X.\psi$: $f' :=$ `ff`; $\mathscr{V}' = \mathscr{V}[X \mapsto f']$; $f := \mathrm{MC}(\psi,\mathscr{V}')$; `while` $f \neq f'$ `do` $f' := f$; $\mathscr{V}' := \mathscr{V}[X \mapsto f']$; $f := \mathtt{MC}(\psi,\mathscr{V}')$; `done`; `return` f
- $\{\bar{y}\leftarrow\bar{z}\}\psi$: $f := \mathrm{MC}(\psi,\mathscr{V})$; `let` $(x_{i_1},\ldots,x_{i_k}) = \bar{y}$; `let` $(x_{j_1},\ldots,x_{j_k}) = \bar{z}$; `return` $f[\bar{x}_{j_0}/\bar{x}_{i_0},\ldots,\bar{x}_{j_k}/\bar{x}_{i_k}]$

Fig. 1. The symbolic model checking algorithm for $\mathcal{L}^{\omega}_{\mu}$.

4 Various Problems as Model Checking Problems

The model checking algorithms we presented can be exploited to solve any polynomial-time problem that can be encoded as a model checking problem in $\mathcal{L}^{\omega}_{\mu}$. This is hard to figure out which problems really fall into this category. By means of examples, we now intend to show that these problems are quite numerous.

4.1 Process Equivalences

The first examples are process equivalences encountered in process algebras. We only consider here strong simulation equivalence and bisimilarity, and let the interested reader think about how to encode other process equivalences, like weak bisimimilarity for instance.

Let us first remind some standard definitions. Let $\mathfrak{M} = (S, s_0, \Delta, \rho)$ be a fixed LTS. A *simulation* is a binary relation $R \subseteq S \times S$ such that for all (s_1, s_2) in R,

- for all $p \in \mathsf{P}$, if $p \in \rho(s_1)$ then $p \in \rho(s_2)$;
- for all $a \in \Sigma$ and $s'_1 \in S$, if $s_1 \xrightarrow{a} s'_1$, then there is $s'_2 \in S$ such that $s_2 \xrightarrow{a} s'_2$ and $(s'_1, s'_2) \in R$.

Two states s, s' are *simulation equivalent*, $s \backsimeq s'$, if there are simulations R, R' such that $(s, s') \in R$ and $(s', s) \in R'$. A simulation R is a *bisimulation* if $R = R^{-1}$; we say that s, s' are *bisimilar*, $s \sim s'$, if there is a bisimulation that contains (s, s'). We say that two valuations are bisimilar, $v \sim v'$, if for all $x \in \mathsf{Var}$, $v(x) \sim v'(x)$.

Proposition 2. *[Ott99] $\mathcal{L}^{\omega}_{\mu}$ is closed under bisimulation: if $v \in [\![\phi]\!]$ and $v \sim v'$, then $v' \in [\![\phi]\!]$.*

Let us now explain how these process equivalences can be decided by the model checking algorithms: the following formula captures valuations v such that $v(x) \sim v(y)$

$$\nu X. \bigwedge_{p \in \mathsf{P}} p(x) \Rightarrow p(y) \quad \wedge \quad \bigwedge_{a \in \Sigma} [a]_x \langle a \rangle_y X \quad \wedge \quad \{x, y \leftarrow y, x\} X$$

whereas the following formula captures valuations v such that $v(x) \backsimeq v(y)$

$$\nu X \big(\nu Y. \bigwedge_{p \in \mathsf{P}} p(x) \Rightarrow p(y) \quad \wedge \quad \bigwedge_{a \in \Sigma} [a]_x \langle a \rangle_y Y \big) \ \wedge \ \{x, y \leftarrow y, x\} X.$$

4.2 Automata Theory

A second application of $\mathcal{L}_\mu^\omega$ is in the field of automata theory. To illustrate this aspect, we pick some language inclusion problems that can be solved in polynomial-time.

A non-deterministic Büchi automaton can be viewed as a finite LTS $A = (S, s_0, \Delta, \rho)$ where ρ interpretes a predicate final. Remember that a run of an infinite word $w \in \Sigma^\omega$ over A is accepting if it visits infinitely many times a final state. The set of words $L(A) \subseteq \Sigma^\omega$ that have an accepting run is called the language accepted by A.

The language inclusion problem $L(A) \subseteq L(B)$ is exponential time for arbitrary Büchi automata, but in the restricted case of B being deterministic, it becomes polynomial time - remember that a Büchi automaton is called deterministic if for all $a \in \Sigma$, for all $s, s_1, s_2 \in S$, if $s \xrightarrow{a} s_1$ and $s \xrightarrow{a} s_2$, then $s_1 = s_2$.

Let us now encode the language inclusion problem $L(A) \subseteq L(B)$ as a $\mathcal{L}_\mu^\omega$ model checking problem. To shorten a bit the formula, we assume that B is moreover *complete*, *i.e.* for all $s \in S$, for all $a \in \Sigma$, there is at least one s' such that $s \xrightarrow{a} s'$. Consider the formula $\phi_{incl} \triangleq$

$$\langle synch \rangle^* \nu Z_1. \Big(\mathsf{final}(x) \wedge \neg \mathsf{final}(y) \wedge \mu Z_2. \langle synch \rangle \big(Z_1 \vee (\neg \mathsf{final}(y) \wedge Z_2) \big) \Big)$$

Let $\mathfrak{M}_{A,B}$ be the LTS obtained by disjoint union of A and B, creating an additional initial state s_0 and two transitions in_A and in_B from s_0 to the initial states of A and B. Then $L(A)$ is included in $L(B)$ if and only if $\mathfrak{M}_{A,B} \not\models \langle \mathsf{in}_A \rangle_x \langle \mathsf{in}_B \rangle_y \phi_{incl}$. Indeed, this formula is satisfiable if there is a run r_A of A and a run r_B of B reading the same word $w \in \Sigma^\omega$ such that r_A visits a final state of A infinitely often, whereas r_B eventually stops visiting the final states of B. Since B is deterministic, no other run r'_B could read w, thus $w \in L(A) \backslash L(B)$.

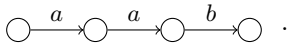
The same ideas can be applied to parity automata. A parity automaton is a finite automaton where states are assigned priorities; it can be seen as a LTS (S, s_0, Δ, ρ) where ρ interprets *priority predicates* prty_k in such a way that $\rho(s)$ is a singleton $\{\mathsf{prty}_k\}$ for all $s \in S$. A word $w \in \Sigma^\omega$ is accepted by a parity automaton if there is a run of w such that the largest priority infinitely often

visited is even. Consider the formulas $\mathsf{prty}_{\leq m}(x) = \mathsf{prty}_0(x) \vee \ldots \vee \mathsf{prty}_m(x)$ and $\phi_{n,m}$

$$\langle synch \rangle^* \nu Z. \langle synch' \rangle^+ \big(\mathsf{prty}_n(x) \wedge \langle synch' \rangle^+ (\mathsf{prty}_m(y) \wedge Z) \big)$$

where $\langle synch' \rangle^+ \phi$ is a shorthand for $\mu Z. \langle synch \rangle \mathsf{prty}_{\leq n}(x) \wedge \mathsf{prty}_{\leq m}(y) \wedge (\phi \vee Z)$. Then $\phi_{n,m}$ asserts that there are two runs r_A and r_B of two parity automata A and B recognizing the same word w such that the highest priorities visited infinitely often by r_A and r_B are respectively n and m; using the same idea as above, $\phi_{n,m}$ can be used to encode the inclusion problem $L(A) \subseteq L(B)$ when B is deterministic complete.

4.3 Parsing of Formal Languages

A third application of $\mathcal{L}_\mu^\omega$ is in the field of parsing for formal, namely context-free languages. To each finite word w, we may associate its linear LTS $\mathfrak{M}_w$ - for instance, for $w = aab$, $\mathfrak{M}_w$ is the LTS 

. Let us now consider a context-free grammar G, and define a formula that describes the language of G. To ease the presentation, we assume that G is in Chomsky normal form, but a linear-size formula would be derivable for an arbitrary grammar as well. The production rules of G are thus of the form either $X_i \to X_j X_k$ or $X_i \to a$, for $X_1, \ldots, X_n$ the non-terminals of G. To every non terminal X_i, we associate the recursive definition:

$$\phi_i \quad =_\mu \quad \bigvee_{X_i \to a} \langle a \rangle_x \; x \sim y \quad \vee \quad \bigvee_{X_i \to X_j X_k} \{z \leftarrow x\} \langle - \rangle_z^* \big((\{y \leftarrow z\} \phi_j) \wedge (\{x \leftarrow z\} \phi_k) \big)$$

where $x \sim y$ is the formula characterizing bisimilarity and $\langle - \rangle_z^* \phi$ is $\mu Z. \phi \vee \bigvee_{a \in \Sigma} \langle a \rangle_z Z$. If $v(x)$ and $v(y)$ are respectively the initial and final states of $\mathfrak{M}_w$, then $\mathfrak{M}_w, v \vDash \phi_i$ is equivalent to w being derivable in G starting with the symbol X_i.

4.4 String Problems

Model Checking for $\mathcal{L}_\mu^\infty$ can even be useful for computation (as opposed to decision) problems. Consider for example the Longest Common Subword problem: given words $w_1, \ldots, w_m$ over some alphabet Σ, find a longest v that is a subword of all w_i. This problem is NP-complete for an unbounded number of input words. Thus, we consider the problem restricted to some fixed m, and it is possible to define a formula $\varphi_{\mathrm{LCSW}}^m \in \mathcal{L}_\mu^m$ such that model checking this formula on a suitable representation of the w_i essentially computes such a common subword.

For the LTS take the disjoint union of all $\mathfrak{M}_{w_i}$ for $i = 1, \ldots, m$, and assume that each state in $\mathfrak{M}_{w_i}$ is labeled with a proposition p_i which makes it possible to define m-tuples of states in which the i-th component belongs to $\mathfrak{M}_{w_i}$. Now consider the formula

$$\varphi_{\mathrm{LCSW}}^m := \nu X. \bigwedge_{i=1}^m p_i(x_i) \wedge \bigvee_{a \in \Sigma} \langle a \rangle_1 \ldots \langle a \rangle_m X$$

Note that $\varphi^m_{\mathrm{LCSW}}$ is unsatisfiable for any $m \geq 1$. Thus, the symbolic model checking algorithm of Section 3.2 for instance would always return the empty set of tuples when called on this formula and any LTS. However, on an LTS representing $w_1, \ldots, w_m$ as described above it consecutively computes in the j-th round of the fixpoint iteration, all tuples of positions $h_1, \ldots, h_m$ such that the subwords in w_i from position $h_i - j$ to h_i are all the same for every $i = 1, \ldots, m$. Thus, it computes, in its penultimate round the positions inside the input words in which the longest common substrings end. Their starting points can easily be computed by maintaining a counter for the number of fixpoint iterations done in the model checking run.

In the same way, it is possible to compute the longest common subsequence of input words $w_1, \ldots, w_m$. A subsequence of w is obtained by deleting arbitrary symbols, whereas a subword is obtained by deleting an arbitrary prefix and suffix from w. The Longest Common Subsequence problem is equally known to be NP-complete for unbounded m. For any fixed m, however, the following formula can be used to compute all longest common subsequences of such input words using model checking technology in the same way as it is done in the case of the Longest Common Subword problem.

$$\varphi^m_{\mathrm{LCSS}} := \nu X. \bigwedge_{i=1}^{m} p_i(x_i) \wedge \bigvee_{a \in \Sigma} \langle a \rangle_1 \langle - \rangle_1^* \ldots \langle a \rangle_m \langle - \rangle_m^* X$$

where $\langle - \rangle_i^* \psi$ stands for $\mu Y. \psi \vee \bigvee_{a \in \Sigma} \langle a \rangle_i Y$.

4.5 Games

The Cat and Mouse Game is played on a directed graph with three distinct nodes c, m and t as follows. Initially, the cat resides in node c, the mouse in node m. In each round, the mouse moves first. He can move along an edge to a successor node of the current one or stay on the current node, then the cat can do the same. The mouse wins when he reaches the target node t, otherwise the cat wins. The problem of solving the Cat and Mouse Game is to decide whether or not the mouse has a winning strategy for such a given graph.

Note that this problem is not bisimulation-invariant under the straightforward encoding of the directed graph as an LTS with a single proposition t to mark the target node. Consider for example the following two, bisimilar game arenas.

Clearly, if the cat and mouse start on the two separate leftmost nodes then the mouse can reach the target first. However, these nodes are bisimilar to the left node of the right graph, and if they both start on this one then the cat has caught the mouse immediately.

Thus, winning strategies cannot necessarily be defined in $\mathcal{L}_\mu^\infty$. However, it is possible to define them when a new atomic formula $eq(x, y)$ expressing that x and y evaluate to the same node, is being added to the syntax of $\mathcal{L}_\mu^\infty$ (the model checking procedures in Section 3 can straightforwardly be extended to handle the equality predicate eq as well).

$$\varphi_{\mathrm{CMG}} := \mu X.(t(x) \wedge \neg eq(x,y)) \vee \langle - \rangle_x (t(x) \wedge \neg eq(x,y)) \wedge [-]_y X)$$

We have $v \models \varphi_{\mathrm{CMG}}$ if and only if the mouse can win from position $v(x)$ when the cat is on position $v(y)$ initially.

5 Reduction to the Ordinary μ-Calculus

The main ingredient for the correctness proofs of the model checking methods presented in the previous sections is the reduction theorem presented below. It describes how the model checking problem for $\mathcal{L}_\mu^n$ can be reduced to that of $\mathcal{L}_\mu^1$, i.e. the ordinary μ-calculus. Let us assume a fixed non-empty finite subset V of first-order variables. A formula φ of $\mathcal{L}_\mu^\omega$ with $fv(\varphi) \subseteq V$ can be seen as a formula $\hat{\varphi}$ of $\mathcal{L}_\mu^1$ over the set of the atomic propositions $\mathsf{P} \times V$ and the action labels $\Sigma \times V \cup (V \to V)$. We write p_x instead of (p, x) for elements of $\mathsf{P} \times V$, and equally a_x for elements from $\Sigma \times V$. Then $\varphi \mapsto \widehat{\varphi}$ can be defined as the homomorphism such that $\widehat{p(x)} \triangleq p_x$, $\widehat{\langle a \rangle_x \varphi} \triangleq \langle a_x \rangle \varphi$, and $\widehat{\{\bar{x} \leftarrow \bar{y}\}\varphi} \triangleq \langle \bar{x} \leftarrow \bar{y} \rangle \widehat{\varphi}$.

We call *higher-dimensional* a LTS that interprets the extended propositions p_x and modalities $\langle a_x \rangle$ and $\langle \kappa \rangle$ introduced by the formulas $\widehat{\varphi}$, and *ground* when it interprets the standard propositions and modalities. For a ground LTS $\mathfrak{M}$ and a formula φ, we thus need to define the higher-dimensional LTS over which $\widehat{\varphi}$ should be interpreted: we call it the V-*clone* of $\mathfrak{M}$, and write it $\mathsf{clone}_V(\mathfrak{M})$. Let us now detail the construction of $\mathsf{clone}_V(\mathfrak{M})$. Assume $\mathfrak{M} = (S, s_0, \Delta, \rho)$. Then $\mathsf{clone}_V(\mathfrak{M}) = (S', s_0', \Delta', \rho')$ is defined as follows.

- The states are valuations of the variables in V by states in S, *e.g* $S' = V \to S$, and s_0' is the constant function $\lambda x \in V.s_0$.
- The atomic propositions p_x is true in those new states, which assign x to an original state that satisfies p, *e.g.* $\rho'(v) = \{p_x \ : \ p \in \rho(v(x))\}$.
- The transitions contain labels of two kinds. First, there is an a_x-edge between two valuations v and v', if there is an a-edge between $v(x)$ and $v'(x)$ in the original LTS $\mathfrak{M}$:

$$v \xrightarrow{a_x} v' \quad \text{iff} \quad \exists t.v(x) \xrightarrow{a} t \text{ and } v' = v[x \mapsto t].$$

  For the other kind of transitions we need to declare the effect of applying a replacement to a valuation. Let $v : V \to S$ be a valuation of the first-order variables in V, and $\kappa : V \to V$ be a replacement operator. Let ${}^t\kappa(v)$ be the valuation such that ${}^t\kappa(v)(x) = v(\kappa(x))$. Then we add the following transitions to Δ'.

$$v \xrightarrow{\kappa} v' \quad \iff \quad v' = {}^t\kappa(v)$$

Note that the relation with label κ is functional for any such κ, i.e. every state in $\mathsf{clone}_V(\mathfrak{M})$ has exactly one κ-successor. Hence, we have $\langle\kappa\rangle\psi \equiv [\kappa]\psi$ over cloned LTS.

Theorem 4. *Let V be a finite set of first-order variables, let $\mathfrak{M} = (S, s_0, \Delta, \rho)$ be a ground LTS, and let φ be a $\mathcal{L}^\omega_\mu$ formula such that $fv(\varphi) \subseteq V$. Then*

$$\mathfrak{M} \models \varphi \quad \textit{if and only if} \quad \mathsf{clone}_V(\mathfrak{M}) \models \widehat{\varphi}.$$

The proof goes by straightforward induction on φ. This result may suggest that the theory of $\mathcal{L}^\omega_\mu$ should be "simple", as it is "nothing more" than the theory of $\mathcal{L}^1_\mu$ over cloned LTS. The following result may further suggest this impression.

Theorem 5. *Let V be a finite set of first-order variables. Then there is a formula φ_{cloned} in $\mathcal{L}^2_\mu$ such that for all higher-dimensional LTS $\mathfrak{M}$, the following two are equivalent:*

1. *$\mathfrak{M} \vDash \varphi_{cloned}$, and*
2. *there is a ground LTS $\mathfrak{M}'$ such that for all formula φ with $fv(\varphi) \subseteq V$,*

$$\mathfrak{M} \vDash \varphi \qquad \textit{if and only if} \qquad \mathsf{clone}_V(\mathfrak{M}') \vDash \varphi.$$

The proof goes by the logical characterization of conditions we will describe soon. Let us first comment this result. Theorem 5 shows that it is theoretically possible to reduce the satisfiability problem for the whole $\mathcal{L}^\omega_\mu$ to the one of $\mathcal{L}^2_\mu$: indeed, a $\mathcal{L}^\omega_\mu$ formula φ is satisfiable if and only if the $\mathcal{L}^2_\mu$ formula $\varphi_{cloned} \wedge \widehat{\varphi}$ is satisfiable. It would have been nice to further reduce to $\mathcal{L}^1_\mu$, since it would entail the decidability of the satisfiability of $\mathcal{L}^\omega_\mu$. But expressing φ_{cloned} in $\mathcal{L}^1_\mu$ is actually impossible: Otto showed that the satisfiability problem for $\mathcal{L}^2_\mu$ is undecidable, and even Σ^1_1-hard [Ott99]. Theorem 5 thus shows that all the complexity of $\mathcal{L}^\omega_\mu$ arises at level 2, and that the other levels are not more complicated; in other words, the complexity results of Otto for $\mathcal{L}^2_\mu$ extend to $\mathcal{L}^\omega_\mu$.

Let us now give the main ingredients of the proof of Theorem 5. Let $\mathfrak{M} = (S, s_0, \Delta, \rho)$ be an arbitrary higher-dimensional LTS. We define the ground LTS $\mathsf{unclone}_V(\mathfrak{M}) = (S', s'_0, \Delta', \rho')$ as follows:

- states of $\mathsf{unclone}_V(\mathfrak{M})$ are of the form $\langle s\ x\rangle$, *i.e.* $S' = S \times V$; such a state $\langle s\ x\rangle$ can be thought as "the x view of s";
- s'_0 is $\langle s_0\ x\rangle$ for one arbitrary x;
- $\langle s\ x\rangle$ satisfies p if s satisfies p_x in $\mathfrak{M}$, *i.e.* $\rho'(\langle s\ x\rangle) = \{p\ :\ p_x \in \rho(s)\}$;
- $\langle s\ x\rangle$ evolves to $\langle s'\ x\rangle$ after a if s evolves to s' after a_x in $\mathfrak{M}$, *i.e.* $\Delta' = \big\{\big(\langle s\ x\rangle, a, \langle s'\ x\rangle\big)\ :\ (s, a_x, s') \in \Delta\big\}$.

To provide a better readability, we write $\sim_g$ for the bisimilarity in a ground LTS and $\sim_{hd}$ for the one in a higher-dimensional one. We say that a higher-dimensional LTS $\mathfrak{M} = (S, s_0, \Delta, \rho)$ is *cloned* if all of the following conditions hold:

- for all $x, y \in V$, $\langle s_0\ x\rangle \sim_g \langle s_0\ y\rangle$;

– for all $s \in S$, for all $\kappa : V \to V$, there is at least one state s' such that $s \xrightarrow{\kappa} s'$, and if s_1, s_2 are two such κ successors of s, then $s_1 \sim_{hd} s_2$; we write ${}^t\kappa(s)$ to denote one (arbitrary) such successor;
– for all $s \in S$, for all $x \in V$, for all $\kappa : V \to V$, $\langle s\ \kappa(x)\rangle \sim_g \langle {}^t\kappa(s)\ x\rangle$
– for all $s, s' \in S$, for all $a \in \Sigma$, for all $x, y \in V$, $x \neq y$, if $s \xrightarrow{a_x} s'$, then $\langle s\ y\rangle \sim_g \langle s'\ y\rangle$.

Lemma 1. *For all ground LTS $\mathfrak{M}$, $\mathsf{clone}_V(\mathfrak{M})$ is cloned. Conversely, for all cloned higher-dimensional LTS $\mathfrak{M} = (S, s_0, \Delta, \rho)$, the higher-dimensional LTS $\mathsf{clone}_V(\mathsf{unclone}_V(\mathfrak{M}))$ is bisimilar to $\mathfrak{M}$ in the following sense:*

$$\textit{for all } s \in S \qquad s \sim_{hd} \lambda x \in V.\langle s\ x\rangle$$

Conclusion

We showed a reduction of the model checking problem of $\mathcal{L}_\mu^\omega$ to $\mathcal{L}_\mu^1$, and another reduction of the satisfiability problem of $\mathcal{L}_\mu^\omega$ to $\mathcal{L}_\mu^2$. The LTS transformations $\mathsf{clone}_V(.)$ and $\mathsf{unclone}_V(.)$ that support these reductions are obviously valid for infinite-state transition systems. It is thus worth discussing the model checking problem of $\mathcal{L}_\mu^\omega$ over infinite-state transition systems. The prominent result in the area is probably the decidability of this problem for $\mathcal{L}_\mu^1$ over pushdown LTS [Wal96]. However, the clone of a pushdown LTS is not a pushdown LTS in general, hence the model checking problem of $\mathcal{L}_\mu^\omega$ over pushdown LTS is not shown decidable by Theorem 4 — we actually conjecture that this problem is undecidable. The second author conjecture that reversal-bounded LTS could be a good example of infinite LTS for which model checking $\mathcal{L}_\mu^\omega$ would be decidable, due to the encouraging results of Bersani and Demri [**?**].

References

[AvBN98] H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27(3):217–274, 1998,.

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logics of Programs*, pages 52–71, Yorktown Heights, New York, 1981.

[EH86] E. A. Emerson and J. Y. Halpern. "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[Ehr61] A. Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fund. Math.*, 49:129–141, 1961.

[EJS93] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model-checking for fragments of μ-calculus. In *CAV'93*, pages 385–396, 1993.

[Fag74] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity and Computation*, 7:43–73, 1974.

[FL09] O. Friedmann and M. Lange. Solving parity games in practice. In *Proc. 7th Int. Symp. on Automated Technology for Verification and Analysis, ATVA'09*, volume 5799 of *LNCS*, pages 182–196, 2009.

[FL11] O. Friedmann and M. Lange. Two local strategy improvement schemes for parity game solving. *Journal of Foundations of Computer Science*, 2011. To appear.

[Fra54] R. Fraïssé. Sur quelques classifications des systèmes de relations. *Publ. Sci. Univ. Alger. Sér. A*, 1:35–182, 1954.

[Imm82] N. Immerman. Relational queries computable in polynomial time. In *Proc. 14th Symp. on Theory of Computing, STOC'82*, pages 147–152, Baltimore, USA, 1982. ACM.

[Imm99] N. Immerman. *Descriptive Complexity*. Springer-Verlag, New York, 1999.

[Jur98] M. Jurdziński. Deciding the winner in parity games is in $UP \cap$co-UP. *Inf. Process. Lett.*, 68(3):119–124, 1998.

[JW96] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional μ-calculus with respect to monadic second order logic. In *CONCUR*, pages 263–277, 1996.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, December 1983.

[Ott99] M. Otto. Bisimulation-invariant PTIME and higher-dimensional μ-calculus. *Theor. Comput. Sci.*, 224(1-2):237–265, 1999.

[SS98] P. Stevens and C. Stirling. Practical model-checking using games. In B. Steffen, editor, *Proc. 4th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'98*, volume 1384 of *LNCS*, pages 85–101. Springer, 1998.

[Sti95] C. Stirling. Local model checking games. In *Proc. 6th Conf. on Concurrency Theory, CONCUR'95*, volume 962 of *LNCS*, pages 1–11. Springer, 1995.

[Sti01] C. Stirling. *Modal and Temporal Properties of Processes*. Texts in Computer Science. Springer, 2001.

[SW91] C. Stirling and D. Walker. Local model checking in the modal μ–calculus. *TCS*, 89(1):161–177, 1991.

[Var82] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *Proc. 14th Symp. on Theory of Computing, STOC'82*, pages 137–146, San Francisco, CA, USA, 1982. ACM.

[Wal96] Igor Walukiewicz. Pushdown processes: Games and model checking. In *CAV*, pages 62–74, 1996.

[Zie98] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *TCS*, 200(1–2):135–183, 1998.