Theoretical Computer Science 412 (2011) 3129–3139

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

A linear translation from CTL* to the first-order modal μ-calculus

Sjoerd Cranen, Jan Friso Groote *, Michel Reniers

Technische Universiteit Eindhoven, The Netherlands

ARTICLE INFO

Keywords:
Modal μ-calculus
CTL*
LTL
Temporal logic
Translation
Linear

ABSTRACT

The modal μ-calculus is a very expressive temporal logic. In particular, logics such as LTL, CTL and CTL* can be translated into the modal μ-calculus, although existing translations of LTL and CTL* are at least exponential in size. We show that an existing simple first-order extension of the modal μ-calculus allows for a linear translation from LTL. Furthermore, we show that solving the translated formulae is as efficient as the best known methods to solve LTL formulae directly.

1. Introduction

Designing complex distributed systems in such a way that they behave correctly is a challenging task. One attempt to deal with this challenge is to describe the system using a behavioural modelling formalism, such as interacting automata or process algebras. Experience teaches that such descriptions are not by themselves correct and therefore it is useful to establish behavioural properties such as absence of deadlock, safety and liveness properties that are described in a modal logic.

We use mCRL2 as a behavioural modelling formalism, which is a process algebraic behavioural specification formalism endowed with data and time, which is used extensively to model real life systems [1,13]. Properties about the behaviour of processes are described in a modal μ-calculus enriched with data and time, the first-order modal μ-calculus [14]. The modal μ-calculus [9,19] is an extension of Hennessy–Milner logic [16] with fixpoint operators.

By using data in mCRL2, one can specify state machines with an infinite action alphabet, giving rise to the need for a formalism that can express behavioural properties over such systems. The first-order modal μ-calculus, in which quantification over data can be used and in which fixpoint variables may have data parameters, fulfils this need and is more practical as it is syntactically less minimalistic. It is very expressive and – after some training – very pleasant to use. Indeed, over the years we have not yet encountered any behavioural property that we could not express in this formalism. Driven by these positive results, we developed theories [12,15] and tools to verify properties in the first-order modal μ-calculus with data. These tools are distributed in the freely available, open source mCRL2 toolkit [1,13].

The purpose of this paper is to formally establish what we already experienced in practice, namely that the first-order μ-calculus with data is indeed very expressive in the sense that properties formulated in other modal logics can be translated to it with only a linear growth in size.

Already in 1986, Emerson and Lei suggested that the modal μ-calculus might serve as a uniform model checking framework, and showed that CTL can be translated succinctly into the modal μ-calculus. However, they also noted that the only known translation from CTL* to the μ-calculus was not succinct [10]. But if the modal μ-calculus is to become a framework for model checking, it is certainly of importance that system properties can be expressed in a formula that is roughly comparable in size with a CTL* formula.

The original translation that Emerson and Lei mentioned consisted of the composition of an unpublished translation from CTL* to PDL-Δ by Wolper, and a translation from PDL-Δ to the μ-calculus [10]. A simpler translation procedure was

* Corresponding author. Tel.: +31 40 2475003.
E-mail addresses: s.cranen@tue.nl (S. Cranen), j.f.groote@tue.nl (J.F. Groote), m.a.reniers@tue.nl (M. Reniers).

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.02.034

proposed in [8], but this translation still yields formulae doubly exponential in the size of the input formula. Only in 1996, this translation was improved upon by Bhat et al. with an algorithm that translates CTL* to an equational variant of the modal μ-calculus, causing only a single exponential blowup.

In this paper, we use a strategy similar to that of Bhat et al.; we first focus on translating LTL, and then extend the translation to handle CTL* formulae. We show that a linear translation to the first-order modal μ-calculus is possible using only very simple data types.

From the context of the mCRL2 toolkit, there is also a very practical reason to have a succinct translation from LTL or CTL*. For those unfamiliar with the modal μ-calculus, or for those who favour these logics over the modal μ-calculus (and admittedly, many people do so at the time of writing), a linear translation enables us to easily use available μ-calculus-checkers to verify properties formulated in these other formalisms also. To support this, we show that model checking a translated formula is as efficient as the most efficient algorithms known for model checking the original.

The paper is structured as follows. Section 2 introduces CTL* and LTL, and shows how an LTL formula can be represented by a Büchi automaton. In Section 3, we present the first-order modal μ-calculus to which LTL formulae are translated in Section 4. This translation is then lifted to one for CTL* in Section 5.

2. LTL and Büchi automata

In this section we introduce CTL* and LTL and their semantics in terms of Kripke structures. Then, a translation from LTL to Büchi automata is discussed, which is the basis of the translation presented in Section 4. Readers familiar with the subject matter may skip any part of this section, although we point out that the rest of this article relies quite heavily on the topics presented in Sections 2.3 and 2.4, and therefore also on the notation used in those sections.

2.1. Kripke structures

A *Kripke structure* M is a tuple $\langle S, \rightarrow, AP, L\rangle$, where

- S is a set of states,
- $\rightarrow \subseteq S \times S$ is a transition relation such that $\forall_{s\in S}\, \exists_{s'\in S}\, \langle s, s'\rangle \in \rightarrow$,
- AP is a set of atomic propositions and
- $L : S \rightarrow 2^{AP}$ is a labelling function.

A path π is a (possibly infinite) sequence $s_0, s_1, \ldots$ of nodes from S in which every pair s_i, s_{i+1} of subsequent nodes satisfies $s_i \rightarrow s_{i+1}$. If π is a path and i a natural number, then π^i denotes π without its first i states.

The *size* of a Kripke structure M, denoted $|M|$, is equal to the number of states plus the number of transitions in M, *i.e.*, $|M| = |S| + |{\rightarrow}|$.

2.2. CTL* and LTL

In this paper we use the definition of CTL* and LTL as given in [18]. We first describe the syntax of *state formulae* and *path formulae*. Given some set AP of atomic predicates, the syntax of a state formula is given by the following grammar:

$$f ::= a \mid \neg f \mid f \vee f \mid \mathsf{A}g$$

In the above, $a \in AP$ and g is a path formula generated by the following grammar, in which f is again a state formula:

$$g ::= f \mid \neg g \mid g \vee g \mid \mathsf{X}g \mid g\,\mathsf{U}\,g$$

CTL formulae* are state formulae as described by the top grammar. The *size* of a CTL* formula f, denoted $|f|$, is the number of subformulae it contains. The following abbreviations are in common use.

$$\begin{array}{lll} \mathbf{true} \triangleq a \vee \neg a & \mathsf{F}f \triangleq \mathbf{true}\,\mathsf{U}\,f & \mathsf{E}f \triangleq \neg\mathsf{A}\neg f \\ \mathbf{false} \triangleq \neg\mathbf{true} & \mathsf{G}f \triangleq \neg\mathsf{F}\neg f & \\ f \wedge g \triangleq \neg(\neg f \vee \neg g) & f\,\mathsf{R}\,g \triangleq \neg(\neg f\,\mathsf{U}\,\neg g) & \end{array}$$

Linear temporal logic (LTL) is a subset of CTL*. *LTL formulae* are CTL* formulae of the form $\mathsf{A}f$, where the A and E operators do not occur in f.

The semantics of CTL* formulae (and LTL formulae) is defined on states in a Kripke structure $M = \langle S, \rightarrow, AP, L\rangle$. We will write $M, s \models f$ if CTL* formula f holds in state $s \in S$. If g is a path formula, then we also define $\pi \models g$ for every path π in M, meaning that g holds along π. If a is an atomic predicate, f and f' are state formulae, and g is a path formula, then the interpretation for state formulae is as follows:

$$\begin{array}{ll} M, s \models a & \text{iff } a \in L(s) \\ M, s \models \neg f & \text{iff not } M, s \models f \\ M, s \models f \vee f' & \text{iff } M, s \models f \text{ or } M, s \models f' \\ M, s \models \mathsf{A}g & \text{iff for every path } \pi \text{ starting in } s,\ \pi \models g \end{array}$$

The interpretation for path formulae is described by the following rules, where f is a state formula, and g and g' are path formulae.

$M, \pi \models f$	iff s is the first state of π and $M, s \models f$
$M, \pi \models \neg g$	iff not $M, \pi \models g$
$M, \pi \models g \vee g'$	iff $M, \pi \models g$ or $M, \pi \models g'$
$M, \pi \models \mathsf{X} g$	iff $M, \pi^1 \models g$
$M, \pi \models g \mathsf{U} g'$	iff $M, \pi^k \models g'$ for some k and $M, \pi^j \models g$ for all $0 \leq j < k$

If it is clear from the context which Kripke structure should be used to evaluate a CTL* formula, we will omit it and write $s \models f$ rather than $M, s \models f$.

2.3. Nondeterministic Büchi automata

A *nondeterministic Büchi automaton* is defined as a tuple $\langle Q, \Sigma, \delta, Q_0, F\rangle$ where

- Q is a set of states,
- Σ is an alphabet,
- $\delta: Q \times \Sigma \to 2^Q$ is a transition function,
- $Q_0 \subseteq Q$ is the set of initial states and
- $F \subseteq Q$ is the acceptance set.

If the alphabet is not relevant, Σ and δ may be replaced by a transition relation $\rightarrowtail \subseteq Q \times Q$. A *run* in a Büchi automaton is an infinite sequence of states $q_0, q_1, \ldots$ such that $\exists_{p \in \Sigma}\ q_{i+1} \in \delta(q_i, p)$ for all $i \in \mathbb{N}$. Such a run is *accepting* if and only if it passes through an accepting state infinitely often, *i.e.*, $\{i \mid q_i \in F\}$ is infinitely large.

A *generalized Büchi automaton* is a Büchi automaton that has a set of acceptance sets $\mathcal{F}$ rather than a single acceptance set F. A run $q_0, q_1, \ldots$ in such an automaton is accepting if and only if it passes through a state in every acceptance set infinitely often, *i.e.*, for all $F \in \mathcal{F}$, $\{i \mid q_i \in F\}$ is infinitely large.

The *accepted language* of a (generalized) Büchi automaton is the set of all accepting runs in that automaton starting in a state from Q_0.

The *product* of a Kripke structure $M = \langle S, \rightarrow, AP, L\rangle$ and a Büchi automaton $\mathcal{A} = \langle Q, AP, \delta, Q_0, F\rangle$ with respect to starting state $s_0 \in S$ is defined as the Büchi automaton $\langle M, s_0\rangle \otimes \mathcal{A} = \langle Q', \rightarrowtail, Q_0', F'\rangle$, where

- $Q' = S \times Q$,
- $\langle s, q\rangle \rightarrowtail \langle s', q'\rangle$ if and only if $s \rightarrow s'$ and $q' \in \delta(q, L(s))$,
- $Q_0' = \{\langle s_0, q\rangle \in Q' \mid \exists_{q_0 \in Q_0} q \in \delta(q_0, L(s_0))\}$ and
- $F' = \{\langle s, q\rangle \in Q' \mid q \in F\}$.

Note that the alphabet of $\mathcal{A}$ is the set of atomic propositions from M.

2.4. Translation from LTL to Büchi automata

Below we sketch how to create a generalized Büchi automaton $\mathcal{A}$ for an LTL formula $\mathsf{A}f$ of which the accepted language consists of all sentences (paths) that satisfy $\neg f$. Checking that the accepted language of $\langle M, s_0\rangle \otimes \mathcal{A}$ is empty is then sufficient to conclude that there is no path in M starting in s_0 that satisfies $\neg f$, *i.e.*, $s_0 \models \mathsf{A}f$. The definitions below construct such a Büchi automaton for an LTL formula. These definitions are equivalent to those in [3], to which we refer for a full explanation.

If $\mathsf{A}f$ is an LTL formula, then we define the *closure* of f, denoted $\mathcal{C}(f)$, to be the set of all subformulae of f and their negation. Double negations are omitted, *i.e.*, formulae of the form $\neg\neg g$ are represented by g. For example, $\mathcal{C}(a \mathsf{U} \neg b)$ is defined to be the set $\{a, \neg a, \neg b, b, a \mathsf{U} \neg b, \neg(a \mathsf{U} \neg b)\}$.

An *LTL automaton* is a generalized Büchi automaton $\mathcal{A} = \langle Q, \Sigma, \delta, Q_0, \mathcal{F}\rangle$ corresponding to an LTL formula. The LTL automaton corresponding to $\mathsf{A}f$ is denoted $\mathcal{A}_f$. It is assumed that any atomic predicates in f belong to some set AP.

- Q is the largest subset of $2^{\mathcal{C}(f)}$ such that for all $B \in Q$ we have the following:
  - $g \notin B \Leftrightarrow \neg g \in B$
  - $g \wedge h \in B \Leftrightarrow g \in B$ and $h \in B$
  - if $g \mathsf{U} h \in \mathcal{C}(f)$, then
    - $h \in B \Rightarrow g \mathsf{U} h \in B$
    - $g \mathsf{U} h \in B$ and $h \notin B \Rightarrow g \in B$
- $\Sigma = 2^{AP}$
- $\delta(B, A) = B'$ if and only if
  - $A = B \cap AP$
  - For every $\mathsf{X} g \in \mathcal{C}(f)$: $\mathsf{X} g \in B \Leftrightarrow g \in B'$
  - For every $g \mathsf{U} h \in \mathcal{C}(f)$: $g \mathsf{U} h \in B \Leftrightarrow (h \in B \vee (g \in B \wedge g \mathsf{U} h \in B'))$
- $Q_0 = \{B \in Q \mid \neg f \in B\}$
- $\mathcal{F} = \{F_{g \mathsf{U} h} \mid g \mathsf{U} h \in \mathcal{C}(f)\}$, where $F_{g \mathsf{U} h} = \{B \in Q \mid g \mathsf{U} h \notin B \text{ or } h \in B\}$

An LTL automaton $\mathcal{A}^G = \langle Q^G, \Sigma^G, \delta^G, Q_0^G, \mathcal{F}\rangle$ can be transformed to a normal Büchi automaton by making a copy for every acceptance set and linking those copies together cyclically. This construction is also explained in [3]. We give a precise definition here. Suppose that $k = |\mathcal{F}|$ for some k and $f : \{0, \ldots, k-1\} \to \mathcal{F}$ enumerates $\mathcal{F}$ in an arbitrary way. A normal Büchi automaton that is equivalent to $\mathcal{A}^G$ is given by $\mathcal{A} = \langle Q, \Sigma, \delta, Q_0, F\rangle$, where

$$\begin{array}{ll} Q = Q^G \times \{0, \ldots, k-1\} & Q_0 = \{\langle q, 0\rangle \in Q \mid q \in Q_0^G\} \\ \Sigma = \Sigma^G & F = \{\langle q, i\rangle \in Q \mid q \in f(i)\} \end{array}$$

and where δ is defined as follows:

$$\langle q', j\rangle \in \delta(\langle q, i\rangle, p) \quad \text{iff} \quad q' \in \delta^G(q, p) \text{ and } \begin{cases} i = j, & q \notin f(i) \\ (i+1) \bmod k = j & q \in f(i) \end{cases}$$

The resulting Büchi automaton accepts the same language as $\mathcal{A}^G$.

3. The first-order modal μ-calculus

In this section we introduce a first-order extension of the modal μ-calculus. It is a propositional variant of the μ-calculus described in [12]. In this formalism, a notion of data is used, which we present first.

A data sort D is a set of atomic elements, associated with a semantic set $\mathbb{D}$. Operations on data sorts represent operations on their semantic sets and yield (closed) terms that represent elements from those sets. We assume the existence of an interpretation function $[\![\_]\!]$ that maps a closed term t of sort D to an element $[\![t]\!]$ of $\mathbb{D}$.

Throughout the paper, we assume that for a sort D there is an associated set of variables $\mathcal{D}$ of sort D. A data environment $\varepsilon : \mathcal{D} \to \mathbb{D}$ is used to map variable names to elements of $\mathbb{D}$. In the obvious way, $[\![\_]\!]$ is extended to open terms, and we denote the data element associated with an open term t given a data environment ε with $[\![t]\!]^\varepsilon$.

We write $\varepsilon[d \mapsto v]$ to denote a data environment ε' for which $\varepsilon'(d') = \varepsilon(d')$ for all $d' \neq d$ and $\varepsilon'(d) = v$.

In this paper we assume the existence of a data sort B representing the booleans $\mathbb{B}$ and a sort N representing the natural numbers $\mathbb{N}$.

3.1. Syntax and semantics

We assume the existence of sort D with variables $\mathcal{D}$ that corresponds to some semantic set $\mathbb{D}$ as explained earlier. The syntax of a μ-calculus formula is defined by the following grammar:

$$\varphi ::= b \mid p \mid X(e) \mid \neg\psi \mid \chi \wedge \psi \mid [\cdot]\psi \mid \mu X(d{:}D = e)\,.\,\psi \mid \forall_{d:D}\,\psi$$

In the above, χ and ψ are μ-calculus formulae, b is a boolean expression, p is an atomic proposition (sometimes called propositional constant), $d \in \mathcal{D}$ is a variable name, e is a data expression of sort D and X is a fixpoint variable taken from a set $\mathcal{X}$ of variable names.

The interpretation of a μ-calculus formula φ, denoted by $[\![\varphi]\!]^{\rho\varepsilon}$, is given in the context of a data environment $\varepsilon : D \to \mathbb{D}$, a predicate environment $\rho : \mathcal{X} \to (\mathbb{D} \to 2^S)$ and a Kripke structure $\langle S, \to, I, AP, L\rangle$.

$$\begin{aligned} [\![b]\!]^{\rho\varepsilon} &\triangleq \begin{cases} S, & [\![b]\!]^\varepsilon \\ \emptyset, & \text{otherwise} \end{cases} \\ [\![p]\!]^{\rho\varepsilon} &\triangleq \{s \in S \mid p \in L(s)\}, \quad p \in AP \\ [\![X(e)]\!]^{\rho\varepsilon} &\triangleq \rho(X)([\![e]\!]^\varepsilon) \\ [\![\neg\psi]\!]^{\rho\varepsilon} &\triangleq S \setminus [\![\psi]\!]^{\rho\varepsilon} \\ [\![\chi \wedge \psi]\!]^{\rho\varepsilon} &\triangleq [\![\chi]\!]^{\rho\varepsilon} \cap [\![\psi]\!]^{\rho\varepsilon} \\ [\![[\cdot]\psi]\!]^{\rho\varepsilon} &\triangleq \{s \in S \mid \forall_{s' \in S}(s \to s' \Rightarrow s' \in [\![\psi]\!]^{\rho\varepsilon})\} \\ [\![\forall_{d:D}\psi]\!]^{\rho\varepsilon} &\triangleq \bigcap_{v \in \mathbb{D}} [\![\psi]\!]^{\rho\varepsilon[d \mapsto v]} \\ [\![\mu X(d{:}D = e).\psi]\!]^{\rho\varepsilon} &\triangleq (\mu\Phi_d)([\![e]\!]^\varepsilon) \end{aligned}$$

where $\Phi_d : (D \to 2^S) \to (D \to 2^S)$ is given as

$$\Phi_d \triangleq \lambda F{:}D \to 2^S.\lambda v{:}\mathbb{D}.[\![\psi]\!]^{\rho[X \mapsto F]\varepsilon[d \mapsto v]}$$

We use the following standard abbreviations to denote some useful derived operators, where $\psi[\neg X/X]$ stands for the expression ψ in which every occurrence of X has been replaced by $\neg X$:

$$\begin{aligned} &\chi \vee \psi \triangleq \neg(\neg\chi \wedge \neg\psi) \\ &\langle\cdot\rangle\psi \triangleq \neg[\cdot]\neg\psi \\ &\exists_{d:D}\psi \triangleq \neg\forall_{d:D}\neg\psi \\ &\nu X(d{:}D = e)\,.\,\psi \triangleq \neg\mu X(d{:}D = e)\,.\,\neg\psi[\neg X/X] \end{aligned}$$

For readability, we allow fixpoint variables to be parameterised with multiple data parameters, separated by commas, rather than using a structured sort and projection functions. We also introduce one non-standard abbreviation. If P is a set of atomic propositions, then the μ-calculus formula P represents states that are labelled with exactly the labels in P:

$$P \triangleq \bigwedge_{a \in P} a \wedge \bigwedge_{a \in AP \setminus P} \neg a, \qquad P \subseteq AP$$

When a data domain Γ is used that consists of the single element γ (*i.e.*, when data is not used), the formula $\sigma X(d{:}\Gamma = \gamma) \,.\, \psi$ is abbreviated to $\sigma X \,.\, \psi$ (for $\sigma \in \{\mu, \nu\}$).

It is important to note that the least fixpoint of Φ_d does not always exist. However, if in the above definition, φ can be transformed to *positive normal form* [6], in which negation only occurs on the level of atomic propositions and in which all bound variables are distinct, then the existence of such a fixpoint is guaranteed. This claim is justified by the fact that we can define an ordering $\sqsubseteq$ on $D \to 2^S$ such that $f \sqsubseteq g$ if and only if for all $d{:}D, f(d) \subseteq g(d)$. Then $\langle D \to 2^S, \sqsubseteq \rangle$ is a complete lattice and because the functionals are monotonic over this lattice (see [12]), Tarski's theorem [20] can be applied to establish that the least fixpoint of Φ_d exists.

Furthermore, this fixpoint may be approximated by applying Φ a number of times to the infimum of the lattice (in case of a least fixpoint) or to the supremum of the lattice (for a greatest fixpoint).

The *size* of a formula φ, denoted $|\varphi|$, is the number of subformulae it contains. The size of data expressions is defined in the same manner.

Given a Kripke structure M with states S, and environments ρ and ε, we say that a formula φ holds in state $s \in S$, denoted $M, s \models_{\rho\varepsilon} \varphi$, if and only if $s \in [\![\varphi]\!]^{\rho\varepsilon}$ (where $[\![\varphi]\!]^{\rho\varepsilon}$ is interpreted on M). If $M, s \models_{\rho\varepsilon} \varphi$ for any ρ and ε (this holds for any closed formula), then we write $M, s \models \varphi$. If M is clear from the context, we omit it.

In the remainder of this paper we assume that $D = \mathbb{D}$ to make reasoning about the semantics of a formula less troublesome.

4. Translating LTL to the first-order μ-calculus

In this section we provide a translation of LTL to the first-order modal μ-calculus. We first remark that this translation is not straightforward, in the sense that a simple syntactic translation procedure has not been found. Consider the following two standard translations from LTL to the μ-calculus.

$$p \,\mathsf{U}\, q \overset{\text{trans}}{=} \mu X \,.\, (p \wedge [\cdot]X) \vee q \qquad p \,\mathsf{R}\, q \overset{\text{trans}}{=} \nu X \,.\, (p \vee [\cdot]X) \wedge q$$

The above translations appear often in literature, and at first sight seem very convenient. For example, it is easy to see that a translation for $\mathsf{AF}q = \textbf{true}\,\mathsf{U}\,q$ and $\mathsf{AG}q = \textbf{false}\,\mathsf{R}\,q$ can be obtained from the above by simply replacing p by **true** and **false** respectively, yielding $\mu X \,.\, [\cdot]X \vee q$ and $\nu X \,.\, [\cdot]X \wedge q$ respectively. However, $\mathsf{AFG}q$ cannot be obtained by the same simple syntactic replacing, as that would result in the formula $\mu X \,.\, [\cdot]X \vee \nu Y \,.\, [\cdot]Y \wedge q$, which expresses the CTL formula $\mathsf{AFAG}\ q$ (for a more detailed treatment on the expressive power of CTL and LTL, including an explanation of this specific case, see [17]). The following μ-calculus formula is the proper translation of $\mathsf{AFG}q$.

$$\mu X \,.\, \nu Y \,.\, [\cdot]X \vee (q \wedge [\cdot]Y) \tag{1}$$

Notice that this formula expresses the absence of an accepting path in a Büchi automaton if we label all non-accepting states of that automaton with q. Because a translation to Büchi automata is already known, it seems natural to use formula (1) as a framework for our translation.

We note that this formula can be replaced by the apparently stronger formula $\mu X \,.\, \nu Y \,.\, (\neg q \wedge [\cdot]X) \vee (q \wedge [\cdot]Y)$. This alteration does not change the meaning of the formula, because both fixpoints must identify the same set of nodes, and therefore in particular $[\cdot]Y \Rightarrow [\cdot]X$. We use a similar construction in this paper to make the complexity analysis easier, even though we do not need this alternative formulation for our proof of correctness.

The introduction of data allows us to formulate certain properties more concisely, by exploiting repetitive structures in the formula. Consider for instance the following formula.

$$\mu X \,.\, \langle\cdot\rangle X \vee (p(0) \wedge \mu Y \,.\, \langle\cdot\rangle Y \vee (p(1) \wedge \mu Z \,.\, \langle\cdot\rangle Z \vee p(2)))$$

This formula expresses that first a state in which $p(0)$ holds is reachable, then a state in which $p(1)$ holds and finally one in which $p(2)$ holds. This formula (and any extension thereof) can also be expressed as follows:

$$\mu X(i{:}N = 0) \,.\, \langle\cdot\rangle X(i) \vee (p(i) \wedge \langle\cdot\rangle X(i+1)) \vee i \approx 3$$

In the above, $i \approx 3$ has the standard arithmetic meaning of 'i equals 3'. Note that in the above formula the fixpoints have collapsed into a single one, and that the number of boolean operators has also diminished.

Another example is the following formula, which expresses that out of the first $2k$ states visited, k states must be labelled with p:

$$\begin{aligned} \mu X(n{:}N = 0, m{:}N = 0) \,.\, & (n \approx k \wedge m \approx k) \vee \\ & ([\cdot]X(n+1, m) \wedge p) \vee \\ & ([\cdot]X(n, m+1) \wedge \neg p) \end{aligned}$$

The size of this formula is $O(1)$, where the equivalent in the normal μ-calculus would take $O(2^k)$ space (or $O(k^2)$ in the equational μ-calculus).

We now return to the problem of translating LTL to the first-order μ-calculus. We base our translation on Büchi automaton representations of LTL formulae as referred to in Section 2.3. This representation is encoded into a data structure consisting of booleans and natural numbers. We express a μ-calculus property that in a sense 'synchronizes' steps in the Büchi automaton (utilizing the data structure) with steps that are made in the transition system against which the formula is checked. Keeping this synchrony intact, we can use the standard translation of AFGq to express that this Büchi automaton does not accept any path of the transition system.

Formally speaking, we assume that we are given some Büchi automaton $\mathcal{A}$, and construct a μ-calculus formula that accepts a state s_0 of the Kripke structure M it is interpreted on if and only if $\mathcal{L}(\langle M, s_0\rangle \otimes \mathcal{A}) = \emptyset$, *i.e.*, the accepted language of the product of the Kripke structure and the Büchi automaton is empty.

Definition 4.1 $(T, T', \mathbf{Tr})$**.** We define a translation function **Tr** that generates a μ-calculus formula from a Büchi automaton. Let $\mathcal{A} = \langle Q, \Sigma, \delta, Q_0, F\rangle$ be a Büchi automaton.

$$\mathbf{Tr}(\mathcal{A}) = \forall_{q\in Q, p\in\Sigma}\big((p \wedge \exists_{q_0\in Q_0} q \in \delta(q_0, p)) \Rightarrow T(q)\big)$$

with $T(q)$ defined as

$$\begin{aligned} T(q_0) &= \mu X(q'':Q = q_0)\,.\,T'(q'') \\ T'(q'') &= \nu Y(q:Q = q'')\,.\,\forall_{p,q':q'\in\delta(q,p)}[\cdot]\Big(p \Rightarrow \big((X(q') \wedge q \in F) \vee (Y(q') \wedge q \notin F)\big)\Big)\end{aligned}$$

In the above, note that the quantifier selects those q' and p that form a single step in the Büchi automaton from state q. The implication $(p \Rightarrow \ldots)$ ensures that the required property is only checked along paths realizing such steps. In this manner, the quantifier and implication realize the aforementioned synchrony. In effect the formula $\mu X\,.\,\nu Y\,.\,[\cdot]X \vee (q \notin F \wedge [\cdot]Y)$ is checked on the paths of the Büchi automaton that have a corresponding path in the Kripke structure (*i.e.*, exactly the paths in $\langle M, s_0\rangle \otimes \mathcal{A}$).

The semantics of $T(q_0)$ is $[\![T(q_0)]\!]^{\rho\varepsilon}$, which equals $(\mu\Phi)(q_0)$, where

$$\Phi = \lambda\hat{X}{:}Q \to 2^S.\lambda\hat{q}{:}Q\,.\,[\![T'(q'')]\!]^{\rho[X\mapsto\hat{X}]\varepsilon[q''\mapsto\hat{q}]}$$

As explained in Section 3, we can calculate this fixpoint by starting with an initial approximation $\hat{X}^0$ that is the minimal element of the lattice $\langle Q \to 2^S, \sqsubseteq\rangle$, and then choosing the next approximation $\hat{X}^{m+1} = \Phi(\hat{X}^m)$.

Concretely, these approximations for T are given as follows:

$$\begin{aligned} \hat{X}^0 &= \lambda q'':Q.\mathbf{false} \\ \hat{X}^{m+1} &= \lambda q'':Q.\nu Y(q:Q = q'')\,.\,\forall_{p,q':q'\in\delta(q,p)}[\cdot]\Big(p \Rightarrow \big((\hat{X}^m(q') \wedge q \in F) \vee (Y(q') \wedge q \notin F)\big)\Big)\end{aligned}$$

Because $\hat{X}^0 = \lambda\hat{q}{:}Q\,.\emptyset$ can be written as $\lambda\hat{q}{:}Q\,.\,[\![\mathbf{false}]\!]^{\rho[X\mapsto\hat{X}^{-1}]\varepsilon[q''\mapsto\hat{q}]}$ (however we wish to define $\hat{X}^{-1}$), every approximation $\hat{X}^{m+1}$ can be written as a function of the form $\lambda\hat{q}{:}Q\,.\,[\![\phi]\!]^{\rho[X\mapsto\hat{X}^m]\varepsilon[q''\mapsto\hat{q}]}$. To increase legibility, we omit the interpretation function and abbreviate this to $\lambda q'':Q\,.\,\phi[\hat{X}^m/X]$, where $\phi[\hat{X}^m/X]$ is the formula ϕ with all occurrences of X replaced by $\hat{X}^m$. Note that T is equal to $\hat{X}^\alpha$ for some sufficiently large α.

Similarly, we can approximate T', given an approximation $\hat{X}^m$ of T:

$$\begin{aligned} \hat{Y}^0_m &= \lambda q{:}Q.\mathbf{true} \\ \hat{Y}^{n+1}_m &= \lambda q{:}Q.\forall_{p,q':q'\in\delta(q,p)}[\cdot]\Big(p \Rightarrow \big((\hat{X}^m(q') \wedge q \in F) \vee (\hat{Y}^n_m(q') \wedge q \notin F)\big)\Big)\end{aligned}$$

Using these definitions, we show the relationship between the μ-calculus formula of Definition 4.1 and the Büchi automaton it was generated from.

Lemma 4.2 $(\Rightarrow)$**.** *Let $\mathcal{A} = \langle Q, \Sigma, \delta, Q_0, F\rangle$ be a Büchi automaton and let $M = \langle S, \to, AP, L\rangle$ be a Kripke structure. If $s_0 \in S$, $q_0 \in Q$ and $s_0 \models T(q_0)$, then there is no accepting run in $\langle M, s_0\rangle \otimes \mathcal{A}$ from state $\langle s_0, q_0\rangle$.*

Proof. We give a proof by contraposition. Suppose that there is an accepting run $\pi = \langle s_0, q_0\rangle, \langle s_1, q_1\rangle, \ldots$ in $\langle M, s_0\rangle \otimes \mathcal{A}$. We prove that $s_0 \not\models T(q_0)$ by showing that $\forall_{m\in\mathbb{N}}\forall_{i\in\mathbb{N}}\, s_i \not\models \hat{X}^m(q_i)$ by using induction on m. It then follows that $\forall_{i\in\mathbb{N}}\, s_i \not\models T(q_i)$. The induction hypothesis is the following.

$$\forall_{i\in\mathbb{N}}\, s_i \not\models \hat{X}^m(q_i) \tag{2}$$

For $m = 0$, this trivially holds. For $m > 0$ we have to show that the greatest fixpoint T' of Y does not contain any of these s_i either. We show that there is some n for which $\forall_{i\in\mathbb{N}}\, s_i \not\models \hat{Y}^n_m(q_i)$ and therefore $\forall_{i\in\mathbb{N}}\, s_i \not\models T'(q_i)$.

Observe that, because of the definition of the transition function of $\langle M, s_0\rangle \otimes \mathcal{A}$, $s_i \to s_{i+1}$ and $q_{i+1} \in \delta(q_i, L(s_{i+1}))$ for all $i \in \mathbb{N}$. The definition of $\hat{Y}_m^{n+1}$ therefore implies that if $s_i \models \hat{Y}_m^{n+1}(q_i)$, then we must also have $s_{i+1} \models L(s_{i+1}) \Rightarrow ((\hat{X}^m(q_{i+1}) \wedge q_i \in F) \vee (\hat{Y}_m^n(q_{i+1}) \wedge q_i \notin F))$. Because by definition $s_{i+1} \models L(s_{i+1})$, and because of (2), we have an even stronger implication:

$$\text{For } n \in \mathbb{N}, \text{ if } s_i \models \hat{Y}_m^{n+1}(q_i), \text{ then } s_{i+1} \models \hat{Y}_m^n(q_{i+1}) \text{ and } q_i \notin F \tag{3}$$

Suppose that $q_i \in F$ for some i, then $s_{i+1} \not\models \hat{Y}_m^n(q_{i+1}) \wedge (q_i \notin F)$ for any n, and it follows immediately that also $s_i \not\models \hat{Y}_m^1(q_i)$. Now suppose $q_i \notin F$. Because π is an accepting run, there must be some $k \in \mathbb{N}$ for which $q_{i+k} \in F$, in which case $s_{i+k} \not\models \hat{Y}_m^n(q_{i+k})$ for any n. In particular this holds for $n = 1$ and therefore we have, by transitivity of implication (3), $s_i \not\models \hat{Y}_m^{k+1}(q_i)$. □

Lemma 4.3 (⇐). *Let $\mathcal{A} = \langle Q, \Sigma, \delta, Q_0, F\rangle$ be a Büchi automaton and let $M = \langle S, \to, AP, L\rangle$ be a Kripke structure. If $s_0 \in S$, $q_0 \in Q$ and $s_0 \not\models T(q_0)$, then there is an accepting run in $\langle M, s_0\rangle \otimes \mathcal{A}$ from state $\langle s_0, q_0\rangle$.*

Proof. In this proof, s and s' are always taken from S, q and q' from Q and n, m and k from $\mathbb{N}$. Assume that $s \not\models T(q)$ for some s and q. Let $\rightarrowtail$ be the transition relation of $\langle M, s_0\rangle \otimes \mathcal{A}$. We first define a function $G : S \times Q \times \mathbb{N} \to \mathbb{B}$.

$$G(s, q, n) = \begin{cases} q \in F \wedge s \not\models T(q), & n = 0 \\ \exists_{s',q'}\ \langle s, q\rangle \rightarrowtail \langle s', q'\rangle \wedge G(s', q', n-1), & n > 0 \end{cases}$$

This function is true for those s, q and n for which there is a path of length n from $\langle s, q\rangle$ to an accepting state $\langle s', q'\rangle$ of $\langle M, s_0\rangle \otimes \mathcal{A}$ in which $T(q')$ does not hold. We will prove the following for all s and q.

$$s \not\models T(q) \Rightarrow \exists_k\, G(s, q, k) \wedge k > 0. \tag{4}$$

Observe that if this implication holds, we can construct an accepting path in $\langle M, s_0\rangle \otimes \mathcal{A}$ from any $\langle s, q\rangle$ for which $s \not\models T(q)$. In particular we can then do so from $\langle s_0, q_0\rangle$, thus proving our claim.

First note that $s \not\models T(q)$ is equivalent to $\forall_m \exists_n\, s \not\models \hat{Y}_m^n(q)$, because $T(q) = \hat{X}^\alpha(q) = \hat{Y}_\alpha^\beta(q)$ for sufficiently large α and β, and because $\hat{Y}_m^n(q) \supseteq \hat{Y}_m^{n+1}(q)$ and $\hat{X}^m(q) \subseteq \hat{X}^{m+1}(q)$.

Now take arbitrary m, n, s and q such that $s \not\models \hat{Y}_m^n(q)$. Filling in the definition of $\hat{Y}_m^n(q)$, this is equal to

$$s \not\models \forall_{p,q':q' \in \delta(q,p)}[\cdot]\left(p \Rightarrow \left((\hat{X}^m(q') \wedge q \in F) \vee (\hat{Y}_m^n(q') \wedge q \notin F)\right)\right)$$

This must mean that there are s', q' such that $\langle s, q\rangle \rightarrowtail \langle s', q'\rangle$ and

$$s' \models L(s') \wedge \neg\left((\hat{X}^m(q') \wedge q \in F) \vee (\hat{Y}_m^n(q') \wedge q \notin F)\right) \tag{5}$$

In particular, we have found

$$s \not\models \hat{Y}_m^n(q) \Rightarrow \exists_{s',q'}\ \langle s, q\rangle \rightarrowtail \langle s', q'\rangle \wedge (s' \not\models \hat{Y}_m^{n-1}(q') \vee s' \models q \in F) \tag{6}$$

Let M be a number so large that $\hat{X}^M(q) = T(q)$ for all q. We now show that

$$\forall_s \forall_q \left(\forall_m \exists_n\, s \not\models \hat{Y}_m^n(q)\right) \Rightarrow \exists_k\, G(s, q, k),$$

$$\text{or equivalently } \forall_s \forall_q \left(\exists_m \forall_n\, s \not\models \hat{Y}_m^n(q) \Rightarrow \exists_k\, G(s, q, k)\right)$$

$$\text{by showing } \forall_n \forall_s \forall_q\, s \not\models \hat{Y}_M^n(q) \Rightarrow \exists_k\, G(s, q, k)$$

The proof proceeds by induction on n. Suppose $n = 1$, then by (6) we find that there is some $s' \in S$ such that $s' \models q \in F$, *i.e.*, $q \in F$ must hold. If $s \not\models \hat{Y}_M^n(q)$, then also $s \not\models \hat{X}^M(q)$ and therefore $s \not\models T(q)$ and so $G(s, q, 0)$ holds.

If $n > 1$, then (6) yields s' and q' such that either $s' \models q \in F$, in which case we apply the same reasoning as before, or $s' \not\models \hat{Y}_M^{n-1}(q')$. Using the induction hypothesis, we find some k for which $G(s', q', k)$ holds, but then $G(s', q', k+1)$ must also hold.

Since $T(q) \Rightarrow \exists_n\, \hat{Y}_M^n(q)$, we can use the same proof for (4), except for the case that $n = 1$. In that case however, we had already concluded that $q \in F$, so by (5) there must be s', q' such that $s' \not\models \hat{X}^M$ and $\langle s, q\rangle \rightarrowtail \langle s', q'\rangle$. But then we can use the previous result to conclude $G(s', q', k)$ and therefore $G(s, q, k+1)$ for some k. □

Theorem 4.4. *Let $\mathcal{A} = \langle Q, \Sigma, \delta, Q_0, F\rangle$ be a Büchi automaton and let $M = \langle S, \to, AP, L\rangle$ be a Kripke structure. For any $s_0 \in S$ we have that $s_0 \models \mathbf{Tr}(\mathcal{A})$ if and only if $\mathcal{L}(\langle M, s_0\rangle \otimes \mathcal{A}) = \emptyset$.*

Proof. Note that $s_0 \models p \wedge \exists_{q_0 \in Q_0} q \in \delta(q_0, p)$ for exactly those $q \in Q$ for which $\langle s_0, q\rangle$ is in the set of initial states of $\langle M, s_0\rangle \otimes \mathcal{A}$. The language of $\langle M, s_0\rangle \otimes \mathcal{A}$ is empty if and only if there is no accepting run starting in any of these states. $\mathbf{Tr}(\mathcal{A})$ demands that in these states $T(q)$ must hold. Lemmas 4.2 and 4.3 show that $s_0 \models T(q)$ is true if and only if there is no accepting run in $\langle M, s_0\rangle \otimes \mathcal{A}$ starting in $\langle s_0, q\rangle$. □

The above proofs, together with the fact that every LTL formula can be represented by a Büchi automaton, lead to the conclusion that we can translate any LTL formula into an equivalent μ-calculus formula:

Corollary 4.5. *Let s be a state in a Kripke structure. If $\mathsf{A}\varphi$ is an LTL formula, then $s \models \mathsf{A}\varphi$ if and only if $s \models \mathbf{Tr}(\mathcal{A}_\varphi)$.*

4.1. Data specifications

We have formulated our translation in such a way that it uses a Büchi automaton directly (Q, δ, Q_0 and F occur in our formula). In order to use existing techniques [7] to be able to automatically check the μ-calculus formula against a Kripke structure, and also to exploit the structured manner in which we can build a Büchi automaton from an LTL formula, we encode Q into a datatype which consists of only booleans and natural numbers.

Let $\mathsf{A}f$ be an LTL formula consisting of subformulae Φ, and let $\mathcal{A}$ be a normal Büchi automaton constructed for $\mathsf{A}f$ as described in Section 2.4. Because of the way $\mathcal{A}$ was constructed, it may be described using only booleans (denoted by $\mathbb{B}$) and natural numbers (denoted by $\mathbb{N}$). Recall that a state in $\mathcal{A}$ is represented by an element from 2^{Φ} and a counter $c \in \{0, \ldots, k-1\}$, with k the number of until operators in f.

Now we use the fact that, given some mapping from Φ to $\{0, \ldots, |\Phi|\}$, we can substitute 2^{Φ} by the isomorphic domain $\mathbb{B}^{|\Phi|}$. The counter can be represented by a value from $\mathbb{N}$, and so we may represent states by an element from $\mathbb{B}^{|\Phi|} \times \mathbb{N}$.

If $q \in \mathbb{B}^{|\Phi|}$, then by $P(q)$ we denote the set of atomic propositions of which the corresponding bit in q is set, *i.e.*, if q represents a set $\Psi \in 2^{\Phi}$, then $P(q) = \Psi \cap AP$.

We proceed by giving an encoding of the Büchi automaton corresponding to an LTL formula $\mathsf{A}f$ over atomic propositions AP, consisting of arbitrarily ordered subformulae $g_0 \ldots g_n$ of f. We fix a datatype $D = B^n \times N$ and we define the following four mappings:

$$\mathsf{inQ}{:}D \to B \qquad \mathsf{inQ}_0{:}D \to B$$
$$\mathsf{inF}{:}D \to B \qquad \mathsf{trans}{:}D \times D \to B$$

Intuitively, these mappings represent predicates on states from the Büchi automaton. For instance, if a data element $d{:}D$ represents some state q in a Büchi automaton with states Q and acceptance set F, then $\mathsf{inF}(d)$ will have the same truth value as the predicate $q \in F$. The inQ mapping is needed to identify those elements in D that represent a valid state in the Büchi automaton (there are subsets of the closure of f that are not in Q, see Section 2.4).

We now give the definitions of these mappings. Let $\mathfrak{U}$ be a set of indices, and let L and R be mappings from indices to indices. We have $i \in \mathfrak{U}$, $L(i) = j$ and $R(i) = k$ for some i, j and k if and only if $g_i = g_j \,\mathsf{U}\, g_k$. Let $U : \mathbb{N} \to \mathfrak{U}$ enumerate $\mathfrak{U}$ in an arbitrary way.

Similarly, let $\mathfrak{X}$ be another set of indices, such that $i \in \mathfrak{X}$ and $R(i) = j$ if and only if $g_i = \mathsf{X}g_j$ for some i and j.

$$\begin{aligned}
\mathsf{inQ}(\langle b_0, \ldots, b_n, c\rangle) &= \textstyle\bigwedge_{i\in\mathfrak{U}}(b_{R(i)} \Rightarrow b_i) \wedge (b_i \Rightarrow (b_{L(i)} \vee b_{R(i)}))\\
\mathsf{inQ}_0(\langle b_0, \ldots, b_n, c\rangle) &= \mathsf{inQ}(\langle b_0, \ldots, b_n, c\rangle) \wedge b_0\\
\mathsf{inF}(\langle b_0, \ldots, b_n, c\rangle) &= \mathsf{inQ}(\langle b_0, \ldots, b_n, c\rangle) \wedge (\neg b_{U(c)} \vee b_{R(U(c))})\\
\mathsf{trans}(b_0, \ldots, b_n, c\rangle, \ \langle b'_0, \ldots, b'_n, c'\rangle) &= \textstyle\bigwedge_{i\in\mathfrak{X}}(b_i \Leftrightarrow b'_{R(i)}) \wedge\\
&\quad \textstyle\bigwedge_{i\in\mathfrak{U}}(b_i \Leftrightarrow (b_{R(i)} \vee (b_{L(i)} \wedge b'_i))) \wedge\\
&\quad \mathsf{inF}(\langle b_0, \ldots, b_n, c\rangle) \Leftrightarrow (c' = (c+1) \bmod |\mathfrak{U}|)
\end{aligned}$$

Clearly, this specification is linear in the number of subformulae of f.[1]

We use the fact that $q' \in \delta(\langle b_0, \ldots, b_n, c\rangle, p)$ implies $\mathsf{trans}(\langle b_0, \ldots, b_n, c\rangle, q')$ and $p = P(\langle b_0, \ldots, b_n, c\rangle) = \{a \in AP \mid b_{I(a)}\}$, where I maps a subformula ψ_i to its index i. The μ-calculus formula in Definition 4.1 can be rewritten to the following formula using only quantifiers over D and using the previously defined mappings (*i.e.*, $q \in F$ is replaced by $\mathsf{inF}(q)$, $q' \in \delta(q, p)$ by $\mathsf{trans}(\langle b_0, \ldots, b_n, c\rangle, q')$ while replacing all occurrences of p by $P(q)$, etc.).

$$\begin{aligned}
&\forall_{q_0\in D}\Big(\mathsf{inQ}(q_0) \wedge \exists_{q'_0\in D}(\mathsf{inQ}_0(q'_0) \wedge P(q'_0) \wedge \mathsf{trans}(q'_0, q_0))\Big) \Rightarrow\\
&\mu X(q'' : D = q_0)\,.\,\nu Y(q : D = q'')\,.\\
&\quad \forall_{q'\in D}(\mathsf{inQ}(q') \wedge \mathsf{trans}(q, q')) \Rightarrow [\cdot]\Big(P(q) \Rightarrow \big((X(q') \wedge \mathsf{inF}(q)) \ \vee (Y(q') \wedge \neg\mathsf{inF}(q))\big)\Big)
\end{aligned}$$

The formula may grow to a size linear in $|f|$ due to the expansion of $P(q)$ to $\bigwedge_{a\in AP}(b_{I(a)} \Leftrightarrow a)$.

4.2. Complexity

We have given a translation from an LTL formula to a first-order μ-calculus formula over a data structure. We now show that model checking the resulting formula against a Kripke structure has the same time complexity as other LTL model checking methods. In particular, we establish the same worst-case time complexity as Bhat et al. [5].

Theorem 4.6. *Let ψ be the μ-calculus formula that is the result of the above translation for an LTL formula $\mathsf{A}f$. Verifying ψ on a Kripke structure M can be done in $O(|M|) \cdot 2^{O(|f|)}$ time.*

[1] Note that our definition of size does not take into account the space needed to represent an identifier, as for all practical intents and purposes it can be seen as a constant.

Proof. Using the Bekič principle [4] and the fact that we can transform a μ-calculus formula into an equational equivalent [2], we transform the first-order modal μ-calculus formula into the equational modal μ-calculus. Because D is a finite data type, we can transform – in linear time – the formula to the following system, where $N = |D|$ and $h : \{1, \ldots, N\} \to D$ enumerates D. The inverse mapping is denoted by h^{-1}.

$$\begin{aligned}
\mu X_\psi &= \bigwedge_{q_0 \in D} \left(\mathsf{inQ}(q_0) \wedge \bigvee_{q_0' \in D} (\mathsf{inQ}_0(q_0') \wedge P(q_0') \wedge \mathsf{trans}(q_0', q_0)) \right) \Rightarrow X_{h^{-1}(q_0)} \\
\mu X_0 &= Y_0 \\
&\vdots \\
\mu X_N &= Y_N \\
\nu Y_0 &= \bigwedge_{q' \in D} (\mathsf{inQ}(q') \wedge \mathsf{trans}(h(0), q')) \Rightarrow [\cdot] \left(P(h(0)) \Rightarrow (X_{h^{-1}(q')} \wedge \mathsf{inF}(h(0))) \vee (Y_{h^{-1}(q')} \wedge \neg \mathsf{inF}(h(0))) \right) \\
&\vdots \\
\nu Y_N &= \bigwedge_{q' \in D} (\mathsf{inQ}(q') \wedge \mathsf{trans}(h(N), q')) \Rightarrow [\cdot] \left(P(h(N)) \Rightarrow (X_{h^{-1}(q')} \wedge \mathsf{inF}(h(N))) \vee (Y_{h^{-1}(q')} \wedge \neg \mathsf{inF}(h(N))) \right)
\end{aligned}$$

The structure of the above expression becomes more apparent after computing the truth values of all expressions that are only dependent on data terms. This computation takes $O(|D|^2 \cdot \log |D|)$ time, as $\mathsf{trans}(q, q')$ has to be calculated for every pair q, q' and every such calculation costs $\log |D|$ time. Note that because $|D| = 2^{|f|}$, the time complexity for this computation is also $2^{O(|f|)}$. After computation, the system can be written as follows, where sets $S, S', S_0, \ldots, S_N$ contain indices between 0 and N for which certain data expressions evaluated to **true**.

$$\begin{aligned}
\mu X_\psi &= \bigwedge_{i \in S} \left(\bigvee_{j \in S'} P(h(j)) \right) \Rightarrow X_i \\
\mu X_0 &= Y_0 \\
&\vdots \\
\mu X_N &= Y_N \\
\nu Y_0 &= \bigwedge_{i \in S_0} [\cdot] \left(P(h(0)) \Rightarrow R_i \right), \quad R \in \{X, Y\} \\
&\vdots \\
\nu Y_N &= \bigwedge_{i \in S_N} [\cdot] \left(P(h(N)) \Rightarrow R_i \right), \quad R \in \{X, Y\}
\end{aligned}$$

From this system it is easy to see that when it is checked against a Kripke structure M, the disjuncts (including implications) disappear, as all $P(\ldots)$ terms are substituted by a truth value. Furthermore, the $[\cdot]$ operators change into conjuncts.

The solution of the resulting system $\mathcal{E}$ can therefore be found as the solution of a conjunctive boolean equation system [11]. Such a solution can be found in $O(|\mathcal{E}| \cdot |M|)$ time. The complexity of checking an LTL formula $\mathsf{A}f$ through the first-order modal μ-calculus is therefore $O(|M|) \cdot 2^{O(|f|)}$, as the size of D is exponential in the size of the LTL formula. □

5. Translation of CTL*

Assume that **Tr** generates data structures with fresh names every time it is used. We define a translation **Tr**′ that translates a CTL* formula into a modal μ-calculus formula. The intuition is that every CTL* formula can be seen as a CTL structure containing linear time fragments. Nested linear time fragments form a problem, because we cannot use the LTL translation on them directly. Instead, we take the innermost fragment (which must be LTL), and translate that using our translation function. We then replace this fragment in the original by a placeholder and repeat the procedure. In the translated fragments, the placeholders are again substituted for the translated counterparts of the linear time fragment they represent.

Definition 5.1 (*Equivalence*)**.** For two formulae (either μ-calculus or CTL*) φ and ψ, we say that φ is equivalent to ψ, denoted $\varphi \approx \psi$, if and only if for every Kripke structure M with states S we have $\forall_{s \in S}(M, s \models \varphi) \Leftrightarrow (M, s \models \psi)$.

If φ and ψ are both μ-calculus formula, then we define, given a predicate environment ρ and a data environment ε, $\varphi \approx_{\rho\varepsilon} \psi$ if and only if for every Kripke structure M with states S we have $\forall_{s \in S}(M, s \models_{\rho\varepsilon} \varphi) \Leftrightarrow (M, s \models_{\rho\varepsilon} \psi)$.

We introduce a set AP' that is disjunct from AP, which contains for every CTL* formula f an atomic proposition a_f. A function R is defined that takes a CTL* path formula, and returns a CTL* formula in which all top level Ag subformulae are replaced by a_g:

$$\begin{array}{llll} R(\mathsf{A}f) & \triangleq a_f & R(\mathsf{X}f) & \triangleq \mathsf{X}R(f) \\ R(f \mathsf{U} g) & \triangleq R(f) \mathsf{U} R(g) & R(f \vee g) & \triangleq R(f) \vee R(g) \\ R(\neg f) & \triangleq \neg R(f) & R(a) & \triangleq a \end{array}$$

The $\bar{R}$ function takes a μ-calculus formula and syntactically replaces every $a_g \in AP'$ that occurs in it by $\mathbf{Tr}'(g)$. What is left is to define $\mathbf{Tr}'$, which takes a CTL* formula and yields a μ-calculus formula:

$$\begin{array}{rcl} \mathbf{Tr}'(\mathsf{A}f) & = & \begin{cases} \mathbf{Tr}(\mathcal{A}_f), & \neg\exists_g\, \mathsf{A}g \in f \\ \bar{R}(\mathbf{Tr}'(\mathsf{A}R(f))), & \text{otherwise} \end{cases} \\ \mathbf{Tr}'(f \vee g) & = & \mathbf{Tr}'(f) \vee \mathbf{Tr}'(g) \\ \mathbf{Tr}'(\neg f) & = & \neg \mathbf{Tr}'(f) \\ \mathbf{Tr}'(a) & = & a \end{array}$$

It is immediately apparent from this definition that the size of the resulting formula is again linear in the size of the original.

Lemma 5.2. *Let φ, χ and ψ be a first-order modal μ-calculus formulae, and let $\varphi[\chi/\psi]$ denote φ in which all occurrences of ψ are syntactically replaced by χ. If, for some predicate environment ρ and data environment ε, $\chi \approx_{\rho\varepsilon} \psi$, then $\varphi \approx_{\rho\varepsilon} \varphi[\chi/\psi]$, provided that χ nor ψ contain free variable names (either fixpoint or data) that are bound in φ.*

Proof. Fix formulae φ, χ and ψ. The proof goes by structural induction on φ. The induction hypothesis states that if $\varphi' \in \varphi$ and $\varphi' \neq \varphi$, and also $\chi \approx_{\rho\varepsilon} \psi$ for some ρ and ε, then $\varphi'[\chi/\psi] \approx_{\rho\varepsilon} \varphi'$.

The base cases are trivial: either ψ does not occur in φ, in which case $\varphi[\chi/\psi] = \varphi$, or $\varphi = \psi$.

The other cases are also very straightforward. We demonstrate the case that $\varphi = \mu X(d{:}D = e)\,.\,\varphi'$. Suppose $\chi \approx_{\rho\varepsilon} \psi$ for some ρ and ε. Then $[\![\varphi[\chi/\psi]]\!]^{\rho\varepsilon} = (\mu\Phi_d)([\![e]\!]^{\varepsilon})$, where

$$\Phi_d \triangleq \lambda F{:}D \to 2^S.\lambda v{:}\mathbb{D}.[\![\varphi'[\chi/\psi]]\!]^{\rho[X\mapsto F]\varepsilon[d\mapsto v]}.$$

Because X and d do not occur freely in χ and ψ,

$$[\![\chi]\!]^{\rho[X\mapsto F]\varepsilon[d\mapsto v]} = [\![\chi]\!]^{\rho\varepsilon} = [\![\psi]\!]^{\rho\varepsilon} = [\![\psi]\!]^{\rho[X\mapsto F]\varepsilon[d\mapsto v]}.$$

We may substitute $[\![\varphi']\!]^{\rho[X\mapsto F]\varepsilon[d\mapsto v]}$ for $[\![\varphi'[\chi/\psi]]\!]^{\rho[X\mapsto F]\varepsilon[d\mapsto v]}$ in the above using the induction hypothesis. But then $(\mu\Phi_d)([\![e]\!]^{\varepsilon}) = [\![\varphi]\!]^{\rho\varepsilon}$, which implies that $\varphi \approx_{\rho\epsilon} \varphi[\chi/\psi]$. □

Lemma 5.3. *Let f be a CTL* formula. If $g \in f$, and $h \approx g$, then $f \approx f[h/g]$, where $f[h/g]$ is f in which all occurrences of g are syntactically replaced by h.*

Proof. The proof is again by induction on the structure of the formula. □

Theorem 5.4. *If f is a CTL* formula, then $f \approx \mathbf{Tr}'(f)$.*

Proof. Let $M = \langle S, \to, AP, L\rangle$. Our goal is to prove that $M \models f$ if and only if $M \models \mathbf{Tr}'(f)$. We introduce a Kripke structure $M' = \langle S, \to, AP \cup AP', L'\rangle$, where L' is such that for all $s \in S$, $a \in AP$ and $a_f \in AP'$ we have $a \in L'(s) \iff a \in L(s) \wedge a_f \in L'(s) \iff s \models \mathsf{A}f$. In words, M' is the extension of M such that states in which a CTL* formula Af holds are labelled with a_f. Note that this extension is conservative in the sense that for all $s \in S$ and CTL* formulae f over AP we have $M, s \models f$ iff $M', s \models f$.

We prove for all $s \in S$ that $M', s \models f$ if and only if $M', s \models \mathbf{Tr}'(f)$ by induction on the structure of f. There are two base cases:

$f \in AP$. It follows trivially from the semantics of CTL* and the modal μ-calculus that $f \approx \mathbf{Tr}'(f)$ in this case.

$f = \mathsf{A}g$ *and* $\neg\exists_h \mathsf{A}h \in g$. In this case, f is an LTL formula and is translated using **Tr**, which was proven to satisfy the desired property in the previous section.

The base of our induction is therefore sound. For the inductive step, we distinguish three cases.

$f = \neg g$. It follows directly from the semantics of CTL* and the semantics of the modal μ-calculus that if $g \approx \mathbf{Tr}'(g)$, then also $\neg g \approx \neg\mathbf{Tr}'(g)$.

$f = g \vee h$. Again $f \approx \mathbf{Tr}'(f)$ follows directly from the semantics of CTL* and of the modal μ-calculus.

$f = \mathsf{A}g$ *and* $\exists_h \mathsf{A}h \in g$. Note that $\mathsf{A}g \approx \mathsf{A}R(g)$, and $\mathsf{A}R(g)$ contains only a single A operator. Then $\mathbf{Tr}'(\mathsf{A}R(g)) = \mathbf{Tr}(\mathcal{A}_{R(g)})$, so we know from the previous section that $\mathbf{Tr}'(\mathsf{A}R(g)) \approx \mathsf{A}R(g)$. But then also $\mathbf{Tr}'(\mathsf{A}R(g)) \approx \mathsf{A}g$, by Lemma 5.3 and transitivity of $\approx$. This translation, $\mathbf{Tr}'(\mathsf{A}R(g))$, may contain some atomic predicate a_h that we introduced for a subformula Ah of g. By the induction hypothesis, we have that $\mathbf{Tr}'(\mathsf{A}h) \approx \mathsf{A}h$, and because by definition $a_h \approx \mathsf{A}h$, we also have $a_h \approx \mathbf{Tr}'(\mathsf{A}h)$. We can by Lemma 5.2 syntactically replace every a_h by $\mathbf{Tr}'(\mathsf{A}h)$ by applying $\bar{R}$, and obtain an equivalent formula. It follows that $\bar{R}(\mathbf{Tr}'(\mathsf{A}R(g))) \approx f$.

We now know that $M', s \models f$ if and only if $M', s \models \mathbf{Tr}'(f)$. However, f nor $\mathbf{Tr}'(f)$ contain atomic predicates from AP', and therefore both would be evaluated the same on M. Therefore also $M, s \models f$ if and only if $M, s \models \mathbf{Tr}'(f)$, which concludes our proof. □

We note that the above translation may be inefficient in practice, as it does not take advantage of the fact that for CTL formulae there is a much more straightforward translation to the modal μ-calculus. Without any change, $\mathbf{Tr}'$ may therefore generate translations for CTL formulae that require exponential time to solve. However, it is easy to adapt $\mathbf{Tr}'$ to include case distinctions for CTL operators, following the translation in, *e.g.*, [18]. The adapted translation procedure then yields translations for CTL formulae that can be solved in linear time, and will provide more efficient translations for certain types of CTL* formulae.

Theorem 5.5. *The complexity of checking a CTL* formula f through the first-order modal μ-calculus is $O(|M|) \cdot 2^{O(|f|)}$.*

Proof. This can be seen by looking at the structure of $\mathbf{Tr}'(f)$. Because of the way it was constructed, every innermost subformula $\varphi \in \mathbf{Tr}'(f)$ that was generated by **Tr**, is again a closed μ-calculus formula. We can introduce a fresh fixpoint variable X, and evaluate $[\![\mathbf{Tr}'(f)]\!]^{\rho[X \mapsto F]\varepsilon}$ for some ρ and ε, where $F = [\![\varphi]\!]^{\rho\varepsilon}$. As f is closed, this yields the same result as evaluating the expression using any other predicate environment. Note that we can calculate $[\![\varphi]\!]^{\rho\varepsilon}$ in $O(|M|) \cdot 2^{O(|g|)}$ time, where g is the subformula of f to which φ corresponds. By Lemma 5.2, we may substitute X for φ and repeat this procedure. When no more substitutions can be made, the remainder can be solved in $O(|M|)$ time, as the formula contains no fixpoints. Since the sum of the lengths of all g that were substituted is less than or equal to $|f|$, the entire operation is $O(|M|) \cdot 2^{O(|f|)}$. □

6. Conclusion

In this paper we presented a translation from CTL* formulae to first-order modal μ-calculus formulae. By using this specific variant of the μ-calculus, we are able to give a translation that is succinct, but that does not introduce performance penalties when checking the formula against a Kripke structure. Indeed, the time complexity of CTL* model checking via the first-order modal μ-calculus is no worse than that of CTL* model checking using the best existing direct method.

Acknowledgement

We would like to thank Tim Willemse for many valuable comments and discussions.

References

[1] mCRL2 web site, http://www.mcrl2.org.
[2] A. Arnold, D. Niwiński, Rudiments of μ-Calculus, in: Studies in Logic and the Foundations of Mathematics, vol. 146, North-Holland, 2001.
[3] C. Baier, J.-P. Katoen, Principles of Model Checking, The MIT Press, 2008.
[4] H. Bekič, Definable operation in general algebras, and the theory of automata and flowcharts, in: Programming Languages and Their Definition, in: LNCS, vol. 177, Springer, 1984, pp. 30–55.
[5] G. Bhat, R. Cleaveland, Efficient model checking via the equational μ-calculus, in: Logic in Computer Science, LICS'96, IEEE Computer Society, 1996, pp. 304–312.
[6] J. Bradfield, C. Stirling, Modal μ-calculi, in: Handbook of Modal Logic, 2006, pp. 721–756.
[7] A. van Dam, B. Ploeger, T.A.C. Willemse, Instantiation for parameterised boolean equation systems, in: Theoretical Aspects of Computing, ICTAC 2008, in: LNCS, vol. 5160, Springer, 2008, pp. 440–454.
[8] M. Dam, CTL* and ECTL* as fragments of the modal μ-calculus, in: 17th Colloquium on Trees in Algebra and Programming, CAAP'92, in: LNCS, vol. 581, Springer, 1992, pp. 145–164.
[9] E. Allen Emerson, Model checking and the mu-calculus, in: DIMACS Series in Discrete Mathematics, American Mathematical Society, 1997, pp. 185–214.
[10] E.A. Emerson, C.L. Lei, Efficient model checking in fragments of the propositional mu-calculus, in: Logic in Computer Science, LICS'86, IEEE Computer Society Press, 1986, pp. 267–278.
[11] J.F. Groote, M. Keinänen, A sub-quadratic algorithm for conjunctive and disjunctive boolean equation systems, in: Theoretical Aspects of Computing, ICTAC 2005, in: LNCS, vol. 3722, Springer, 2005, pp. 532–545.
[12] J.F. Groote, R. Mateescu, Verification of temporal properties of processes in a setting with data, in: Algebraic Methodology and Software Technology, in: LNCS, vol. 1548, Springer, 1998, pp. 74–90.
[13] J.F. Groote, A.H.J. Mathijssen, M.A. Reniers, Y.S. Usenko, M.J. van Weerdenburg, Analysis of distributed systems with mCRL2, in: M. Alexander, W. Gardner (Eds.), Process Algebra for Parallel and Distributed Processing, Chapman Hall, 2009, pp. 99–128.
[14] J.F. Groote, T.A.C. Willemse, Model-checking processes with data, Science of Computer Programming 56 (3) (2005) 251–273.
[15] J.F. Groote, T.A.C. Willemse, Parameterised boolean equation systems, Theoretical Computer Science 343 (3) (2005) 332–369.
[16] M. Hennessy, R. Milner, On observing nondeterminism and concurrency, in: Automata, Languages and Programming, in: LNCS, vol. 85, Springer, 1980, pp. 299–309.
[17] M. Huth, M. Ryan, Logic in Computer Science, Cambridge University Press, Cambridge, 2004.
[18] Edmund M. Clarke Jr., Orna Grumberg, Doron A. Peled, Model Checking, The MIT Press, 1999.
[19] D. Kozen, Results on the propositional μ-calculus, in: Automata, Languages and Programming, in: LNCS, vol. 140, 1982, pp. 348–359.
[20] A. Tarski, A lattice-theoretical fixpoint theorem and its applications, Pacific journal of Mathematics 5 (2) (1955) 285–309.