5th Asian Computing Science Conference, ASIAN '99
Proceedings: P.S. Thiagarajan and R. Yap (eds.)
Springer Lecture Notes in Computer Science 1742 (1999), 227–238.

Faster Model Checking for Open Systems

Madhavan Mukund[1], K. Narayan Kumar[1], and
Scott A. Smolka[2]

[1] Chennai Mathematical Institute, 92 G.N. Chetty Road, Chennai 600 017, India.
E-mail: {madhavan,kumar}@smi.ernet.in
[2] Department of Computer Science, State University of New York at Stony Brook,
NY 11794-4400, USA. E-mail: sas@cs.sunysb.edu.

Abstract. We investigate $\text{O}\scriptstyle{\text{R}}_{\text{E}}\text{X}$, a temporal logic for specifying open systems. Path properties in $\text{O}\scriptstyle{\text{R}}_{\text{E}}\text{X}$ are expressed using ω-regular expressions, while similar logics for open systems, such as ATL* of Alur et al., use LTL for this purpose. Our results indicate that this distinction is an important one. In particular, we show that $\text{O}\scriptstyle{\text{R}}_{\text{E}}\text{X}$ has a more efficient model-checking procedure than ATL*, even though it is strictly more expressive. To this end, we present a single-exponential model-checking algorithm for $\text{O}\scriptstyle{\text{R}}_{\text{E}}\text{X}$; the model-checking problem for ATL* in contrast is provably double-exponential.

1 Introduction

Reactive systems are computing systems where the computation consists of an ongoing interaction between components. This is in contrast to functional systems whose main aim is to compute sets of output values from sets of input values. Typical examples of reactive systems include schedulers, resource allocators, and process controllers.

Pnueli [Pnu76] proposed the use of linear-time temporal logic (LTL) to specify properties of reactive systems. A system satisfies an LTL specification if all computations of the system are models of the formula. Later, branching-time temporal logics such as CTL and CTL* [CE81,EH86] were developed, which permit explicit quantification over computations of a system. For both linear-time and branching-time logics, the *model checking* problem—verifying whether a given system satisfies a correctness property specified as a temporal logic formula—has been well studied. Model checkers such as SPIN for LTL [Hol97] and SMV for CTL [McM93] are gaining acceptance as debugging tools for industrial design of reactive systems.

Alur et al. [AHK97] have argued that linear-time and branching-time temporal logics may not accurately capture the spirit of reactive systems. These logics treat the entire system as a *closed* unit. When specifying properties of reactive systems, however, it is more natural to describe how different parts of the system behave when placed in *contexts*. In other words, one needs to specify properties of *open* systems that interact with environments that cannot be controlled.

A fruitful way of modelling open systems is to regard the interaction between the system and its environment as a game. The system's goal is to move in such a way that no matter what the environment does, the resulting computation satisfies some desired property. In this formulation, the verification problem reduces to checking whether the system has a winning strategy for such a game.

The *alternating temporal logic* ATL* proposed in [AHK97] reflects the game-theoretic formulation of open systems. In ATL*, the property to be satisfied in each run of the system is described using an LTL formula, and LTL formulas may be nested within game quantifiers. The quantifiers describe the existence of winning strategies for the system. Moreover, a restricted version of ATL* called ATL has an efficient model-checking algorithm. However, the model-checking problem for the considerably more expressive logic ATL* is shown to be 2EXPTIME-complete which compares unfavourably with the PSPACE-completeness of CTL*.

In this paper, we propose a logic for open systems called $\text{O}\scriptstyle{\text{R}}_{\text{E}}\text{X}$, which is interpreted over Σ-labelled transition systems and uses ω-regular expressions rather than LTL formulas to specify properties along individual runs. As in ATL*, ω-regular expressions may be nested within game quantifiers. Since ω-regular expressions are more expressive than temporal logics for specifying properties of infinite sequences, our logic is more expressive than ATL*. Moreover, it turns out that $\text{O}\scriptstyle{\text{R}}_{\text{E}}\text{X}$ has an exponential-time model checking algorithm, which is considerably better than the complexity of ATL*.

The paper is organized as follows. Section 2 contains some basic definitions about transition systems and games. Section 3 introduces the syntax and semantics of $\text{O}\scriptstyle{\text{R}}_{\text{E}}\text{X}$ and gives some examples of how to write temporal specifications in the logic. Section 4 describes an automata-theoretic algorithm for $\text{O}\scriptstyle{\text{R}}_{\text{E}}\text{X}$'s model-checking problem. Section 5 discusses related work and Section 6 offers our concluding remarks.

2 Transition Systems, Games, Strategies

Transition systems A *Σ-labelled transition system* is a tuple $TS = \langle X, \Sigma, \longrightarrow\rangle$ where X is a finite set of *states*, Σ is a finite alphabet of *actions* and $\longrightarrow \subseteq X \times \Sigma \times X$ is the *transition relation*. We use $x, y, \ldots$ to denote states and $a, b, \ldots$ to denote actions.

An element $(x, a, y) \in \longrightarrow$ is called an a-transition. For $S \subseteq \Sigma$, the set of S-transitions is given by $\longrightarrow_S = \{(x, a, y) \mid a \in S\}$. As usual, we write $x \xrightarrow{a} y$ to denote that $(x, a, y) \in \longrightarrow$. The set of transitions enabled at a state x, denoted *enabled*(x), is the set of transitions of the form $x \xrightarrow{a} y$ that originate at x. To simplify the presentation, we assume that there are no deadlocked states—that is, for each $x \in X$, *enabled*$(x) \neq \emptyset$.

A path in TS is a sequence $x_0 \xrightarrow{a_1} x_1 \xrightarrow{a_2} \cdots x_i \xrightarrow{a_{i+1}} x_{i+1} \cdots$. Let *paths*$(TS)$ and *finite_paths*(TS) denote the set of infinite and finite paths in TS, respectively. For a finite path p, let *last*(p) denote the final state of p.

Games and strategies Let $S \subseteq \Sigma$. An *S-strategy* is a function f from *finite_paths*(TS) to the set of S-transitions $\longrightarrow_S$ such that for each $p \in$ *finite_paths*(TS), $f(p) \subseteq$ *enabled*(*last*(p)). If $\longrightarrow_S \cap$ *enabled*(*last*(p)) $\neq \emptyset$, we require that $f(p) \neq \emptyset$.

An infinite path $\rho = x_0 \xrightarrow{a_1} x_1 \xrightarrow{a_2} \cdots x_n \xrightarrow{a_n} \cdots$ in *paths*(TS) is said to be *consistent with the S-strategy f* if for each $n \geq 1$ the transition $x_{n-1} \xrightarrow{a_n} x_n$ belongs to the set $f(x_0 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} x_{n-1})$ whenever $a_n \in S$.

A *game* over TS is a pair (Π, x) where Π is a subset of *paths*(TS) and x is a state in X. We say that S has a *winning strategy* for the game (Π, x) if there is an S-strategy f such that every infinite path that begins at x and is consistent with f belongs to Π.

Often, we are interested in restricting our attention to fair computations. An infinite path $\rho \in$ *paths*(TS) is said to be *weakly S-fair* if there are infinitely many states along ρ where no S-transition is enabled or if there are infinitely many S-transitions in ρ. The path is said to be *strongly S-fair* if there are only finitely many positions where an S-transition is enabled or if there are infinitely many S-transitions in ρ.

We can relativize our notion of winning strategies to fair paths in the obvious way—S has a weakly (strongly) fair winning strategy for (Π, x) if there is an S-strategy f such that every weakly (strongly) S-fair path that begins at x and is consistent with f belongs to Π.

3 The Logic

3.1 Syntax

Fix an alphabet Σ and a set of propositions *Prop*. We simultaneously define the sets *FPE* of *finite path expressions*, *PE* of *(infinite) path expressions*, and Φ of *state formulas* as follows. We use e to denote a typical element of *FPE*, α to denote a typical element of *PE*, and φ to denote a typical element of Φ.

$$\begin{aligned} e &::= \varphi \in \Phi \mid a \in \Sigma \mid e \cdot e \mid e + e \mid e^* \\ \alpha &::= e \cdot e^\omega \mid \alpha + \alpha \\ \varphi &::= \mathtt{tt} \mid P \in \mathit{Prop} \mid \neg\varphi \mid \varphi \vee \varphi \mid E_S\alpha,\ (S \subseteq \Sigma) \mid A_S\alpha,\ (S \subseteq \Sigma) \end{aligned}$$

Observe that finite path expressions are just regular expressions whose alphabet may include state formulas. Similarly, path expressions are traditional ω-regular expressions [Tho90] built from finite path expressions. For state formulas, we can use $\neg$ and $\vee$ to derive the usual Boolean connectives such as $\wedge$ and $\Rightarrow$. Finally, note that the wffs of $\mathrm{O}\textsc{r}_{\mathrm{E}}\mathrm{X}$ are just the state formulas.

3.2 Semantics

$\mathrm{O}\textsc{r}_{\mathrm{E}}\mathrm{X}$ formulas are interpreted over Σ-labelled transitions systems equipped with valuations. Let $TS = \langle X, \Sigma, \longrightarrow \rangle$ be a Σ-labelled transition system. A *valuation* $v : X \to 2^{Prop}$ specifies which propositions are true in each state.

We interpret finite path expressions over finite paths in TS, path expressions over infinite paths in TS, and state formulas at states of TS. If $p \in$ *finite_paths*(TS) and $e \in FPE$, we write $TS, p \models e$ to denote that e is satisfied along p. Similarly, for $\rho \in$ *paths*(TS) and $\alpha \in PE$, $TS, \rho \models \alpha$ denotes α is satisfied along ρ. Finally, for $x \in X$ and $\varphi \in \Phi$, $TS, x \models \varphi$ denotes that φ is satisfied at the state x.

We define concatenation of paths in the usual way. Let $p = x_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} x_n$ and $p' = x'_0 \xrightarrow{a'_1} \cdots \xrightarrow{a'_m} x'_m$ be paths such that $x_n = x'_0$. The path $p \cdot p'$ is the path $x_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} x_n \xrightarrow{a'_1} \cdots \xrightarrow{a'_m} x'_m$. The concatenation of a finite path p with an infinite path ρ is defined similarly.

The three forms of the satisfaction relation $\models$ are defined through simultaneous induction as follows.

Finite path expressions

$TS, p \models \varphi$ iff $p = x_0$ and $TS, x_0 \models \varphi$.
$TS, p \models a$ iff $p = x_0 \xrightarrow{a_1} x_1$ and $a_1 = a$.
$TS, p \models e_1 \cdot e_2$ iff $p = p_1 \cdot p_2$, $TS, p_1 \models e_1$ and $TS, p_2 \models e_2$
$TS, p \models e_1 + e_2$ iff $TS, p \models e_1$ or $TS, p \models e_2$
$TS, p \models e^*$ iff $p = x_0$ or
$p = p_1 \cdot p_2 \cdots p_m$ and for $i \in \{1, 2, \ldots, m\}$, $TS, p_i \models e$

Path expressions

$TS, \rho \models e_1 \cdot e_2^\omega$ iff there is a prefix $p_0 \cdot p_1 \cdot p_2 \cdots$ of ρ such that
$TS, p_0 \models e_1$ and for $i \in \{1, 2, \ldots\}$, $TS, p_i \models e_2$
$TS, \rho \models \alpha_1 + \alpha_2$ iff $TS, \rho \models \alpha_1$ or $TS, \rho \models \alpha_2$

We can associate with each path expression α a set Π_α of infinite paths in TS in the obvious way—$\Pi_\alpha = \{\rho \in paths(TS) \mid TS, \rho \models \alpha\}$. We let $\Pi_{\neg\alpha}$ denote the set $paths(TS) \setminus \Pi_\alpha$.

State formulas

$TS, x \models \mathtt{tt}$ always.
$TS, x \models P \in Prop$ iff $P \in v(x)$.
$TS, x \models \neg\varphi$ iff $TS, x \not\models \varphi$.
$TS, x \models \varphi_1 \vee \varphi_2$ iff $TS, x \models \varphi_1$ or $TS, x \models \varphi_2$.
$TS, x \models E_S\alpha$ iff S has a winning strategy for the game (Π_α, x).
$TS, x \models A_S\alpha$ iff S does not have a winning strategy for the game $(\Pi_{\neg\alpha}, x)$.

It is useful to think of $E_S\alpha$ as asserting that S has a strategy to *enforce* α along every path. Conversely, $A_S\alpha$ asserts that S does *not* have a strategy to *avoid* α along every path—in other words, no matter what strategy S chooses, there is at least one path consistent with the strategy along which α holds.

In the absence of fairness constraints, the games we have described are *determined* and $E_S\varphi$ is logically equivalent to $A_{\Sigma\setminus S}\varphi$. In other words, we can derive one quantifier from the other. However, once we introduce fairness constraints, the games are no longer determined, in general, and the two quantifiers need to be introduced independently, as we have done here. However, even with fairness constraints, $E_S\varphi$ implies $A_{\Sigma\setminus S}\varphi$.

3.3 Examples

Let e be a finite path expression. We shall use the following abbreviations for convenience.

- The formula $E_S e$ (respectively, $A_S e$) abbreviates the formula $E_S e\mathtt{tt}^\omega$ (respectively, $A_S e\mathtt{tt}^\omega$). Every path in $\Pi_{e\mathtt{tt}^\omega}$ has a finite prefix that satisfies the property e.
- The formula $E_S e^\omega$ (respectively, $A_S e^\omega$) abbreviates the formula $E_S \mathtt{tt} e^\omega$ (respectively, $A_S \mathtt{tt} e^\omega$).

The operators E and A of logics like CTL* are just abbreviations for E_Σ and $E_\emptyset$, respectively. Using these branching-time operators, we can specify some traditional temporal properties as follows. Let the set of actions be $\{a_1, a_2, \ldots, a_m\}$. In the examples, Σ abbreviates the finite path expression $(a_1 + a_2 + \cdots + a_m)$.

- The LTL formula $\varphi \mathcal{U} \psi$ (φ holds *until* ψ holds) is captured by the path expression $(\varphi\Sigma)^* \cdot \psi$. Thus, the state formula $A(\varphi\Sigma)^* \cdot \psi$ asserts that $\varphi\mathcal{U}\psi$ is true of every computation path.
- In LTL, the formula $\Diamond\varphi$ (φ holds eventually) can be written as $\mathtt{tt}\mathcal{U}\varphi$. The corresponding path expression is $(\mathtt{tt}\Sigma)^* \cdot \varphi$, which we shall denote *Eventually*(φ).
- Let *Invariant*(φ) denote the path expression $(\varphi\Sigma)^\omega$. The expression *Invariant*(φ) asserts that the state formula φ is *invariant* along the path, corresponding to the LTL formula $\Box\varphi$. Thus, the state formula A *Invariant*(φ) says that along all paths φ always holds.
- The formula $A((E(\Sigma^* \cdot \varphi) \cdot \Sigma)^\omega)$ asserts that along every path φ is always attainable (*Branching Liveness*). This property has been used in the verification of the Futurebus protocol [CGH$^+$95].

We can also assert properties that are *not* expressible in CTL* (or ATL*). For instance, $A(\Sigma a)^\omega$ asserts that along all paths, every even position is an a.

Now, let us see how to use OREX to describe properties of a typical open system. Figure 1 shows a variant of the train gate controller described in [AHK97]. In this system, the atomic propositions are $\{\mathsf{in\_gate}, \mathsf{out\_of\_gate}, \mathsf{waiting}, \mathsf{admitted}\}$. In the figure, the labels on the states indicate the valuation. The actions can be partitioned into two sets, *train* and *ctr*, corresponding to two components of the system, the train and the controller. The set *train* is given by $\{$*idle*, *request*, *relinquish*, *enter*, *leave*$\}$ while *ctr* is given by $\{$*grant*, *reject*, *eject*$\}$.

Here are some properties that one might like to assert about this system.

- If the train is outside the gate and has not yet been granted permission to enter the gate, the controller can prevent it from entering the gate:

$$A\ \mathit{Invariant}((\mathsf{out\_of\_gate} \wedge \neg\mathsf{admitted}) \Rightarrow E_{ctr}\mathit{Invariant}(\mathsf{out\_of\_gate}))$$

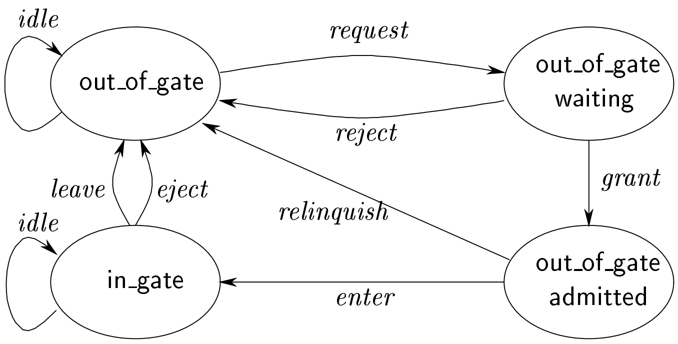

Fig. 1. A train gate controller

- If the train is outside the gate, the controller cannot force it to enter the gate:

$$A\ \mathit{Invariant}(\mathsf{out\_of\_gate} \Rightarrow A_{ctr}\mathit{Invariant}(\neg \mathsf{in\_gate}))$$

  This property can also expressed by the dual assertion that the train can choose to remain out of the gate.

$$A\ \mathit{Invariant}(\mathsf{out\_of\_gate} \Rightarrow E_{train}\mathit{Invariant}(\neg \mathsf{in\_gate}))$$

- If the train is outside the gate, the train and the controller can cooperate so that the train enters the gate:

$$A\ \mathit{Invariant}(\mathsf{out\_of\_gate} \Rightarrow E\ \mathit{Eventually}(\mathsf{in\_gate}))$$

- If the train is in the gate, the controller can ensure that it eventually leaves the gate.

$$A\ \mathit{Invariant}(\mathsf{in\_gate} \Rightarrow E_{ctr}\mathit{Eventually}(\mathsf{out\_of\_gate}))$$

  Notice that this property is not satisfied by the system unless fairness is introduced—after entering the gate, the train may execute the action *idle* forever. However, since *eject* is continuously enabled at this state, even weak fairness suffices to ensure that the controller can eventually force the train to leave the gate.
- Actually, it is unrealistic to assume that the controller can eject the train once it is in the gate. If we eliminate the transition labelled *eject*, we can make the following assertion which guarantees that as long as the train does not idle continuously, it eventually leaves the gate.

$$A\ \mathit{Invariant}(\mathsf{in\_gate} \Rightarrow (\neg(\mathit{idle})^{\omega} \Rightarrow \mathit{Eventually}(\mathsf{out\_of\_gate}))$$

  To state this property, we need to integrate assertions about actions and states in the formula. This formula cannot be conveniently expressed in ATL*, where formulas can only refer to properties of states.

4 The Model-Checking Algorithm

We now describe an automata-theoretic model checking algorithm for $\mathrm{O}\mathrm{R}_{\mathrm{E}}\mathrm{X}$ formulas. As we remarked earlier, path expressions can be regarded as ω-regular expressions over the infinite alphabet consisting of the finite set of actions Σ together with the set of all state formulas. However, if we fix a finite set of state formulas Ψ, we can regard each path expression which does not refer to state formulas outside Ψ as an ω-regular expression over the finite alphabet $\Sigma \cup \Psi$. We can associate with each such path expression α a language $L(\alpha)$ of infinite words over $\Sigma \cup \Psi$ using the standard interpretation of ω-regular expressions [Tho90].

Unfortunately, this naïve translation does not accurately reflect the meaning of path expressions: $\overline{L(\alpha)}$, the complement of $L(\alpha)$, may contain sequences that are models of α. For instance, if $L(\alpha)$ contains $a\varphi_1\varphi_2 b^\omega$ but does not contain $a\varphi_2\varphi_1 b^\omega$, the second path will be present in $\overline{L(\alpha)}$ though it models α. We require more structure in the infinite sequences we associate with path expressions to faithfully translate logical connectives into automata-theoretic operations.

We begin by defining an alternative model for finite path expressions and path expressions in terms of sequences over an extended alphabet. Let Ψ be a finite set of state formulas. A *Ψ-decorated sequence* over Σ is a sequence in which subsets of Ψ alternate with elements of Σ. More precisely, a Ψ-decorated sequence over Σ is an element of $(2^\Psi \cdot \Sigma)^*(2^\Psi) \cup (2^\Psi \cdot \Sigma)^\omega$. We use s to denote a finite Ψ-decorated sequence and σ to denote an arbitrary (finite or infinite) Ψ-decorated sequence. A finite Ψ-decorated sequence s can be concatenated with a Ψ-decorated sequence σ if the last element of s is identical to the first element of σ. The concatenation is obtained by deleting the last element of s.

We denote the set of finite path expressions and path expressions that do not refer to any state formulas outside Ψ as FPE_Ψ and PE_Ψ, respectively. We can interpret expressions from FPE_Ψ and PE_Ψ over Ψ-decorated sequences. The satisfaction relation is defined as follows:

$s \models \varphi$ iff $s = X_0$ and $\varphi \in X_0$
$s \models a$ iff $s = X_0 a X_1$
$s \models e_1 \cdot e_2$ iff $s = s_1 \cdot s_2$, $s_1 \models e_1$ and $s_2 \models e_2$
$s \models e_1 + e_2$ iff $s \models e_1$ or $s \models e_2$
$s \models e^*$ iff $s = X_0$ or $s = s_1 \cdot s_2 \cdots s_n$ and for $i \in \{1, 2, \ldots, n\}$, $s_i \models e$
$\sigma \models e_1 \cdot e_2^\omega$ iff there is a prefix $s_0 \cdot s_1 \cdot s_2 \cdots$ of σ such that $s_0 \models e_1$ and for $i \in \{1, 2, \ldots\}$, $s_i \models e_2$
$\sigma \models \alpha_1 + \alpha_2$ iff $\sigma \models \alpha_1$ or $\sigma \models \alpha_2$

Let $TS = \langle X, \Sigma, \longrightarrow\rangle$ be a Σ-labelled transition system and $v : X \to \mathit{Prop}$ be a valuation over TS. For each path $\rho = x_0 \xrightarrow{a_1} x_1 \ldots$ in TS, the corresponding Ψ-decorated sequence σ_ρ is $X_0 a_1 X_1 \ldots$ where $X_i = \{\varphi \mid \varphi \in \Psi \text{ and } TS, x_i \models \varphi\}$. Then, for any path expression $\alpha \in PE_\Psi$, $TS, \rho \models \alpha$ if and only if $\sigma_\rho \models \alpha$.

There is a natural connection between the language $L(\alpha)$ defined by $\alpha \in PE_\Psi$ and the semantics of α in terms of Ψ-decorated sequences. To formalize this, we need to define when a Ψ-decorated sequence embeds a sequence over $\Sigma \cup \Psi$.

Embedding Let $w = w_0 w_1 \ldots$ be an infinite word over $\Sigma \cup \Psi$ and $\sigma = \sigma_0 \sigma_1 \ldots$ be a Ψ-decorated sequence. Observe that $\sigma_i \subseteq \Psi$ if i is even and $\sigma_i \in \Sigma$ if i is odd. We say that σ *embeds* w if there is a monotone function $f : \mathbb{N}_0 \to \mathbb{N}_0$ mapping positions of w into positions of σ such that:

(i) If i is in the range of f, $j < i$ and j is odd, then j is in the range of f.
(ii) If $w_i \in \Sigma$ then $w_i = \sigma_{f(i)}$.
(iii) If $w_i \in \Psi$ then $w_i \in \sigma_{f(i)}$.

We can then establish the following connection between $L(\alpha)$ and the set of Ψ-decorated sequences that model α.

Lemma 4.1. *A Ψ-decorated sequence σ models a path expression $\alpha \in PE_\Psi$ if and only if σ embeds at least one of the sequences in the language $L(\alpha)$ over the alphabet $\Sigma \cup \Psi$.*

Let $|\alpha|$ denote the number of symbols in a path expression α. From the theory of ω-regular languages [Tho90], we know that for each path expression $\alpha \in PE_\Psi$, we can construct a nondeterministic Büchi automaton $\mathcal{A}_\alpha$ over $\Sigma \cup \Psi$ whose size is polynomial in $|\alpha|$ and whose language is precisely $L(\alpha)$.

Using our notion of embedding, we can associate with each Büchi automaton $\mathcal{A}$ over $\Sigma \cup \Psi$ an extended language $L^+(\mathcal{A})$ of Ψ-decorated sequences as follows:

$$L^+(\mathcal{A}) = \{\sigma \mid \exists w \in L(\mathcal{A}) : \sigma \text{ embeds } w\}$$

Lemma 4.2. *For any Büchi automaton $\mathcal{A}$ over $\Sigma \cup \Psi$, we can construct a Büchi automaton $\mathcal{A}^+$ over $\Sigma \cup 2^\Psi$ such that $\mathcal{A}^+$ reads elements from Σ and 2^Ψ alternately, $L^+(\mathcal{A}) = L(\mathcal{A}^+)$ and the number of states of $\mathcal{A}^+$ is polynomial in the number of states of $\mathcal{A}$.*

Proof Sketch: Let $\mathcal{A} = (Q, \delta, q_0, F)$ be a given nondeterministic Büchi automaton over $\Sigma \cup \Psi$. For states $p, q \in Q$ and $U \subseteq \Psi$, we write $p \overset{U}{\Rightarrow} q$ if there is a sequence of transitions from p ending in q whose labels are drawn from the set U. We write $p \overset{U}{\Rightarrow}_F q$ to indicate that there is a sequence of transitions from p to q that passes through a state from F, all of whose labels are drawn from the set U. We write $p \overset{U}{\Rightarrow}_A$ if there is an accepting run starting at p, all of whose labels are drawn from the set U.

The automaton $\mathcal{A}^+ = (Q^+, \Sigma \cup 2^\Psi, \delta^+, F^+, q_0)$ where:

$$\begin{aligned}
Q^+ &= (Q \times \{0,1,2\}) \cup \{q_f, p_f\} \\
F^+ &= (Q \times \{2\}) \cup \{p_f\} \\
\delta^+ &= \{((p,0), U, (q,1)) \mid p \overset{U}{\Rightarrow} q\} \cup \{((p,0), U, (q,2)) \mid p \overset{U}{\Rightarrow}_F q\} \cup \\
&\quad \{((p,0), U, q_f) \mid p \overset{U}{\Rightarrow}_A\} \cup \\
&\quad \{((p,1), a, (q,0)), ((p,2), a, (q,0)) \mid (p,a,q) \in \delta\} \cup \\
&\quad \{((p,0), U, (p,1)) \mid p \in Q, U \subseteq \Psi\} \cup \{(q_f, a, p_f) \mid a \in \Sigma\} \cup \\
&\quad \{(p_f, U, q_f) \mid U \subseteq \Psi\}
\end{aligned}$$

□

The construction allows us to convert a path expression $\alpha \in PE_\Psi$ into an automaton $\mathcal{A}^+_\alpha$ over $\Sigma \cup 2^\Psi$ whose state space is polynomial in $|\alpha|$. Notice that the alphabet, and hence the number of transitions, of $\mathcal{A}^+_\alpha$ is exponential in $|\alpha|$.

The lemma assures us that if we complement $\mathcal{A}^+_\alpha$, we obtain an automaton that accepts precisely those Ψ-decorated sequences that do *not* model α. As we remarked earlier, this is not true of the original automaton $\mathcal{A}_\alpha$: $\overline{L(\alpha)}$ may contain sequences that can be embedded into Ψ-decorated sequences that model α. In a sense, $L^+(\mathcal{A})$ can be thought of as the semantic closure of $L(\mathcal{A})$.

Lemma 4.3 ([Saf88,EJ89]). *A Büchi automaton $\mathcal{A}$ can be effectively determinized (complemented) to get a deterministic Rabin automaton whose size is exponential in the number of states of $\mathcal{A}$ and linear in the number of transitions of $\mathcal{A}$ and whose set of accepting pairs is linear in the number of states of $\mathcal{A}$. This automaton can be constructed in time exponential in the number of states of $\mathcal{A}$ and polynomial in the number of transitions of $\mathcal{A}$.*

Lemma 4.4 follows from the previous two lemmas.

Lemma 4.4. *From a path expression $\alpha \in PE_\Psi$, we can construct a deterministic Rabin automaton $\mathcal{A}^R_\alpha$ ($\overline{\mathcal{A}}^R_\alpha$) that accepts exactly those Ψ-decorated paths that do (do not) model α. The number of states of $\mathcal{A}^R_\alpha$ ($\overline{\mathcal{A}}^R_\alpha$) is exponential in $|\alpha|$ and the number of accepting pairs of $\mathcal{A}^R_\alpha$ ($\overline{\mathcal{A}}^R_\alpha$) is polynomial in $|\alpha|$.*

Theorem 4.5. *Given a transition system TS, a state x in TS and an* ORᴇX *formula φ, the model checking problem $TS, x \overset{?}{\models} \varphi$ can be solved in time $O((mk)^k)$, where m is given by $2^{O(|\varphi|)} \cdot |TS|$ and k is a polynomial in $|\varphi|$.*

Proof Sketch: We use the bottom-up labelling algorithm of CTL* and ATL*. The only interesting case is when the formula φ to be checked is of the form $E_S\alpha$ (or $A_S\alpha$). Each S-strategy f at a state x can be represented as a tree by unfolding TS starting at x, retaining all $\Sigma \setminus S$ edges in the unfolding, but discarding all S-transitions that are not chosen by the strategy. Conversely, if we unfold TS at x and prune S-transitions while ensuring that at least one S-transition is retained at each state y where *enabled*$(y) \cap \longrightarrow_S \neq \emptyset$, we obtain a representation of some S-strategy at x.

Let Ψ be the set of state formulas that appear in α. We may assume that each state of the transition system TS is labelled with the formulas from Ψ that hold at that state. In the trees that we use to represent strategies, each state is labelled by the same subset of formulas as the corresponding state in TS.

The trees that represent strategies carry labels on both the states and the edges. We can convert these to state-labelled trees by moving the label on each edge to the destination state of the edge.

Given a transition system TS, we can construct a Büchi tree automaton $\mathcal{T}_{TS}$ that accepts precisely those trees labeled by $\Sigma \cup 2^\Psi$ that arise from strategies, as described in [KV96]. The size of $\mathcal{T}_{TS}$ is linear in the size of TS.

An S-strategy f is a witness for $TS, x \models E_S\alpha$ if and only if every infinite path in the tree corresponding to f models α. Equivalently, every infinite path in this tree, when treated as a sequence over $\Sigma \cup 2^{\Psi}$, is accepted by the Rabin automaton $\mathcal{A}_\alpha^R$ constructed from $\mathcal{A}_\alpha^+$ as described in Lemma 4.4. Since $\mathcal{A}_\alpha^R$ is deterministic, we can convert it into a Rabin tree automaton $\mathcal{T}_\alpha$ that runs $\mathcal{A}_\alpha^R$ along all paths and accepts exactly those trees in which all infinite paths model α. The automaton $\mathcal{T}_\alpha$ has the same size as $\mathcal{A}_\alpha^R$—that is, the number of states is exponential in $|\alpha|$ and the number of accepting pairs is polynomial in $|\alpha|$.

Finally, we construct the product of $\mathcal{T}_{TS}$ and $\mathcal{T}_\alpha$ to obtain a tree automaton that accepts a tree if and only if it corresponds to a strategy f such that all infinite paths in TS consistent with f satisfy α. We can then use the algorithm of Emerson and Jutla [EJ88] to check whether this product automaton accepts a nonempty set of trees. The Emerson-Jutla algorithm solves the emptiness problem for a tree automaton with i states and j accepting pairs in time $O((ij)^{3j})$. From this, we can derive that the emptiness of the product tree automaton we have constructed can be checked in time $O((mk)^k)$, where m is given by $2^{O(|\alpha|)} \cdot |TS|$ while k is a polynomial in $|\alpha|$.

The case $A_S\alpha$ is similar to $E_S\alpha$ with one modification—use the construction due to Emerson and Jutla [EJ89] instead of Safra's construction to obtain a deterministic Rabin automaton $\overline{\mathcal{A}}_\alpha^R$ that accepts exactly those sequences over $\Sigma \cup 2^{\Psi}$ that do *not* model α.

In the constructions above, we have not taken into account the case when we are interested only in (weakly or strongly) S-fair computations. Fairness constraints can be handled by adding a simple Rabin condition to the automaton $\mathcal{T}_{TS}$ that accepts all trees corresponding to S-strategies. This extra structure does not materially affect the overall complexity of the procedure.

□

5 Related Work

In [AHK97], the logic ATL* is interpreted over two kinds of transition systems: synchronous structures and asynchronous structures. In each case, the notion of *agent* plays a prominent role in the definition. In a *synchronous structure*, each state is associated with a unique agent. An important subclass of synchronous structures is the one with two agents, one representing the system and the other the environment. Such systems have been studied by Ramadge and Wonham [RW89] in the context of designing controllers for discrete-event systems.

In an *asynchronous structure*, each transition is associated with an agent. A synchronous structure can be thought of as a special kind of asynchronous structure where all transitions out of a state are associated with a single agent. Our Σ-labelled transition systems correspond to asynchronous structures, where the set of agents corresponds to the alphabet Σ.

The notion of strategy described in [AHK97] is slightly different from the one we use here. In their framework, each agent has a strategy and an S-strategy at a state allows any move permitted by all the individual strategies of the agents in S. Our definition of global strategy is more generous. We believe that decompositions of strategies where each agent has a view of the global state do not arise naturally and have therefore chosen not to incorporate this into our logic. Nevertheless, we can easily simulate the framework of [AHK97] in our setup without changing in the complexity of the construction.

In the Introduction, we mentioned that $\mathrm{O}\mathrm{R}_\mathrm{E}\mathrm{X}$ subsumes ATL* in expressive power since ω-regular expressions are more powerful than LTL. We also note that in the context of open systems, it is natural to refer to properties of *both* states and actions in a transition system. If we use LTL to express properties of paths, as is the case with ATL*, we have to encode actions in the states of the system since LTL can only talk about properties of states. On the other hand $\mathrm{O}\mathrm{R}_\mathrm{E}\mathrm{X}$ provides a seamless way of interleaving assertions about actions and states along a path. This makes for more natural models of open systems—for instance, compare our example of the train gate controller with explicit action labels with the version in [AHK97] where the action labels are encoded as propositions.

There has been earlier work on game logics that mention actions. In [Par83], Parikh defines a propositional logic of games that extends propositional dynamic logics [Har84]. The logic studied in [Par83] is shown to be decidable. A complete axiomatization is provided, but the model-checking problem is not studied.

6 Conclusions

As Theorem 4.5 indicates, the model-checking complexity of $\mathrm{O}\mathrm{R}_\mathrm{E}\mathrm{X}$ is at least one exponential better than that of ATL*, though $\mathrm{O}\mathrm{R}_\mathrm{E}\mathrm{X}$ subsumes ATL* in expressive power. If we restrict the quantifiers of $\mathrm{O}\mathrm{R}_\mathrm{E}\mathrm{X}$ to the normal branching-time interpretation A and E, we obtain a system called $\mathrm{B}\mathrm{R}_\mathrm{E}\mathrm{X}$ that corresponds to the branching-time logic ECTL*. The model-checking algorithm for $\mathrm{B}\mathrm{R}_\mathrm{E}\mathrm{X}$ is exponential, which is comparable to the complexity of model-checking CTL*.

Why does going from CTL* to ATL* add an exponential to the complexity of model-checking, unlike the transition from $\mathrm{B}\mathrm{R}_\mathrm{E}\mathrm{X}$ to $\mathrm{O}\mathrm{R}_\mathrm{E}\mathrm{X}$? The construction in [VW86] can be used to build nondeterministic Büchi automata for LTL formulas α and $\neg\alpha$ with exponential blow-up. Thus the complexity of model-checking CTL* stays exponential. When we translate $\mathrm{B}\mathrm{R}_\mathrm{E}\mathrm{X}$ path formulas into automata we only incur a polynomial blowup in size, but we may have to complement the automaton for α to get an automaton for $\neg\alpha$. Complementation is, in general, exponential and, consequently, model-checking for $\mathrm{B}\mathrm{R}_\mathrm{E}\mathrm{X}$ is also exponential.

When going from CTL* to ATL*, we need to determinize the Büchi automaton constructed from the LTL formula α so that we can interpret the automaton over trees. This blows up the automaton by another exponential, resulting in a model-checking algorithm that is doubly exponential in the size of the input formula. In $\mathrm{O}\mathrm{R}_\mathrm{E}\mathrm{X}$ the automaton we construct for path formulas is already determinized, so we avoid the second exponential blow-up.

Finally, we note that if we restrict our syntax to only permit the existential path quantifier $E\alpha$, we can model-check $\mathrm{B}\mathrm{R}_\mathrm{E}\mathrm{X}$ in polynomial time — essentially, it suffices to construct a nondeterministic automaton for α. On the other hand, the model-checking problem for $\mathrm{OR_EX}$ remains exponential even with this restriction because we have to determinize the automaton for α. Since ω-regular languages are closed under complementation, we can reduce any formula in $\mathrm{BR_EX}$ or $\mathrm{OR_EX}$ to one with just existential path quantifiers. However, in the process, we have to complement ω-regular expressions. Since this is, in general, an exponential operation, the cost of this translation is exponential (perhaps nonelementary, since nested negations introduce nested exponentials). Thus, the full language that we have presented here permits more succinct specifications *and* more efficient verification than the reduced language with just existential quantifiers.

References

[AHK97] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proceedings of the 38th IEEE Symposium on the Foundations of Computer Science*. IEEE Press, 1997.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In D. Kozen, editor, *Proceedings of the Workshop on Logic of Programs,* Yorktown Heights, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.

[CGH+95] E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D. E. Long, K. L. McMillan, and L. A. Ness. Verification of the Future+ cache coherence protocol. *Formal Methods in System Design*, 6:217–232, 1995.

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

[EJ88] E.A.Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proceedings of the IEEE FOCS*, 1988.

[EJ89] E.A.Emerson and C. Jutla. On simultaneously determinising and complementing ω-automata. In *Proceedings of the IEEE FOCS*, 1989.

[Har84] David Harel. Dynamic logic. In *Handbook of Philosophical Logic, Volume II.* Reidel, 1984.

[Hol97] G. J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, 23(5):279–295, 1997.

[KV96] Orna Kupferman and Moshe Vardi. Module checking. In *Proceedings of CAV'96, LNCS 1102*. Springer-Verlag, 1996.

[McM93] K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic, 1993.

[Par83] R. Parikh. Propositional game logic. In *Proceedings of 24th IEEE FOCS*, pages 195–200, 1983.

[Pnu76] A. Pnueli. The temporal logic of programs. In *Proceedings of the IEEE Symposium on the Foundations of Computing Science.* IEEE Press, 1976.

[RW89] P.J.G. Ramadge and W.M. Wonham. The control of discrete-event systems. *IEEE Trans. on Control Theory*, 77:81–98, 1989.

[Saf88] S. Safra. On complexity of ω-automata. In *In the Proceedings of the 29th FOCS*, 1988.

[Tho90] W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science, Volume B.* Elsevier Science Publishers, 1990.

[VW86] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Symposium on Logic in Computer Science (LICS '86)*, pages 332–344, Cambridge, Massachusetts, June 1986. Computer Society Press.