Model-Checking the Higher-Dimensional Modal μ-calculus

Martin Lange Etienne Lozes
School of Electr. Eng. and Computer Science, University of Kassel, Germany

The higher-dimensional modal μ-calculus is an extension of the μ-calculus in which formulas are interpreted in tuples of states of a labeled transition system. Every property that can be expressed in this logic can be checked in polynomial time, and conversely every polynomial-time decidable problem that has a bisimulation-invariant encoding into labeled transition systems can also be defined in the higher-dimensional modal μ-calculus. We exemplify the latter connection by giving several examples of decision problems which reduce to model checking of the higher-dimensional modal μ-calculus for some fixed formulas. This way generic model checking algorithms for the logic can then be used via partial evaluation in order to obtain algorithms for theses problems which may benefit from improvements that are well-established in the field of program verification, namely on-the-fly and symbolic techniques. The aim of this work is to extend such techniques to other fields as well, here exemplarily done for process equivalences, automata theory, parsing, string problems, and games.

1 Introduction

The Modal μ-Calculus $\mathscr{L}_\mu$ [6] is mostly known as a backbone for temporal logics used in program specification and verification. The most important decision problem in this domain is the model checking problem which is used to automatically prove correctness of programs. The model checking problem for $\mathscr{L}_\mu$ is well-understood by now. There are several algorithms and implementations for it. It is known that model checking $\mathscr{L}_\mu$ is equivalent under linear-time translations to the problem of solving a parity game [8] for which there also is a multitude of algorithms available. From a purely theoretical point of view, there is still the intriguing question of the exact computational complexity of model checking $\mathscr{L}_\mu$: the best known upper bound for finite models is UP$\cap$coUP [5], which is not entirely matched by the P-hardness inherited from model checking modal logic.

$\mathscr{L}_\mu$ can express exactly the bisimulation-invariant properties of tree or graph models which are definable in Monadic Second-Order Logic [4], i.e. are regular. This means that for every such set L of trees or graphs there is a fixed $\mathscr{L}_\mu$ formula φ_L s.t. a tree or graph G is a model of φ_L iff it belongs to L. Thus, any decision problem that has an encoding into regular and bisimulation-invariant sets of trees or graphs can in principle be solved using model checking technology. In detail, suppose there is a set M and a function f from the domain of M to graphs s.t. $\{f(x) \mid x \in M\}$ is regular and closed under bisimilarity. By the result above there is an $\mathscr{L}_\mu$ formula φ_M which defines (the encoding of) M. Now any model checking algorithm for $\mathscr{L}_\mu$ can be used in order to solve M.

Note that in theory this is just a reduction from M to the model checking problem for $\mathscr{L}_\mu$ on a fixed formula. Obviously reductions from any problem A to some problem B can be used to transfer algorithms from B to A, and the algorithm obtained for A can in general be at most as good as the algorithm for B unless it can be optimised for the fragment of B resulting from embedding A into it. However, there are two aspects that are worth noting in this context.

- A reduction to model checking for a fixed formula can lead to much more efficient algorithms. A model checking algorithm takes two inputs in general: a structure and a formula. If the formula is

D. Miller and Z. Ésik (Eds.): Fixed Points
in Computer Science 2012 (FICS 2012)
EPTCS 77, 2012, pp. 39–46, doi:10.4204/EPTCS.77.6

fixed then partial evaluation can be used in order to optimise the general scheme, throw away data structures, etc.

- Program verification is a very active research area which has developed many clever techniques for evaluating formulas in certain structures including on-the-fly [8] and symbolic methods [2], partial-order reductions, etc.

We refer to [1] for an example of this scheme of reductions to model checking for fixed formulas, there being done for problems that are at least PSPACE-hard. It also shows how this can be used to solve computation problems in this way. Since the data complexity (model checking with fixed formula) of $\mathscr{L}_\mu$ is in P, using this scheme for $\mathscr{L}_\mu$ is restricted to computationally simpler problems which can nevertheless benefit from developments in program verification. Furthermore, it is the presence of fixpoint operators in such a logic which makes it viable to this approach: fixpoint operators can be used to express inductive concepts—e.g. the derivation relation in a context-free grammar—and at the same time provide the foundation for algorithmic solutions via fixpoint iteration for instance.

Here we consider an extension of $\mathscr{L}_\mu$, the Higher-Dimensional Modal μ-Calculus $\mathscr{L}_\mu^\omega$, and investigate its usefulness regarding the possibility to obtain algorithmic solutions to various decision or computation problems which may benefit from techniques originally developed for program verification purposes only. It is known that $\mathscr{L}_\mu^\omega$ captures the bisimulation-invariant fragment of P. We will sketch how the $\mathscr{L}_\mu^\omega$ model checking problem can be reduced to $\mathscr{L}_\mu$ model checking via a simple product construction on transition systems. Thus we can obtain—in principle—an algorithm for every problem that admits a polynomial-time solution and a bisimulation-invariant encoding into graphs. The reduction from $\mathscr{L}_\mu^\omega$ to $\mathscr{L}_\mu$ is compatible with on-the-fly or BDD-based model checking techniques, thus transferring such algorithms from $\mathscr{L}_\mu$ first to $\mathscr{L}_\mu^\omega$ and then on to such decision problems.

2 The Higher-Dimensional Modal μ-Calculus

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, an LTS over a set $\Sigma = \{a,b,\dots\}$ of edge labels and a set $\mathsf{P} = \{p,q,\dots\}$ of atomic propositions is a tuple $\mathfrak{M} = (S,s_0,\Delta,\rho)$ such that $s_0 \in S$, $\Delta \subseteq S \times \Sigma \times S$ and $\rho : S \to \mathscr{P}(\mathsf{P})$. 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).

Syntax. We assume infinite sets $\mathsf{Var} = \{x,y,\dots\}$ and $\mathsf{Var}_2 = \{X,Y,\dots\}$, of first-order and second-order variables respectively. For tuples of first-order variables $x = (x_1,\dots,x_n)$ and $y = (y_1,\dots,y_n)$, with all x_i distinct, $x \leftarrow y$, denotes the function $\kappa : \mathsf{Var} \to \mathsf{Var}$ such that $\kappa(x_i) = y_i$, and $\kappa(z) = z$ otherwise. It is called a *variable replacement*.

The syntax of the higher-dimensional modal μ-calculus $\mathscr{L}_\mu^\omega$ 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 $\mathscr{L}_\mu^\omega$ are defined by the grammar

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

where $x,y \in \mathsf{Var}$, $\kappa : \mathsf{Var}{\to}\mathsf{Var}$ is a variable replacement with finite domain, $a \in \Sigma$, and $X \in \mathsf{Var}_2$. We 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 $fp_\varphi(X) = \mu X.\psi$. Finally, we allow occurrences of a second-order variable X only under the scope of an even number of negation symbols underneath $fp_\varphi(X)$.

A formula is of dimension n if it contains at most n distinct first-order variables; we write $\mathscr{L}^n_\mu$ to denote the set of formulas of dimension n. Note that $\mathscr{L}^1_\mu$ 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.

As usual, we write $\varphi \vee \psi$, $[a]_x\varphi$, and $\nu X.\varphi$ 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 $\neg X$. Other Boolean operators like $\Rightarrow$ and $\Leftrightarrow$ are defined as usual.

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

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

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 players' pebbles" 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.

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:

$$\begin{array}{rcrcl} \mathsf{Val} & \triangleq & \mathsf{Var} & \to & S \\ \mathsf{Val}_2 & \triangleq & \mathsf{Var}_2 & \to & \mathscr{P}(\mathsf{Val}) \end{array}$$

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

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

We simply write $[\![\varphi]\!]_{\mathfrak{M}}$ to denote the semantics of a closed formula. We write $\mathfrak{M}, v \models \varphi$ if $v \in [\![\varphi]\!]_{\mathfrak{M}}$, and $\mathfrak{M} \models \varphi$ if $\mathfrak{M}, v_0 \models \varphi$, 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 $\varphi \in \mathscr{L}_\mu^\omega$ there is a ψ in positive normal form such that $\varphi \equiv \psi$ and $|\psi| \leq 2 \cdot |\varphi|$.*

Reduction to the Ordinary μ-Calculus. Here we consider $\mathscr{L}_\mu^\omega$ as a formal language for defining decision problems. Algorithms for these problems can be obtained from model checking algorithms for $\mathscr{L}_\mu$ on fixed formulas using partial evaluation. In order to lift all sorts of special techniques which have been developed for model checking in the area of program verification we show how to reduce the $\mathscr{L}_\mu^\omega$ model checking problem to that of $\mathscr{L}_\mu^1$, i.e. the ordinary μ-calculus.

Let us assume a fixed non-empty finite subset V of first-order variables. A formula φ of $\mathscr{L}_\mu^\omega$ with $fv(\varphi) \subseteq V$ can be seen as a formula $\hat{\varphi}$ of $\mathscr{L}_\mu^1$ 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 $\widehat{p(x)} \triangleq p_x$, $\widehat{\langle a \rangle_x \varphi} \triangleq \langle a_x \rangle \widehat{\varphi}$, and $\widehat{\{x{\leftarrow}y\}\varphi} \triangleq \langle x{\leftarrow}y \rangle \widehat{\varphi}$.

We call an LTS *higher-dimensional* when it 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 $\mathsf{clone}_V(\mathfrak{M})$. Roughly speaking, $\mathsf{clone}_V(\mathfrak{M})$ is the asynchronous product of $|V|$ copies of $\mathfrak{M}$. More formally, 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 proposition 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'.

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

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 2. *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 $\mathscr{L}_\mu^\omega$ formula such that $fv(\varphi) \subseteq V$. Then*

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

The proof goes by straightforward induction on φ and is therefore ommitted – see also the chapter on descriptive complexity in [3] for similar results. The importance of Thm. 2 is based on the fact that it transfers many model checking algorithms for the modal μ-calculus to $\mathscr{L}_\mu^1$, for example on-the-fly model checking [8], symbolic model checking [2] with BDDs or via SAT, strategy improvement schemes [9], etc.

3 Various Problems as Model Checking Problems

The model checking algorithms we mentioned can be exploited to solve any polynomial-time problem that can be encoded as a model checking problem in $\mathscr{L}_\mu^\omega$. By means of examples, we now intend to show that these problems are quite numerous.

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 bisimilarity for instance.

Let us first recall 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 $p \in \mathsf{P}$: $p \in \rho(s_1)$ iff $p \in \rho(s_2)$;
- 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*, $s \simeq s'$, 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 3. *[7] $\mathscr{L}_\mu^\omega$ is closed under bisimulation: if $v \in [\![\varphi]\!]$ and $v \sim v'$, then $v' \in [\![\varphi]\!]$.*

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

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

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

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

Automata Theory. A second application of $\mathscr{L}_\mu^\omega$ 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 interprets a predicate final. Remember that a run on an infinite word $w \in \Sigma^\omega$ in A is accepting if it visits infinitely often 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 PSPACE-hard for arbitrary Büchi automata and therefore unlikely to be definable in $\mathscr{L}_\mu^\omega$. In the restricted case of B being deterministic, it becomes solvable in 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 $\mathscr{L}_\mu^\omega$ 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 $s \xrightarrow{a} s'$. Let us introduce the modality $\langle synch \rangle \varphi \triangleq \bigvee_{a \in \Sigma} \langle a \rangle_x \langle a \rangle_y \varphi$. Consider the formula

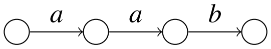
$$\varphi_{incl} \;\triangleq\; \langle synch \rangle^* \nu Z_1. \Big(\mathsf{final}(x) \wedge \neg\mathsf{final}(y) \wedge \mu Z_2. \langle synch \rangle \big(Z_1 \vee (\neg\mathsf{final}(y) \wedge Z_2)\big)\Big)$$

Let $\mathfrak{M}_{A,B}$ be the LTS obtained as the disjoint union of A and B with initial states s_A of A and s_B of B respectively. Then $L(A)$ is included in $L(B)$ if and only if $\mathfrak{M}_{A,B}, v \not\models \varphi_{incl}$ where $v(x) = s_A$ and $v(y) = s_B$. Indeed, this formula is satisfied 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 an 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 visited infinitely often is even. Consider the formulas $\mathsf{prty}_{\leq m}(x) = \mathsf{prty}_0(x) \vee \ldots \vee \mathsf{prty}_m(x)$ and

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

where $\langle synch'\rangle^+\varphi$ is a shorthand for $\mu Z.\langle synch\rangle\, \mathsf{prty}_{\leq n}(x) \wedge \mathsf{prty}_{\leq m}(y) \wedge (\varphi \vee Z)$. Then $\varphi_{n,m}$ 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. Since $L(A) \not\subseteq L(B)$ if and only there is an even n and an odd m such that $\mathfrak{M}_{A,B} \models \varphi_{n,m}$, this gives us again a decision procedure for the language inclusion problem of parity automata when B is deterministic complete.

Parsing of Formal Languages. A third application of $\mathscr{L}_\mu^\omega$ 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 context-free grammar as well. The production rules of G are thus of the form either $X_i{\to}X_jX_k$ or $X_i{\to}a$, for $X_1, \ldots, X_n$ the non-terminals of G. Let us pick variables x,y and z, intended to represent respectively the initial the final, and an intermediate position in the (sub)word currently parsed. To every non-terminal X_i, we associate the recursive definition:

$$\varphi_i \;=_\mu\; \bigvee_{X_i\to a} \langle a\rangle_x\, x \sim y \;\vee\; \bigvee_{X_i\to X_jX_k} \{z{\leftarrow}x\}\langle -\rangle_z^*\big((\{y{\leftarrow}z\}\varphi_j) \wedge (\{x{\leftarrow}z\}\varphi_k)\big)$$

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

String Problems. Model Checking for $\mathscr{L}_\mu^\infty$ 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 $\varphi^m_{\mathrm{LCSW}} \in \mathscr{L}_\mu^m$ 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

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

Note that $\varphi^m_{\mathrm{LCSW}}$ is unsatisfiable for any $m \geq 1$. Thus, a symbolic model checking algorithm for instance would always return the empty set of tuples when called on this formula and any LTS. However, on an LTS representing $w_1,\dots,w_m$ as described above it consecutively computes in the j-th round of the fixpoint iteration, all tuples of positions $h_1,\dots,h_m$ such that the subwords in w_i from position $h_i - j$ to h_i are all the same for every $i = 1,\dots,m$. 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.

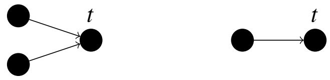
In the same way, it is possible to compute the longest common subsequence of input words $w_1,\dots,w_m$. A subsequence of w is obtained by deleting arbitrary symbols, whereas a subword is obtained by deleting an arbitrary prefix and suffix from w. The Longest Common Subsequence problem is equally known 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.

$$\varphi^m_{\mathrm{LCSS}} := \nu X. \bigwedge_{i=1}^{m} p_i(x_i) \wedge \bigvee_{a\in\Sigma} \langle a\rangle_{x_1} \langle -\rangle^*_{x_1} \dots \langle a\rangle_{x_m} \langle -\rangle^*_{x_m} X$$

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

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. If the cat reaches the mouse, she wins; otherwise, if the mouse reaches the target node t, he wins; otherwise, the mouse runs forever without being caught nor reaching the target node: in that case, 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 a given graph.

Note that this problem is not bisimulation-invariant under the straight-forward 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 $\mathscr{L}^\infty_\mu$. 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 $\mathscr{L}^\infty_\mu$ (standard model checking procedures can be extended to handle the equality predicate eq as well).

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

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.

4 Conclusion

We have considered the modal fixpoint logic $\mathscr{L}_\mu^\omega$ for a potential use in algorithm design and given examples of problems which can be defined in $\mathscr{L}_\mu^\omega$. The combination of fixpoint quantifiers and modal operators has been proved to be very fruitful for obtaining algorithmic solutions for problems in automatic program verification. The examples boost the idea of using successful model checking technology in other areas too.

The use of model checking algorithms on fixed formulas does not provide a generic recipe that miraculously generates efficient algorithms, but it provides the potential to do so. The next step on this route towards an efficient algorithm for some problem P requires partial evaluation on a model checking algorithm and the formula φ_P defining P. This usually requires manual tweaking of the algorithm and is highly dependent on the actual φ_P. Thus, future work on this direction would consist of consequently optimising $\mathscr{L}_\mu^\omega$ model checking algorithms for certain definable problems and testing their efficiency in practice.

On a different note, $\mathscr{L}_\mu^\omega$ is an interesting fixpoint calculus for which the model checking problem over infinite-state transition systems has not been quite studied so far. The most prominent result in this area is the decidability of $\mathscr{L}_\mu^1$ over pushdown LTS [10]. However, model checking $\mathscr{L}_\mu^\omega$ — or even just $\mathscr{L}_\mu^k$ for some $k \geq 2$ — seems undecidable for pushdown LTS. It is questionable whether model checking of $\mathscr{L}_\mu^\omega$ is decidable for any popular class of infinite-state transition systems.

References

[1] R. Axelsson & M. Lange (2007): *Model Checking the First-Order Fragment of Higher-Order Fixpoint Logic*. In: *Proc. 14th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR'07*, *LNCS* 4790, Springer, pp. 62–76, doi:10.1007/978-3-540-75560-9_7.

[2] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill & L. J. Hwang (1992): *Symbolic Model Checking: 10^{20} States and Beyond*. *Information and Computation* 98(2), pp. 142–170, doi:10.1016/0890-5401(92)90017-A.

[3] E. Grädel, P. G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Y. Vardi, Y. Venema & S. Weinstein (2007): *Finite Model Theory and its Applications*. Springer-Verlag, doi:10.1007/3-540-68804-8.

[4] D. Janin & I. Walukiewicz (1996): *On the Expressive Completeness of the Propositional μ-Calculus with Respect to Monadic Second Order Logic*. In: *CONCUR*, pp. 263–277, doi:10.1007/3-540-61604-7_60.

[5] M. Jurdziński (1998): *Deciding the winner in parity games is in UP∩co-UP*. *Inf. Process. Lett.* 68(3), pp. 119–124, doi:10.1016/S0020-0190(98)00150-1.

[6] D. Kozen (1983): *Results on the Propositional μ-calculus*. *TCS* 27, pp. 333–354, doi:10.1007/BFb0012782.

[7] M. Otto (1999): *Bisimulation-invariant PTIME and higher-dimensional μ-calculus*. *Theor. Comput. Sci.* 224(1-2), pp. 237–265, doi:10.1016/S0304-3975(98)00314-4.

[8] C. Stirling (1995): *Local Model Checking Games*. In: *Proc. 6th Conf. on Concurrency Theory, CONCUR'95*, *LNCS* 962, Springer, pp. 1–11, doi:10.1007/3-540-60218-6_1.

[9] J. Vöge & M. Jurdziński (2000): *A Discrete Strategy Improvement Algorithm for Solving Parity Games*. In: *CAV*, pp. 202–215, doi:10.1007/10722167_18.

[10] Igor Walukiewicz (1996): *Pushdown Processes: Games and Model Checking*. In: *CAV*, pp. 62–74, doi:10.1007/3-540-61474-5_58.