

Embedding Finite Automata within Regular Expressions

Shoham Ben-David¹
shoham@il.ibm.com

Dana Fisman^{1,2}
dana@weizmann.ac.il

Sitvanit Ruah¹
sitvanit@il.ibm.com

¹ IBM Haifa Research Lab ² Weizmann Institute of Science

Category: Technical Paper

Abstract. Regular expressions and their extensions have become a major component of industry-standard specification languages such as PSL/Sugar ([2]). The model checking procedure of regular expression based formulas, as well as of many LTL and CTL formulas, involves constructing an automaton which runs in parallel with the model.

In this paper we re-examine this construction. Instead of directly translating a regular expression into an automaton, as traditionally done, we propose an algorithm which allows an intermediate representation mixing both regular expressions and automata. This representation can be thought of as plugging an automaton inside a regular expression, to replace an existing sub-expression. In order to be verified, the intermediate representation is then translated into another automaton, resulting in a set of automata running in parallel. A key feature of this algorithm is that the plug-in automaton is independent of the regular expression from which it originated, and thus can be used in several different properties.

The contribution of our work is manifold, as demonstrated in the paper. It allows *modularity* and *flexibility* of the automata construction, and can increase *expressiveness* when SERES are mixed with CTL. In many cases it significantly reduces the *size* of the automata built for formulas, thus reducing the overall run time of verification.

1 Introduction

Symbolic model checking has been found extremely efficient in the verification of hardware designs, and has been widely adopted in industry in recent years. While traditional model checkers ([13]) used the temporal logics CTL or LTL as their specification language, contemporary industrial languages, have sought ways to make the specification language easier to learn and use. The industry-standard language PSL/Sugar [2], as well as other industry oriented languages (e.g. [4]), augment the logic with the use of Extended Regular Expressions (ERES, or SERES using the formulation of [2]).

A SERE specification can be viewed as a sequence of Boolean events describing a desirable behavior of the model. For example, the SERE formula $\varphi = \{req \cdot \neg ack^* \cdot ack\}!$ asserts that on all execution paths of the model, *req* is active on the first cycle, *ack* is then inactive for zero or more cycles, and then *ack* becomes active. Similarly, SERES can be used to describe an *undesireable* behavior of the model. The formula $\text{not } \{req \cdot \neg ack^* \cdot ack\}!$ asserts that there *does not exist* an execution path on which *req* is active on the first cycle, *ack* is then inactive for zero or more cycles, and then *ack* becomes active.

In this paper we consider formulas given as not SERE! . These formulas have a special importance for two main reasons. The first is that a large subset of LTL, CTL and other SERE-based properties can be automatically reduced to an equivalent formula of the form not $r!$ ¹. The second reason is the efficient model checking methods they enjoy, as explained below. A not $r!$ formula has the nature that it is sufficient to find one execution path of the model satisfying r , in order to conclude the formula does not hold in the model. This nature allows a not $r!$ formula to be modelled by a non-deterministic finite automaton (NFA) N_r , which accepts sequences satisfying r , and which is *linear* in the size of r . Running it together with the model, we then verify the invariant property $\text{AG } \neg \text{bad}$ where bad is a boolean expression asserting that N_r is in an accepting state. The reduction to an invariant property is important, since invariant properties are easier to verify by different model checking engines [7]. The efficient verification algorithm, together with the wide variety of properties which can be transformed into not SERE! formulas, have made them the major translation path of the IBM model checking tool-set RuleBase [5] and the topic of this paper.

The translation of a not SERE! formula into an NFA is re-examined in this paper. Instead of directly translating a formula into an automaton, as traditionally done, we propose an *embedding* algorithm which allows the translation to be modular, by allowing an intermediate representation of the formula which mixes both SERE and NFA. Let r be a SERE and s a sub-SERE of r . We show a process in which s is plugged out of r , a separate side-NFA is built for it, and a simpler SERE referring to the side-NFA is plugged into r , replacing s . A key feature of this algorithm is the fact that the side-NFA built is *independent* of the SERE it originated from. This allows the side-NFA to be plugged-in in any other SERE when appropriate.

The use of the embedding algorithm has several advantages, on which we elaborate in this paper. It allows *modularity* of the automata construction, as automata previously built can be plugged into a more complex SERE. The modularity leads to *flexibility* in the way the automaton is constructed, since different parts of the SERE can use different translations. It can increase *expressiveness* when a not SERE! property is translated to CTL. Since SEREs are more expressive than CTL [15], the embedding is needed for the parts which are not expressible in CTL. Finally, the use of embedding can significantly reduce the *size* of the automata built for formulas, thus reducing the overall run time of verification.

The rest of the paper is organized as follows. Section 2 covers some preliminaries. Section 3 gives our embedding algorithm, and in section 4 we discuss the benefits of the algorithm and give example applications. Section 5 concludes the paper. The proofs appear in Appendix A.

¹ Beer et al. [6] have shown how to translate a subset of CTL and SERE-based formulas into not SERE! formulas. Since those CTL formulas lay in the common fragment of CTL and LTL [12], their LTL counterparts also have an equivalent in this form.

2 Preliminaries

2.1 The computational Model - DTS

We represent a finite state program by a *discrete transition system*. A discrete transition system (DTS) is a symbolic representation of a finite automaton on finite or infinite words. The definition is derived from the definition of a fair discrete system (FDS) [11]. A DTS $\mathcal{D} : \langle V, \Theta, \rho, \mathcal{A}, \mathcal{J} \rangle$ consists of the following components:

- $V = \{u_1, \dots, u_n\}$: A finite set of typed *state-variables* over possibly infinite domains. We define a *state* s to be a type-consistent interpretation of V , assigning to each variable $u \in V$ a value $s[u]$ in its domain. We denote by Σ_V the set of all states, and by B_V the set of all boolean expressions over the state-variables in V (when V is understood from the context we write simply Σ and B , respectively).
- Example 1.* Let V denote the set $\{a, b, c\}$ of Boolean state variables. Then Σ_V the set of interpretation of these variables contains the 8 interpretations giving different truth values to a , b and c . That is, $\Sigma_V = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$. And B_V is the set of all boolean expressions over these variables. For example, the boolean expressions a , $a \vee b$, $\neg c$ and $(a \rightarrow \neg b) \wedge c$ are all in B_V . \dashv
- Θ : The *initial condition*. This is an assertion characterizing all the initial states of the DTS.
- ρ : The *transition relation*. This is an assertion $\rho(V, V')$ relating a state $s \in \Sigma_V$ to its \mathcal{D} -successor $s' \in \Sigma_V$ by referring to both unprimed and primed versions of the state-variables. The transition relation $\rho(V, V')$ identifies state s' as a \mathcal{D} -successor of state s if $\langle s, s' \rangle \models \rho(V, V')$, where $\langle s, s' \rangle$ is the joint interpretation which interprets $u \in V$ as $s[u]$ and u' as $s'[u]$.
- \mathcal{A} : The *accepting condition* for finite words. This is an assertion characterizing all the accepting states for runs of the DTS satisfying finite words.
- $\mathcal{J} = \{J_1, \dots, J_k\}$: The *justice (Büchi) accepting condition* for infinite words. This is a set of assertions characterizing the sets of accepting states for runs of the DTS satisfying infinite words. The justice requirement $J \in \mathcal{J}$ stipulates that every infinite computation contains infinitely many states satisfying J .

Let \mathcal{D} be a DTS for which the above components have been identified. We define a *run* of \mathcal{D} to be a finite or infinite non-empty sequence of states $\sigma : s_0 s_1 s_2 \dots$ satisfying the requirements of *initiality* i.e. that $s_0 \models \Theta$; and of *consecution* i.e. that for each $j = 0, 1, \dots$, the state s_{j+1} is a \mathcal{D} -successor of state s_j . A run satisfying the requirement of *maximality* i.e. that it is either infinite, or terminates at a state s_k which has no \mathcal{D} -successors is termed a *maximal run*. Let $U \subseteq V$ be a subset of the state-variables. A run $\sigma : s_0 s_1 s_2 \dots s_n \dots$ is said to be *satisfying a finite word* $w = b_0 b_1 \dots b_n$ over B_U iff for every i , $0 \leq i \leq n$, $s_i \models b_i$. A run $\sigma : s_0 s_1 s_2 \dots s_{n+1} \dots$ satisfying a finite word $w = b_0 b_1 \dots b_n$ is said to be *accepting* w iff s_{n+1} satisfies \mathcal{A} . An infinite run $\sigma : s_0 s_1 s_2 \dots$ is said to be *satisfying an infinite word* $w = b_0 b_1 \dots$ over B_U iff for every $i \geq 0$, $s_i \models b_i$. An infinite run σ satisfying an infinite word w is said to be *accepting* w iff for each $J \in \mathcal{J}$, the run σ contains infinitely many states satisfying J .

For a state s , we denote by $s|_U$ the restriction of s to the state-variables in U , i.e. the state $s|_U$ agrees with s on the interpretation of the state-variables in U , and does not provide an interpretation for variables in $V \setminus U$. For a run $\sigma = s_0 s_1 s_2 \dots$ we denote by $\sigma|_U$ the run $s_0|_U s_1|_U s_2|_U \dots$

Discrete transition systems can be composed in parallel. Let $\mathcal{D}_i = \langle V_i, \Theta_i, \rho_i, \mathcal{A}_i, \mathcal{J}_i \rangle$, $i \in \{1, 2\}$, be two discrete transition systems. We denote the *synchronous parallel composition* of \mathcal{D}_1 and \mathcal{D}_2 by $\mathcal{D}_1 \parallel \mathcal{D}_2$ and define it to be $\mathcal{D}_1 \parallel \mathcal{D}_2 = \langle V_1 \cup V_2, \Theta_1 \wedge \Theta_2, \rho_1 \wedge \rho_2, \mathcal{A}_1 \wedge \mathcal{A}_2, \mathcal{J}_1 \cup \mathcal{J}_2 \rangle$. We can view the execution of \mathcal{D} as the *joint* execution of \mathcal{D}_1 and \mathcal{D}_2 .

From Finite Automata to DTS

Given a non-deterministic finite automata on finite words (NFA) [10] whose alphabet is a set of boolean expressions over a given set of variables V , it is straightforward to construct the discrete transition system corresponding to it. The same holds for a generalized Büchi automaton on infinite words (GBA) [14].

Example 2. Assume we have the set of state variables $V = \{a, b, c\}$. Let N be the NFA over B_V described in figure 1. Then the DTS $\mathcal{D} = \langle V, \Theta, \rho, \mathcal{A}, \mathcal{J} \rangle$ described below

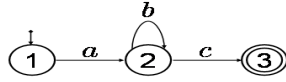


Fig. 1. An NFA over B_V where $V = \{a, b, c\}$.

corresponds to N , where st_N is a new variable with domain $\{0, 1, 2, 3\}$.

$$V = \{a, b, c, st_N\}; \quad \Theta = (st_N = 1); \quad \mathcal{A} = (st_N = 3); \quad \mathcal{J} = \emptyset; \quad \text{and}$$

$$\begin{aligned} \rho = & (st_N = 1 \wedge a \wedge st'_N = 2) \vee (st_N = 2 \wedge b \wedge st'_N = 2) \vee \\ & (st_N = 2 \wedge c \wedge st'_N = 3) \vee (st_N = 1 \wedge \neg a \wedge st'_N = 0) \vee \\ & (st_N = 2 \wedge \neg b \wedge st'_N = 0) \vee (st_N = 2 \wedge \neg c \wedge st'_N = 0) \end{aligned} \quad \lrcorner$$

Formally, let V be a set of state-variables and let B be the corresponding set of boolean expressions. Let $N = \langle B, Q, Q_0, \delta, A \rangle$ be an NFA. Let $state$ be a new variable (not in V) whose domain is $Q \cup \{q_{sink}\}$ where $q_{sink} \notin Q$. Then, N can be represented as the DTS $\mathcal{D}^N = \langle V^N, \Theta^N, \rho^N, \mathcal{A}^N, \mathcal{J}^N \rangle$ where

$$V^N = V \cup \{state\}; \quad \Theta^N = \bigvee_{q_0 \in Q_0} (state = q_0); \quad \mathcal{A}^N = \bigvee_{q \in A} (state = q); \quad \mathcal{J}^N = \emptyset;$$

$$\rho^N = \bigvee_{(q_1, \sigma, q_2) \in \delta} \begin{aligned} & (state = q_1 \wedge \sigma \wedge state' = q_2) \vee \\ & (state = q_1 \wedge \neg \sigma \wedge state' = q_{sink}) \end{aligned}$$

Similarly, we can construct the DTS corresponding to a Büchi automaton. Let $G = \langle B, Q, Q_0, \delta, \mathcal{F} \rangle$ be a GBA with $\mathcal{F} = \{F_1, \dots, F_k\}$. Then, G can be represented as the discrete transition system $\mathcal{D}^G = \langle V^G, \Theta^G, \rho^G, \mathcal{A}^G, \mathcal{J}^G \rangle$ where:

$$\begin{aligned}
V^G &= V \cup \{state\}; & \Theta^G &= \bigvee_{q_0 \in Q_0} (state = q_0); & \mathcal{A}^G &= \emptyset; \\
\mathcal{J}^G &= \{J_1, \dots, J_k\} \text{ where for each } 1 \leq i \leq k: & J_i &= \bigvee_{q \in F_i} (state = q); \\
\rho^G &= \bigvee_{(q_1, \sigma, q_2) \in \delta} \left(\begin{aligned} &(state = q_1 \wedge \sigma \wedge state' = q_2) \vee \\ &(state = q_1 \wedge \neg \sigma \wedge state' = q_{sink}) \vee \\ &(state = q_{sink} \wedge state' = q_{sink}) \end{aligned} \right)
\end{aligned}$$

In this paper, given an NFA $N = \langle B, Q, Q_0, \delta, A \rangle$ we first construct a terminal Büchi automaton [9] by adding a self loop on all accepting states of N and defining the Büchi accepting sets to be the singleton set of accepting states (i.e. $\{A\}$). This Büchi automaton accepts all words which have a finite prefix accepted by N . Then we construct a DTS for the resulting terminal Büchi automaton. We denote the resulting DTS \mathcal{D}_N . Let $\sigma = \sigma_0 s_1 \dots$ be a run of \mathcal{D}_N . We say that the “step” (s_i, s_{i+1}) of \mathcal{D}_N corresponds to the transition $(q_1, \sigma, q_2) \in \delta$ of N iff $(s_i, s_{i+1}) \models (state = q_1 \wedge \sigma \wedge state' = q_2)$.

Example 3. The DTS $\mathcal{D}_N = \langle V_N, \Theta_N, \rho_N, \mathcal{A}_N, \mathcal{J}_N \rangle$ for the terminal Büchi constructed for the NFA N described in Figure 1 is as follows where $\mathcal{D} = \langle V, \Theta, \rho, \mathcal{A}, \mathcal{J} \rangle$ is the DTS from Example 2.

$$\begin{aligned}
V_N &= V; & \Theta_N &= \Theta; & \mathcal{A}_N &= \mathcal{A}; & \mathcal{J}_N &= \{\mathcal{A}_N\}; & \text{and} \\
\rho_N &= \rho \vee (st_N = 3 \wedge st'_N = 3) \vee (st_N = q_{sink} \wedge st'_N = q_{sink}) & \lrcorner
\end{aligned}$$

2.2 The logic

The logic considered in this paper is the fragment of the industry-standard temporal logic PSL/Sugar [2] which consists of only not $r!$ formulas, with r being a Sugar extended regular expression (SERE). These formulas are of special interest for several reasons. First, it is a convenient way for specification which is widely used. Second, a large subset of the PSL/Sugar properties can be automatically translated into not $r!$ properties [6, 12]. Finally, the verification of these properties can be reduced to verification of invariant properties and is thus very efficient (see e.g. [6]).

The semantics of these formulas over a given DTS is defined below. The definition assumes a set of state variables V , the corresponding set Σ of interpretations of the state-variables in V and the set B of boolean expressions over V . We assume two designated boolean expressions *true* and *false* belong to B , such that for every $s \in \Sigma$, $s \models \text{true}$ and $s \not\models \text{false}$.

Definition 1 (SERES).

- The empty set \emptyset and the empty regular expression λ are SERES.
- Every boolean expression $b \in B$ is a SERE.
- If r , r_1 , and r_2 are SERES, then the following are also SERES:
 1. $\{r\}$ (encapsulation)
 2. $r_1 \cup r_2$ (union)
 3. $r_1 \cdot r_2$ (concatenation)
 4. r^* (Kleene closure)
 5. $r_1 \circ r_2$ (fusion)
 6. $r_1 \cap r_2$ (intersection)

We denote by RES standard regular expressions, i.e. the subset of SERES with no fusion or intersection operators. To define the semantics of SERES, we use the following notations.

Notations

We denote a letter from Σ by s (possibly with subscripts) and a word from Σ by u , v , or w . The *concatenation* of u and v is denoted by uv . If u is infinite, then $uv = u$. The empty word is denoted by ϵ , so that $w\epsilon = \epsilon w = w$. The *overlapping concatenation* of us and sv , denoted by $us \circ sv$, is the word usv . Let L_1 and L_2 be sets of words. The *concatenation* of L_1 and L_2 , denoted L_1L_2 is the set $\{w \mid \exists w_1 \in L_1, \exists w_2 \in L_2 \text{ and } w = w_1w_2\}$. The *overlapping concatenation* of L_1 and L_2 , denoted $L_1 \circ L_2$ is the set $\{w \mid \exists w_1s \in L_1, \exists sw_2 \in L_2 \text{ and } w = w_1sw_2\}$. Define $L^0 = \{\epsilon\}$ and $L^i = LL^{i-1}$ for $i \geq 1$. The *Kleene closure* of L denoted L^* is the set $\bigcup_{i < \omega} L^i$.²

We denote the length of a word w by $|w|$. An empty word $w = \epsilon$ has length 0, and a finite word $w = (s_0s_1s_2 \cdots s_n)$ has length $n + 1$. We use i , j , and k to denote non-negative integers. For $i < |w|$, we use w^i to denote the $(i + 1)^{th}$ letter of w (since counting of letters starts at zero). For a subset $U \subseteq V$ of state-variables, we denote by $s|_U$ the restriction of the letter s to the state-variables in U . For a word $w = s_0s_1s_2 \cdots$ we denote by $w|_U$ the restriction of every letter in w to the state-variables in U (i.e. $w|_U = s_0|_U s_1|_U s_2|_U \cdots$).

For a formula φ and a sub-formula of it ψ we denote by $\varphi[\psi \leftarrow \psi']$ the formula obtained by replacing every occurrence of ψ in φ by ψ' .

Definition 2 (SERES semantics). *The semantics of SERES are defined using the relation \models between SERES over B and (possibly empty) finite words over Σ . When $w \models r$ we say that w tightly satisfies r . The semantics of SERES are defined as follows, where w is a finite (possibly empty) word over Σ .*

- $w \not\models \emptyset$ and $w \models \lambda \iff w = \epsilon$
- Let b denote a boolean expression in B . Then, $w \models b \iff |w| = 1$ and $w^0 \models b$
- Let r , r_1 , and r_2 denote SERES over B . Then,
 1. $w \models \{r\} \iff w \models r$
 2. $w \models r_1 \cup r_2 \iff w \models r_1$ or $w \models r_2$
 3. $w \models r_1 \cdot r_2 \iff \exists w_1, w_2$ s.t. $w = w_1w_2$, $w_1 \models r_1$, and $w_2 \models r_2$
 4. $w \models r^* \iff w = \epsilon$ or $\exists w_1, w_2$ s.t. $w_2 \neq \epsilon$, $w = w_1w_2$, $w_1 \models r^*$ and $w_2 \models r$
 5. $w \models r_1 \circ r_2 \iff \exists w_1, w_2$, and s s.t. $w = w_1sw_2$, $w_1s \models r_1$, and $sw_2 \models r_2$
 6. $w \models r_1 \cap r_2 \iff w \models r_1$ and $w \models r_2$

We note that the semantics given here is a bit different than the traditional semantics given to regular expressions (and their extensions). The traditional semantics, repeated below, defines a set of words $Lng(r)$ satisfying a regular expression. This set consists of words over the same alphabet as the regular expressions themselves. While the semantics in Definition 2 relates regular expressions over one alphabet B – the set of boolean expressions over the state variables V – to words over another alphabet Σ – the set of interpretations of the state variables V .

Definition 3 (The Language of RES) *Let Γ be a finite set of symbols (an alphabet). Let b be a letter in Γ and r , r_1 , and r_2 SERES over Γ . The set $Lng(r)$, defined below,*

² Where ω denotes the cardinality of the non-negative integers.

denotes the set of words over Γ satisfying r according to the traditional semantics of regular expressions.

- $Lng(\emptyset) = \emptyset$ • $Lng(\lambda) = \{\epsilon\}$ • $Lng(b) = \{b\}$ • $Lng(r^*) = Lng(r)^*$
- $Lng(r_1 \cup r_2) = Lng(r_1) \cup Lng(r_2)$ • $Lng(r_1 \cdot r_2) = Lng(r_1)Lng(r_2)$
- $Lng(r_1 \circ r_2) = Lng(r_1) \circ Lng(r_2)$ • $Lng(r_1 \cap r_2) = Lng(r_1) \cap Lng(r_2)$

Example 4. According to Definition 2 the SERE $a \cdot b^* \cdot c$ is satisfied over any word over Σ whose first letter interprets a as *true*, whose last letter interprets c as *true* and any letter in between interprets b as *true*. Among which is for example the word $w = s_0 s_1 s_2 s_3$ with $s_0 = \{a, c\}$, $s_1 = \{b\}$, $s_2 = \{b, c\}$ and $s_3 = \{a, c\}$.

According to Definition 3 $Lng(a \cdot b^* \cdot c)$ consists of the set of words over B starting with the letter a followed by a finite number of letters b and ending with the letter c . Thus, the NFA in figure 1 accepts $Lng(a \cdot b^* \cdot c)$. ┘

Another difference between the traditional semantics and the ones given here should be noted. While in the traditional semantics *letters* of the alphabet are mutually exclusive, this is not the case for our semantics. Since in our semantics the letters are actually Boolean expressions, two different letters may hold simultaneously.

Interpreting formulas of the logic over the computational model

Definition 4. Let \mathcal{D} be a discrete transition system, and r a SERE such that $\epsilon \not\models r$. We say that \mathcal{D} satisfies the formula $\text{not } r!$, denoted $\mathcal{D} \models \text{not } r!$, iff for all finite runs σ of \mathcal{D} , $\sigma \not\models r$.

We use the syntax $\text{not } r!$ to be compliant with PSL [2]. The semantics given here to $\text{not } r!$ are equivalent to the ones given in PSL for negating a strong SERE.

To verify a $\text{not } r!$ formula, we can run a DTS representing an NFA accepting r in parallel to the given model, and check that the joint run does not reach a finite accepting state of \mathcal{D}_r .

Proposition 5. Let \mathcal{D}_M be a DTS and r an RE. Let \mathcal{D}_r be the DTS representation of an NFA accepting $Lng(r)$. Let \mathcal{A}_r be the finite acceptance condition of \mathcal{D}_r . Then,

$$\mathcal{D}_M \models \text{not } r! \iff \mathcal{D}_M \parallel \mathcal{D}_r \models \mathbf{AG} \neg \mathcal{A}_r$$

This proposition, proved in [8, Proposition 11], confirms with the observation that the DTS of r corresponds to a terminal Büchi automaton (see subsection 2.1) and the fact that emptiness of a terminal Büchi automaton reduces to checking the CTL property $\neg \mathbf{EF} A$ (see [9]).

Example 5. As stated in Example 4 the NFA of Figure 1 accepts the SERE $\{a \cdot b^* \cdot c\}$. By Proposition 5 verifying $\text{not } \{a \cdot b^* \cdot c\}!$ over a model \mathcal{D}_M reduces to verifying $\mathbf{AG} \neg (st_N = 3)$ over the parallel composition of \mathcal{D}_M with the DTS \mathcal{D}_N of Example 3. ┘

3 Embedding a DTS into an RE

In this section we provide an algorithm to embed a DTS into a SERE. Let r be a SERE, and let s be a sub-SERE of r . We construct an independent side-DTS for s . We replace the occurrence of s inside r with a *placeholder* to get $r' = r[s \leftarrow \text{placeholder}]$. We then construct a DTS for r' and derive the $\text{AG}(p)$ formula in the usual way (see Section 2). Finally, we run the two DTSS in parallel with the model. We prove that this method gives equivalent results to the one without the embedding.

The idea behind the procedure can be explained as follows. Assume we have a side-DTS for s . In order to embed a placeholder for s in r we need the DTS to start running at the place of embedding. That is, if $r = \{a \cdot b \cdot \{s\} \cdot c\}$, we would like the side-DTS to start running after two time units. If, for example, $r = \{a \cdot b^* \cdot \{s\} \cdot c\}$, then we need the side-DTS to start running at many possible time units: after one 'a' and any finite number of 'b's. Furthermore, we want the side DTS to be independent of the SERE in which it is embedded, in order to allow the reuse of it. We therefore let it start non-deterministically at any point of time. This is done by adding an idle state with a self-loop, which can transit to the initial states at any time. The place holder inserted instead of s , will then make sure we check our r' only on those paths where the side-DTS indeed starts when it is expected to.

Since the sub-SERE can be embedded in the RE more than once (e.g. $r = \{a \cdot \{s\} \cdot b^* \cdot \{s\} \cdot c\}$), or within a starred sub-SERE of r (e.g. $r = \{a \cdot \{b \cdot \{s\}\}^* \cdot c\}$), we need to allow the side-DTS to start running again after it completed a run. To achieve this, we add a transition from the predecessors of final states to the initial states.

The DTS we construct is based on an NFA that can start at any cycle, and restart after it has reached its final state. Such an NFA is called a *cruising* NFA. We define it formally as follows.

Definition 6 (Cruising NFA).

An NFA $N = \langle B, Q, Q_0, \delta, A \rangle$ is *cruising* iff the following conditions hold:

1. *Initial stuttering:* There exists an initial state $q_I \in Q_0$ such that $\forall q_0 \in Q_0$ there exists a δ -transition (q_I, true, q_0) . In the sequel we refer to q_I as the idle state.
2. *Distinguished start and repetition:* $Q_0 \setminus \{q_I\} \neq \emptyset$ and $\forall q_0 \in Q_0$ there exists a δ -transition (q, b, q_0) to q_0 with $q \neq q_I$ iff there exists a δ -transition (q, b, q_A) to some accepting state $q_A \in Q_A$.

The following proposition states that for any SERE r one can build a cruising NFA for r . Note that the cruising NFA accepts $\text{Lng}(\{\text{true}^* \cdot r\}^+)$ rather than $\text{Lng}(r)$.

Proposition 7. *Let r be a RE. There exists a cruising NFA accepting $\text{Lng}(\{\text{true}^* \cdot r\}^+)$.*

Proof Sketch. Let $N_r = \langle B, Q, Q_0, \delta, A \rangle$ be an NFA for r . Define $N'_r = \langle B, Q', Q'_0, \delta', A \rangle$ as follows.

- $Q' = Q \cup \{q_S, q_I\}$ where $q_S, q_I \notin Q$
- $Q'_0 = \{q_S, q_I\}$

$$\begin{aligned}
& \delta \cup (q_I, \text{true}, q_I) \cup (q_I, \text{true}, q_S) \cup \\
- \delta' = & \{(q_S, b, q) \mid q \in Q \text{ and } \exists q_0 \in Q_0, b \in B, \text{ such that } (q_0, b, q) \in \delta\} \cup \\
& \{(q, b, q_0) \mid q \in Q, b \in B, q_0 \in Q'_0, \text{ and } \exists q_A \in A \text{ s.t. } (q, b, q_A) \in \delta\}
\end{aligned}$$

Then N'_r is a cruising NFA for r . The correctness of the construction is given in the appendix. \square

Example 6. Applying the construction sketched in the proof of Proposition 7 on the NFA of Figure 1 results in the NFA shown in Figure 2. Note that state 1 is redundant.

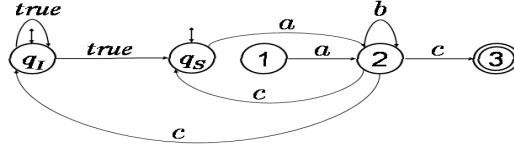


Fig. 2. A cruising NFA accepting the set $\text{Lng}(\{\text{true}^* \cdot \{a \cdot b^* \cdot c\}^+\})$

This does not harm the correctness of the algorithm. In practice, a stage removing such redundant states is added to the procedure, to generate smaller NFAs. \lrcorner

By using the DTS representation of a cruising NFA N we can construct Boolean expressions which refer to the states of N . We use such Boolean expressions to construct the “placeholder” to replace s in r . We define three conditions which state that the DTS has started and completed “working”. Formally,

Definition 8 (start,middle,end) Let $N = \langle B, Q, Q_0, \delta, A \rangle$ be a cruising NFA with idle state q_I . Let \mathcal{D}_N be its DTS representation. Define new boolean expressions *start*, *middle* and *end* over the variables of V and the variable *state* representing the state of N , as follows:

$$\begin{aligned}
\bullet \text{ start} : & \bigvee_{\{(q_0, b, q) \in \delta \mid q_0 \in Q_0 \setminus \{q_I\}\}} (\text{state} = q_0) \wedge b & \bullet \text{ middle} = \neg \text{start} & \bullet \text{ end} : \bigvee_{\{(q, b, q_A) \in \delta \mid q_A \in A\}} (\text{state} = q) \wedge b
\end{aligned}$$

Example 7. Applying Definition 8 on the NFA described in Figure 2 we get:

$$\bullet \text{ start} : (\text{st}_N = q_S) \wedge a \quad \bullet \text{ middle} = (\text{st}_N \neq q_S) \vee \neg a \quad \bullet \text{ end} : (\text{st}_N = 2) \wedge c$$

We are now ready to define the placeholder for a given sub-SERE.

Definition 9 Let r be a SERE and t be a sub-SERE of r . Let \mathcal{D}_N be a cruising DTS for t . Let *start*, *middle*, *end* be as in Definition 8. Define the SERE $t' = \text{placeholder}(t, \text{start}, \text{middle}, \text{end})$ as follows:

$$t' = \begin{cases} \text{start} \wedge \text{end} \cup \{\text{start} \cdot \text{middle}^* \cdot \text{end}\} & \text{if } \mathcal{S}(t) = \text{false} \\ \lambda \cup \text{start} \wedge \text{end} \cup \{\text{start} \cdot \text{middle}^* \cdot \text{end}\} & \text{Otherwise} \end{cases}$$

Example 8. Applying Definition 9 to Example 7 we obtain the following definition for the SERE defining the placeholder p for $\{a \cdot b^* \cdot c\}$.

$$p := \{((st_N = q_S) \wedge a) \cdot ((st_N \neq q_S) \vee \neg a)^* \cdot ((st_N = 2) \wedge c)\}$$

This, since the first disjunct $((st_N = q_S) \wedge a) \wedge ((st_N = 2) \wedge c)$ is equivalent to *false*. \lrcorner

We claim that if we let $\mathcal{D}_{N'}$ run in parallel to the model, then t' can be “plugged” instead of t in (almost) any SERE r containing t as a sub-SERE. The only cases where this claim does not hold, are when t appears in both operands of the intersection or fusion operators (since these operators have a parallel nature). Formally, we claim the following.

Theorem 10. *Let \mathcal{D}_M be a DTS, r a SERE and t a sub-SERE of r such that for every intersection and fusion operator, t is not a sub-SERE of both operands. Let the DTS \mathcal{D}_t be a cruising DTS for t . Then,*

$$\mathcal{D}_M \models \text{not } r! \iff \mathcal{D}_M \parallel \mathcal{D}_t \models \text{not } r[t \leftarrow \text{placeholder}(t, \text{start}, \text{middle}, \text{end})]!$$

Example 9. According to Theorem 10 verifying $\text{not } \{d \cdot e^* \cdot a \cdot b^* \cdot c \cdot f\}!$ over a given model \mathcal{D}_M is equivalent to verifying $\text{not } \{d \cdot e^* \cdot p \cdot f\}!$ (with p as define in Example 8) over the parallel composition of the model \mathcal{D}_M with the model \mathcal{D}_N of Example 3. \lrcorner

Corollary 11. *Let \mathcal{D}_M be a DTS, r a SERE and t a sub-SERE of r such that for every intersection and fusion operator, t is not a sub-SERE of both operands. Let the DTS \mathcal{D}_t be a cruising DTS for t . Finally, let t' be the placeholder defined above and let r' be $r[t \leftarrow t']$. Then,*

$$\mathcal{D}_M \models \text{not } r! \iff \mathcal{D}_M \parallel \mathcal{D}_t \parallel \mathcal{D}_{r'} \models \text{AG } \neg \mathcal{A}_{r'}$$

Example 10. Let r be the SERE $\{d \cdot e^* \cdot a \cdot b^* \cdot c \cdot f\}$. Then, $r' = \{d \cdot e^* \cdot \text{start} \cdot \text{middle}^* \cdot \text{end} \cdot f\}$ where *start*, *middle* and *end* are as defined in Example 7. Using only the original state variables $\{a, b, c, d, e, f, st_N\}$ we get $r' = \{d \cdot e^* \cdot ((st_N = q_S) \wedge a) \cdot ((st_N \neq q_S) \vee \neg a)^* \cdot ((st_N = 2) \wedge c) \cdot f\}$. The NFA $N_{r'}$ for r' is described in Figure 3.

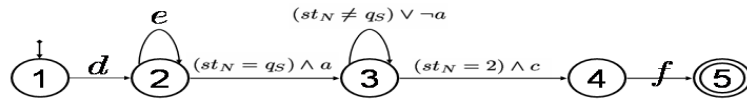


Fig. 3. An NFA accepting the set $Lng(r')$ as in Example 10

The DTS $\mathcal{D}_{r'} = \langle V_{r'}, \Theta_{r'}, \rho_{r'}, \mathcal{A}_{r'}, \mathcal{J}_{r'} \rangle$ described below corresponds to $N_{r'}$, where *state* is a new variable with domain $\{0, 1, 2, 3, 4, 5\}$.

$$\begin{aligned}
V_{r'} &= \{a, b, c, d, e, f, st_N, state\}; \quad \Theta_{r'} = (state = 1); \quad \mathcal{A}_{r'} = (state = 5); \\
\mathcal{J}_{r'} &= \{(state = 5)\}; \quad \text{and } \rho_{r'} = \\
&(state = 1 \wedge d \wedge state' = 2) \vee (state = 1 \wedge \neg d \wedge state' = 0) \vee \\
&(state = 2 \wedge e \wedge state' = 2) \vee (state = 2 \wedge \neg e \wedge state' = 0) \vee \\
&(state = 2 \wedge ((st_N = q_S) \wedge a) \wedge state' = 3) \vee (state = 2 \wedge \neg((st_N = q_S) \wedge a) \wedge state' = 0) \vee \\
&(state = 3 \wedge ((st_N \neq q_S) \vee \neg a) \wedge state' = 3) \vee (state = 3 \wedge \neg((st_N \neq q_S) \vee \neg a) \wedge state' = 0) \vee \\
&(state = 3 \wedge ((st_N = 2) \wedge c) \wedge state' = 4) \vee (state = 3 \wedge \neg((st_N = 2) \wedge c) \wedge state' = 0) \vee \\
&(state = 4 \wedge f \wedge state' = 5) \vee (state = 4 \wedge \neg f \wedge state' = 0) \vee \\
&(state = 5 \wedge state' = 5) \vee (state = 0 \wedge state' = 0)
\end{aligned}$$

Thus, according to Corollary 11 verifying not $\{d \cdot e^* \cdot \{a \cdot b^* \cdot c \cdot f\}^* \cup \{g^* \cdot h\}\}!$ over a given model \mathcal{D}_M is equivalent to verifying **AG** $\neg(state = 5)$ over the parallel composition of the model \mathcal{D}_M with the model \mathcal{D}_N of Example 3 and the DTS $D_{r'}$ defined above. ┘

Note that information about the embedded SERE can be used to simplify the placeholder and thus reduce the size of the overall automata. For example, when the SERE has no computations of length one it is possible to remove the disjunct $\text{start} \wedge \text{end}$ from the definition of the placeholder. Similarly, information about the embedding SERES can be used to simplify the cruising DTS. For example, if the SERE is not embedded more than once nor within a starred sub-SERE, it is possible to build a side-DTS which does not confirm to the condition of repetition.

In the next section we demonstrate the benefits of using the embedding method, and give several applications.

4 Benefits of the Embedding Algorithm

We use the embedding algorithm in several different applications, each of which benefits from the use of it in a different way. In section 4.1 we show how *modularity* is achieved for the *named sequence* construct of PSL, and discuss the implementation of SERE intersection. In section 4.2 we demonstrate how the *size* of the automaton built for the formula can be reduced when using embedding, and in section 4.3 we show how *expressiveness* is gained when SERES are to be translated to CTL.

4.1 Modularity and Reuse

The embedding mechanism described above allows us to follow the guideline of software engineering and reuse a given automaton of one SERE for constructing automata for more complicated SERES. Reusing existing code allows building “libraries” of automata, which can fasten implementation and increase code stability. Using the existing automata as is allows preserving useful automata characteristics.

As an example to code reuse, consider the *named sequences* mechanism of PSL/Sugar [2]. In this language, a SERE can be given a name, and then be used as a sub-SERE in many formulas. For example, one can define: sequence $r := \{a \cdot b^* \cdot c\}$; and then use it in other formulas: not $\{d^* \cdot r\}!$, not $\{a \cdot r \cdot f\}!$ etc. When constructing DTSS for the above

formulas, 3 states would be produced for each occurrence of r . Using the embedding mechanism, we can construct a DTS for r just once, and then reuse it as many times as needed.

Another example is implementing SERE intersection. Unwrapping an intersection operation at the SERE level is very complicated [3]. Instead, we use the embedding algorithm to implement intersection. By the general embedding method, we can construct an NFA for any sub-SERE of the form $r_1 \cap r_2$ and plug the placeholder for its DTS instead. An NFA for $r_1 \cap r_2$ can be built using the product construction [10]. However, in order to simplify the construction, the use of embedding for intersection can be extended even farther. We provide a method to embed *two* DTS, one for each operand of the intersection, replacing them with a single place-holder which serves for both of them together. This allows us to implement intersection using the same type of DTS, making it easier to maintain.

Formally, let $r_1 \cap r_2$ be a sub-SERE of r . Let N'_1 and N'_2 be cruising NFA for r_1 and r_2 respectively, constructed as in Section 3. Let $D_{N'_1}$ and $D_{N'_2}$ be their DTS representations. Define start_i , end_i for $i \in \{1, 2\}$ as the respective Boolean expressions stating that $D_{N'_i}$ is in the real start (resp. end) of its run (exactly as in the general embedding method). Define the new Boolean expressions start and end as the conjunctions $\text{start}_1 \wedge \text{start}_2$, and $\text{end}_1 \wedge \text{end}_2$, respectively. Define the new boolean expression middle as the conjunction $\neg \text{start}_1 \wedge \neg \text{start}_2$. We claim that the placeholder defined using the above start , middle and end can be plugged instead of $r_1 \cap r_2$ while running the model in parallel to both $D_{N'_1}$ and $D_{N'_2}$.

Proposition 12. *Let \mathcal{D}_M be a DTS, r a SERE and $r_1 \cap r_2$ a sub-SERE of r which is not a sub-SERE of both operands of \circ or \cap . Let N'_1 and N'_2 be the cruising NFAs constructed for r_1 and r_2 , respectively. Let start , middle and end be as defined above.*

$$\begin{aligned} \mathcal{D}_M &\models \text{not } r! \\ &\Updownarrow \\ \mathcal{D}_M \parallel \mathcal{D}_{N'_1} \parallel \mathcal{D}_{N'_2} &\models \text{not } r[r_1 \cap r_2 \leftarrow \text{placeholder}(r_1 \cap r_2, \text{start}, \text{middle}, \text{end})]! \end{aligned}$$

4.2 Efficiency of Verification

In some cases, the use of embedding allows us to build a smaller automaton for a given formula, thus reducing the size of the model to be verified.

A simple example is the operator *counters* ($p[= i..j]$). *Counters* can be viewed as abbreviations of the basic SERE operators as follows, where b denotes a boolean expression, i and j denote integer constants such that $i \geq 0$ and $j \geq i$.

$$\bullet b[= i] \stackrel{\text{def}}{=} \overbrace{\{-b^* \cdot b\} \cdot \dots \cdot \{-b^* \cdot b\}}^{i \text{ times}} \cdot \neg b^* \quad \bullet b[= i..j] \stackrel{\text{def}}{=} b[= i] \cup \dots \cup b[= j]$$

A straightforward unwrapping of a counter operator $p[= i..j]$ at the SERE level, using the definition above, will result in a SERE of size quadratic in j . Using other SERE-level unwrapping methods we can reduce the size of the resulting SERE. However, the best result is achieved when directly building an NFA for a counter. We show how to build an NFA for a counter $p[= i..j]$, which has $j + 2$ states. Given such an NFA, the

embedding algorithm can then be used to embed the NFA into the SERE. Below we give the construction of an NFA for a counter, consisting of $j + 2$ states. We then present experimental results which compare the two ways of translation.

An NFA for $p[=i..j]$

Let p be a boolean expression, i, j integers such that $0 \leq i \leq j$ and $B = \{p, \neg p\}$.

$$Lng(\{p[=i..j]\}) = \{w = w_0 \dots w_n \in B \mid i \leq |\{k \mid w_k = p\}| \leq j\}$$

The following is an NFA that accepts $Lng(p[=i..j])$. The automaton has ‘counting’ states q_0, q_1, \dots, q_{j+1} . The idea is that state q_k indicates that up until now k p ’s were read. Thus, all states q_k such that $i \leq k \leq j$ are accepting states.

Formally, define $N_{p[=i..j]} = \langle B, Q, Q_0, \delta, A \rangle$ where

$Q = \{q_k \mid 0 \leq k \leq j + 1\}$; $Q_0 = \{q_0\}$; $A = \{q_k \mid i \leq k \leq j\}$ if $i \neq 0$; and

$$\delta = \{(q_k, p, q_{k+1}) \mid 0 \leq k \leq j\} \cup \{(q_k, \neg p, q_k) \mid 0 \leq k \leq j\}$$

Proposition 13. *The NFA $N_{p[=i..j]}$ accepts $Lng(p[=i..j])$.*

$N_{p[=i..j]}$ is an NFA for r . Thus, by Proposition 10 we can embed the placeholder for $p[=i..j]$ in any SERE containing it, and run both DTS s in parallel with the model.

Experimental Results

We present experimental results of embedding different *counters* formulas. We compare the traditional unwrapping method (see above) to our embedding method, in two aspects: the size of automata built for a formula, and the verification time. The verification time is influenced by the overall size of the model, which includes the design under test as well as the automata built for the formulas. Thus we can see a significant benefit for the embedding method when the design is small, and a smaller improvement when the design under test is large. In Table 1 we present the results of running formulas of the type $b[=i..j]!$ both on a small model of 36 state variables, and on a larger one of 196 state variables (Both numbers present the size of the model *after* model reductions). The experiments were done on Dual 2.4 Ghz Pentium 4 (xeon) with 2.4GB RAM.

In this table, the ‘states’ presented are the number of states in the automaton built for the formula, and time is given in seconds. As can be seen, the embedding algorithm performs better in all cases. The improvement in performance becomes more significant though, as the formula becomes larger.

4.3 Increasing Expressibility

As discussed earlier, not $r!$ formulas are usually verified by constructing an automaton. However, it is sometimes beneficial to translate such formulas to CTL, so that they can be verified using symbolic model checking algorithms.

Given an unwrapped not SERE! formula, the basic operators of regular expressions can be translated to CTL as described below. These include concatenation, union, and

Table 1. Unwrapped Counters vs. Embedded Counters

Formula	unwrap states	unwrap time	embed states	embed time
Small Model				
$p[= 1..10]$	55	5.15	13	1.25
$p[= 10..20]$	165	8.1	23	1.92
$p[= 20..25]$	135	7.85	28	4.8
$p[= 25..30]$	165	10.43	33	6.26
$p[= 25..40]$	455	60.59	43	6.62
Large Model				
$p[= 1..10]$	55	1.3	13	1.25
$p[= 10..20]$	165	1766.03	23	1419.63
$p[= 20..25]$	135	3928.4	28	3188.72
$p[= 25..30]$	165	5105.53	33	3992.37
$p[= 25..40]$	455	12038	43	9397

Kleene closure (a star) on a single letter (e.g. a^*). However, a star over a sub-SERE (e.g. not $\{a \cdot \{b \cdot c \cdot d\}^* \cdot e\}!$) cannot be translated into CTL (see [15]).

In order to support a starred sub-sere when translating a not $r!$ formula into CTL, we use the embedding algorithm. We take out every starred sub-SERE, build a side-DTS for it, and replace it with a simple placeholder, as described in section 3. The formula obtained does not have a starred SERE, and therefore can be translated to CTL.

Definition 14 (Translation procedure). Let r_1, r_2, r be regular expressions, and $b \in B$ be a boolean expression. We define the function T as follows:

$$\begin{aligned}
- T(b \cdot r) &= b \wedge EX(T(r)) & - T(b) &= b \\
- T(b^* \cdot r) &= E[b \cup T(r)] \vee T(r) & - T(r \cdot b^*) &= T(r) \\
- T(r_1 \cup r_2 \cdot r) &= T(r_1 \cdot r) \vee T(r_2 \cdot r) & - T(r_1 \cup r_2) &= T(r_1) \vee T(r_2)
\end{aligned}$$

Proposition 15. Let \mathcal{D}_M be a discrete system and r a SERE with no starred sub-SERES, such that $\epsilon \not\models r$. Then, $\mathcal{D}_M \models \text{not } r! \iff \mathcal{D}_M \models \neg T(r)$.

5 Conclusions

We have revisited the translation of extended regular expressions (SERES) into an automaton, a translation the vast majority of the specifications verified using the IBM model checking tool-set RuleBase [5] go through. We have presented an *embedding* algorithm, which allows an automaton to be embedded into a SERE. Thus, instead of producing one automaton per formula, the translation using embedding may consist of several automata, running in parallel with the model. An important feature of this algorithm is that the auxiliary automata built are independent of the original SERE, and thus can be used more than ones.

Various applications of the embedding algorithm were given, each of which benefits from it in a different way. For *named sequences*, the embedding algorithm allows reuse of the automata code. For *intersection*, the use of embedding allows us to enjoy the same NFA construction, avoiding the need to unwrap at the SERE level or to intersect at the automata level. For *counters* we manage to reduce the size of the automata built for the

formula, and for CTL *translation* we extend the expressive power of the language. The variability of these applications demonstrates its potential, and suggests, we believe, that more applications may be found for it.

Automata for SERE formulas are sometimes used as checkers for simulation [1]. For that purpose, the automaton must be deterministic. Since the embedding algorithm relies strongly on the non-deterministic nature of the side-NFA (the side-NFA can start at any cycle), it should be carefully examined whether the embedding algorithm performs well for simulation checkers, as it does for model checking.

Acknowledgment

We would like to thank Cindy Eisner, Orna Lichtenstein and Avigail Orni for their helpful comments on an early draft of this paper.

References

1. Y. Abarbanel, I. Beer, L. Gluhovsky, S. Keidar, and Y. Wolfsthal. FoCs - automatic generation of simulation checkers from formal specifications. In *Proc. 12th International Conference on Computer Aided Verification (CAV)*, LNCS 1855. Springer-Verlag, 2000.
2. Accellera. Accellera property language reference manual. In <http://www.eda.org/vfv/docs/PSL-v1.1.pdf>, pages 109–117, June 2004. Appendix B.
3. V. M. Antimirov and P. D. Mosses. Rewriting extended regular expressions. *Theoretical Computer Science*, 143(1):51–72, 1995.
4. R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M. Y. Vardi, and Y. Zbar. The forspec temporal logic: A new temporal property-specification language. In *TACAS*, pages 296–211, 2002.
5. I. Beer, S. Ben-David, C. Eisner, D. Geist, L. Gluhovsky, T. Heyman, A. Landver, P. Paanah, Y. Rodeh, G. Ronin, and Y. Wolfsthal. RuleBase: Model checking at IBM. In *Proc. 9th International Conference on Computer Aided Verification (CAV)*, LNCS 1254, pages 480–483. Springer-Verlag, 1997.
6. I. Beer, S. Ben-David, and A. Landver. On-the-fly model checking of RCTL formulas. In *Proc. 10th International Conference on Computer Aided Verification (CAV'98)*, LNCS 1427, pages 184–194. Springer-Verlag, 1998.
7. S. Ben-David, C. Eisner, D. Geist, and Y. Wolfsthal. Model checking at ibm. *Formal Methods in System Design*, 22(2):101–108, 2003.
8. S. Ben-David, D. Fisman, and S. Ruah. Automata construction for regular expressions in model checking, June 2004. IBM research report H-0228.
9. R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In *Computer Aided Verification*, pages 222–235, 1999.
10. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Series in Computer Science. Addison-Wesley, 1979.
11. Y. Kesten, A. Pnueli, and L. Raviv. Algorithmic verification of linear temporal logic specifications. In K. Larsen, S. Skyum, and G. Winskel, editors, *Proc. 25th Int. Colloq. Aut. Lang. Prog.*, volume 1443 of *Lect. Notes in Comp. Sci.*, pages 1–16. Springer-Verlag, 1998.
12. M. Maidl. The common fragment of CTL and LTL. In *IEEE Symposium on Foundations of Computer Science*, pages 643–652, 2000.
13. K. McMillan. Symbolic model checking, 1993.
14. M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, Berlin, 1996.
15. P. Wolper. Temporal logic can be more expressive. In *22nd Annual Symposium on Foundation of Computer Science*, October 1981.

A Proofs

A.1 Proofs of Section 3

Proposition 7 Let r be an RE. There exists a cruising NFA accepting $Lng(\{true * r\}^+)$.

Proof. Let $N_r = \langle B, Q, Q_0, \delta, A \rangle$ be an NFA for r . Define $N'_r = \langle B, Q', Q'_0, \delta', A \rangle$ as follows.

- $Q' = Q \cup \{q_S, q_I\}$ where $q_S, q_I \notin Q$
- $Q'_0 = \{q_S, q_I\}$
- $\delta \cup (q_I, true, q_I) \cup (q_I, true, q_S) \cup$
- $\delta' = \{(q_S, b, q) \mid q \in Q \text{ and } \exists q_0 \in Q_0, b \in B, \text{ such that } (q_0, b, q) \in \delta\} \cup$
 $\{(q, b, q_0) \mid q \in Q, b \in B, q_0 \in Q'_0, \text{ and } \exists q_A \in A \text{ s.t. } (q, b, q_A) \in \delta\}$

It is easy to see that q_I is the initial stuttering state, q_S is the distinguished start state, and that together with the transition relation the conditions of Definition 6 are satisfied. Therefore N'_r is cruising. It remains to show N'_r accepts $Lng(\{true^* \cdot r\}^+)$. Let $w \in Lng(\{true^* \cdot r\}^+)$. The proof is by induction on the structure of w . If $w = uv$ where $u \in Lng(true^*)$ and $v = v^0 \dots v^{n_1-1} \in Lng(r)$ then there exists a run

$$q_0 \xrightarrow{v^0} q_1 \xrightarrow{v^1} q_2 \cdots q_{n_1-1} \xrightarrow{v^{n_1-1}} q_{n_1}$$

of N_r accepting v . If $|u| = 0$ then by the transition relation of N'_r

$$q_S \xrightarrow{v^0} q_1 \xrightarrow{v^1} q_2 \cdots q_{n_1-1} \xrightarrow{v^{n_1-1}} q_{n_1}$$

is a run of N'_r accepting w . If $u = true^{m_1}$ where $m_1 \geq 1$

$$\underbrace{q_I \xrightarrow{true} q_I \cdots q_I \xrightarrow{true} q_S}_{m_1 \text{ times}} \xrightarrow{v^0} q_1 \xrightarrow{v^1} q_2 \cdots q_{n_1-1} \xrightarrow{v^{n_1-1}} q_{n_1}$$

is a run of N'_r accepting w . Assume the claim holds for $w_1 \in Lng(\{true^* \cdot r\}^k)$ where $k \geq 1$. Let $w = w_1 w v$ where $u \in Lng(\{true^*\})$ and $v \in Lng(r)$. Let

$$p_0 \xrightarrow{w_1^0} p_1 \xrightarrow{w_1^1} p_2 \cdots p_{n-1} \xrightarrow{w_1^{n-1}} p_n$$

be an accepting run of N'_r over $w_1 = w_1^0 \dots w_1^{n-1}$ and

$$q_0 \xrightarrow{v^0} q_1 \xrightarrow{v^1} q_2 \cdots q_{n_1-1} \xrightarrow{v^{n_1-1}} q_{n_1}$$

an accepting run of N_r over $v = v^0 \dots v^{n_1-1}$.

If $|u| = 0$ then

$$p_0 \xrightarrow{w_1^0} p_1 \xrightarrow{w_1^1} p_2 \cdots p_{n-1} \xrightarrow{w_1^{n-1}} q_S \xrightarrow{v^0} q_1 \xrightarrow{v^1} q_2 \cdots q_{n_1-1} \xrightarrow{v^{n_1-1}} q_{n_1}$$

is a run of N'_r accepting w . If $|u| > 0$ and $u = \text{true}^{m_1}$

$$p_0 \xrightarrow{w_1^0} p_1 \xrightarrow{w_1^1} p_2 \cdots p_{n_1-1} \xrightarrow{w_1^{n_1-1}} \underbrace{q_I \xrightarrow{\text{true}} q_I \cdots q_I \xrightarrow{\text{true}}}_{m_1 \text{ times}}$$

$$q_S \xrightarrow{v^0} q_1 \xrightarrow{v^1} q_2 \cdots q_{n-1} \xrightarrow{v^{n_2-1}} q_{n_2}$$

is a run of N'_r accepting w .

For the other direction, assume σ is an accepting run of N'_r over w . Let $\text{pred}(A)$ denote the set $\{q \in Q \mid \exists b \in B, q_A \in Q_A, \text{ s.t. } (q, b, q_A) \in \delta\}$. By the transition relation, σ is of the form $\sigma = \bar{\sigma}_1 \dots \bar{\sigma}_k$ for some $k \geq 1$ where for every $i \leq k$, $\bar{\sigma}_i$ is in one of the following forms:

1.

$$q_S \xrightarrow{v_i^0} q_1^i \cdots q_{n_i-1}^i \xrightarrow{v_i^{n_i-1}} q_{n_i}^i$$

2.

$$\underbrace{q_I \xrightarrow{\text{true}} q_I \cdots q_I \xrightarrow{\text{true}}}_{m_i \text{ times}} q_S \xrightarrow{v_i^0} q_1^i \cdots q_{n_i-1}^i \xrightarrow{v_i^{n_i-1}} q_{n_i}^i$$

$q_{n_i-1}^i \in \text{pred}(A)$, and $q_{n_k} \in A$. The rest of the proof is for case (2). The proof for case 1 is similar. Define

$$w_k = \text{true}^{m_1} v_1^0 \dots v_1^{n_1-1} \dots u_k^0 \text{true}^{m_k} v_k^0 \dots v_k^{n_k-1}$$

(the word accepted by σ). We show $w_k \in \text{Lng}(\{\text{true}^* \cdot r\}^k)$ by induction on k . If $k = 1$, then

$$\underbrace{q_I \xrightarrow{\text{true}} q_I \cdots q_I \xrightarrow{\text{true}}}_{m_1 \text{ times}} q_S \xrightarrow{v^0} q_1^1 \cdots q_{n_1-1}^1 \xrightarrow{v^{n_1-1}} q_{n_1}^1$$

and $w_1 = \text{true}^{m_1} v^0 \dots v^{n_1-1}$. By the transition relation $\exists q_0 \in Q_0$ such that $(q_0, v^0, q_1^1) \in \delta$ and $\forall 1 \leq i \leq n_1 - 1, (q_i^1, v^i, q_{i+1}^1) \in \delta$ so $q_0, \dots, q_{n_1}^1$ is an accepting run of N_r over $v^0 \dots v^{n_1-1}$, therefore $v^0 \dots v^{n_1-1} \in \text{Lng}(r)$ and $w_1 \in \text{Lng}(\text{true}^* \cdot r)$. Assume that if $\sigma = \bar{\sigma}_1 \dots \bar{\sigma}_k$ is an accepting run over w_k then $w_k \in \text{Lng}(\{\text{true}^* \cdot r\}^k)$. Let $\bar{\sigma}_k = \hat{\sigma}_k \xrightarrow{v_k^{n_k-1}} q_{n_k}^k$. That is, $\hat{\sigma}_k$ is the prefix of $\bar{\sigma}_k$ obtained by chopping the last state of $\bar{\sigma}_k$. The last state of $\hat{\sigma}_k$ is $q_{n_k-1}^k \in \text{pred}(A)$.

Let

$$\sigma^{k+1} = \bar{\sigma}_1 \dots \bar{\sigma}_{k-1} \hat{\sigma}_k \xrightarrow{v_k^{n_k-1}} \underbrace{q_I \xrightarrow{\text{true}} q_I \cdots q_I \xrightarrow{\text{true}}}_{m_{k+1} \text{ times}}$$

$$q_S \xrightarrow{v_{k+1}^0} q_1^{k+1} \cdots q_{n_{k+1}-1}^{k+1} \xrightarrow{v_{k+1}^{n_{k+1}-1}} q_{n_{k+1}}^{k+1}$$

be an accepting run of N'_r over $w_{k+1} = w_k \cdot \text{true}^{m_{k+1}} v_{k+1}^0 \dots v_{k+1}^{n_{k+1}-1} \cdot q_{n_k-1}^k$ and there exists $q \in \{q_I, q_S\}$ such that $(q_{n_k-1}^k, v_k^{n_k-1}, q) \in \delta'$. By the definition of δ' , there exists $q_{n_k}^k \in A$ such that $(q_{n_k-1}^k, v_k^{n_k-1}, q_{n_k}^k) \in \delta$. Therefore

$$\sigma' = \bar{\sigma}_1 \dots \bar{\sigma}_{k-1} \hat{\sigma}_k \xrightarrow{v_k^{n_k-1}} q_{n_k}^k$$

is an accepting run of N'_r over w_k . By the induction hypothesis $w_k \in \text{Lng}(\{\text{true}^*r\}^k)$.

$$q_I \xrightarrow{\text{true}} q_I \dots q_I \xrightarrow{\text{true}} q_S \xrightarrow{v_{k+1}^0} q_1^{k+1} \dots q_{n_{k+1}-1}^{k+1} \xrightarrow{v_{k+1}^{n_{k+1}-1}} q_{n_{k+1}}^{k+1}$$

$\underbrace{\hspace{10em}}_{m_{k+1} \text{ times}}$

is an accepting run of N'_r over $u_{k+1}^0 u_{k+1}^1 \dots u_k^{m_{k+1}-1} v_k^1 \dots v_{k+1}^{n_{k+1}-1}$ by the induction hypothesis

$$\text{true}^{k+1} \cdot v_{k+1}^0 \dots v_{k+1}^{n_{k+1}-1} \in \text{Lng}(\{\text{true}^*r\})$$

and $w \in \text{Lng}(\{\text{true}^*r\}^{k+1})$. \square

Proof of Theorem 10

We make use of the following notations and assumptions in the proof of Theorem 10, and its lemmas (Lemmas 16-18).

Let r be a SERE over B , and t a sub-SERE of r such that for every intersection and fusion operator, t is not a sub-SERE of both operands. Let $N = \langle B, Q, Q_0, \delta, A \rangle$ be an NFA for t and $N' = \langle B, Q', Q'_0, \delta', A' \rangle$ the cruising NFA constructed from N (as defined in Proposition 7). Let $\mathcal{D}_{N'}$ be the DTS of N' and let *start* and *end* be the boolean expressions defined in Section 3. Let t' be the corresponding placeholder. Define *idle* to be the boolean expression asserting $\text{state} = q_I$.

We use s (possibly with subscripts) to denote states/boolean expressions in a run of a DTS. We use σ (possibly with subscripts) to denote sequences of states/boolean expressions in a run of a DTS. We use b (possibly with subscripts) to denote letters/boolean expressions and w (possibly with subscripts) to denote words i.e. sequences of boolean expressions.

Lemma 16. *Let σ'_1, σ'_2 be runs of $\mathcal{D}_{N'}$ satisfying $(\text{idle} \vee \text{start}) \circ \text{true}^* \circ (\text{end} \vee \text{idle})$ as well as w_1 and w_2 , respectively. Then $\sigma'_1 \sigma'_2$ is a run of $\mathcal{D}_{N'}$ satisfying $(\text{idle} \vee \text{start}) \circ \text{true}^* \circ (\text{end} \vee \text{idle})$ as well as $w_1 w_2$.*

Proof. A run of $\mathcal{D}_{N'}$ satisfying $(\text{idle} \vee \text{start}) \circ \text{true}^* \circ (\text{end} \vee \text{idle})$ corresponds to a run of N' starting with state q_I or state $q_0 \in Q_0$ and ending in state $q \in \text{pred}A$ or in state q_I . Denote by σ_1, σ_2 the runs of N' corresponding to σ'_1, σ'_2 respectively. σ_2 starts in $q_0 \in Q'_0$. Let b_0 be the first letter of w_2 , then there is a transition from $q \in \text{pred}A \cup \{q_i\}$ to q_0 via b_0 thus the concatenation $\sigma_1 \sigma_2$ is possible. The run $\sigma'_1 \sigma'_2$ starts with a state s_0 such that $s_0[\text{state}] = q_0 \in Q_0$ or $s_0[\text{state}] = q_I$ and ends in a state s_n such that $s_n[\text{state}] \in \text{pred}A \cup \{q_I\}$. It thus satisfies $(\text{idle} \vee \text{start}) \circ \text{true}^* \circ (\text{end} \vee \text{idle})$. Clearly if σ'_1 satisfies w_1 and σ_2 satisfies w_2 , the concatenated run $\sigma'_1 \sigma'_2$ satisfies the concatenated word $w_1 w_2$. \square

Lemma 17.

- Let $\sigma'_1 s_1$ be a run of $\mathcal{D}_{N'}$ satisfying $(\text{idle} \vee \text{start}) \circ \text{true}^* \circ (\text{end} \vee \text{idle})$ as well as $w_1 b$. Let $s_2 \sigma'_2$ be a run of $\mathcal{D}_{N'}$ satisfying (idle^*) as well as $b w_2$. Then $\sigma'_1 s_1 \sigma'_2$ is a run of $\mathcal{D}_{N'}$ satisfying $(\text{idle} \vee \text{start}) \circ \text{true}^* \circ (\text{end} \vee \text{idle})$ as well as $w_1 b w_2$.

- Let $\sigma'_1 s_1$ be a run of $D_{N'}$ satisfying (idle^*) as well as $w_1 b$. Let $s_2 \sigma'_2$ be a run of $D_{N'}$ satisfying $(\text{idle} \vee \text{start}) \circ \text{true}^* \circ (\text{end} \vee \text{idle})$ as well as $b w_2$. Then $\sigma'_1 s_2 \sigma'_2$ is a run of $D_{N'}$ satisfying $(\text{idle} \vee \text{start}) \circ \text{true}^* \circ (\text{end} \vee \text{idle})$ as well as $w_1 b w_2$.

Proof. We show only the first case, the second case is symmetric. The run $\sigma'_1 s_1$ of $D_{N'}$ corresponds to a run σq_1 of N' where $s_1[\text{state}] = q_1$ and $\sigma_1 q_1$ starts with a state $q_0 \in Q_0 \cup \{q_I\}$ and ends in state $q \in \text{pred}(A) \cup \{q_I\}$. The run $s_2 \sigma'_2$ of $D_{N'}$ corresponds to a run $q_I \sigma_2$ of N' of the form q_I^* . Since from every $q \in \text{pred}(A) \cup \{q_I\}$ there are transitions to q_I , we can concatenate the second run, after chopping its first state to the end of the first run. The resulting run $\sigma_1 q_1 \sigma_2$, thus, starts with a state $q_0 \in Q_0 \cup \{q_I\}$ and ends in a state q_I (or in state $q \in \text{pred}(A)$, if $|\sigma_2| = 0$). Therefore $\sigma'_1 s_1 \sigma'_2$ satisfies $(\text{idle} \vee \text{start}) \circ \text{true}^* \circ (\text{end} \vee \text{idle})$. Clearly the concatenated run $\sigma'_1 s_1 \sigma'_2$ satisfies the concatenated word $w_1 b w_2$. \square

Lemma 18. Let r''' denote the SERE $(\text{idle} \vee \text{start}) \circ r[t \leftarrow t'] \circ (\text{end} \vee \text{idle})$. Let r'' denote the SERE r''' if $\mathcal{S}(r) = \text{false}$ and the SERE $\lambda \cup r'''$ otherwise.

$$w \models r$$

$$\Updownarrow$$

there exists a run of $\mathcal{D}_{N'}$ satisfying w which satisfies also r'' .

Proof. The proof is by induction on the structure of r with respect to t . The base case is $r = t$. The induction step can be decomposed into 7 cases, where r_1 and r_2 are SERES such that t may be a sub-SERE of them and n_1 and n_2 are SERES such that t is not a sub-SERE of them:

$$r = r_1 \cdot r_2 \quad r = r_1 \cup r_2 \quad r = r_1^* \quad r = r_1 \circ n_2 \quad r = r_1 \cap n_2$$

$$r = n_1 \circ r_2 \quad r = n_1 \cap r_2$$

Denote $r_1[s \leftarrow s']$ and $r_2[s \leftarrow s']$ by r'_1 and r'_2 respectively.

- Base case: $r = t$

$$w \models r$$

\iff [By correctness of N as an NFA recognizing $\text{Lng}(t)$]

If $\mathcal{S}(r) = \text{false}$ There exists $\hat{w} \in \text{Lng}(r)$ and an accepting run $q_0 q_1 \dots q_n$ of N over \hat{w} such that for each $\bar{q}_0 \in \{q_I, q_S\}$ and $\bar{q}_n \in Q'_0 \cup \{q_n\}$ there exists a run $w' = s_0 s_1 \dots s_n$ of $D_{N'}$ satisfying w such that

1. $s_0[\text{state}] = \bar{q}_0$
2. $\forall 1 \leq i \leq n-1 : s_i[\text{state}] = q_i$.
3. $s_n[\text{state}] = \bar{q}_n$.

\iff [By the definitions of idle , start and end]

If $\mathcal{S}(r) = \text{false}$ then there exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies

$$\{\text{start} \wedge \text{end}\} \cup \{\text{start} \cdot \neg \text{start}^* \cdot \text{end}\}$$

otherwise there exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies

$$\lambda \cup \{\text{start} \wedge \text{end}\} \cup \{\text{start} \cdot \neg \text{start}^* \cdot \text{end}\}$$

- \iff There exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies $t''' = (\text{idle} \vee \text{start}) \circ t' \circ (\text{end} \vee \text{idle})$ (if $\mathcal{S}(r) = \text{false}$ and $\lambda \cup t'''$ otherwise).
- \iff There exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies r''

– Induction step:

1. $r = r_1 \cdot r_2$

$w \models r_1 \cdot r_2$

\iff There exist w_1, w_2 s.t. $w = w_1 w_2$, $w_1 \models r_1$ and $w_2 \models r_2$

\iff [By the inductive hypothesis]

There exist w_1, w_2 s.t. $w = w_1 w_2$, there exists a run w'_1 of $\mathcal{D}_{N'}$ satisfying w_1 which also satisfies r''_1 , and there exists a run w'_2 of $\mathcal{D}_{N'}$ satisfying w_2 which also satisfies r''_2

\iff [By Lemma 16]

There exists w_1, w_2 s.t. $w = w_1 w_2$ and there exists a run $w' = w'_1 w'_2$ of $\mathcal{D}_{N'}$ satisfying $w_1 w_2$ which also satisfies $r''_1 \cdot r''_2$

\iff There exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies r''

2. $r = r_1 \cup r_2$

$w \models r_1 \cup r_2$

$\iff w \models r_1$ or $w \models r_2$

\iff [By the inductive hypothesis]

There exists a run w_1 of $\mathcal{D}_{N'}$ satisfying w which also satisfies r''_1 or there exists a run w_1 of $\mathcal{D}_{N'}$ satisfying w which also satisfies r''_2

\iff There exists a run w_1 of $\mathcal{D}_{N'}$ satisfying w which also satisfies r''_1 or r''_2

\iff There exists a run w_1 of $\mathcal{D}_{N'}$ satisfying w which also satisfies $r''_1 \cup r''_2$

\iff There exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies r''

3. $r = r_1^*$

$w \models r_1^*$

\iff Either $w = \epsilon$ or there exist w_1, w_2, \dots, w_k s.t. $w = w_1 w_2 \dots w_k$ and for all $1 \leq i \leq k$, $w_i \models r_1$.

\iff [By the inductive hypothesis]

Either $w = \epsilon$ or there exist w_1, w_2, \dots, w_k s.t. $w = w_1 w_2 \dots w_k$ and for all $1 \leq i \leq k$ there exists a run w'_i of $\mathcal{D}_{N'}$ satisfying w_i which also satisfies r''_1

\iff [By repetitive applications of Lemma 16]

There exists a run w' of $\mathcal{D}_{N'}$ such that either $|w'| = 0$ or $w' = w'_1 w'_2 \dots w'_k$ which satisfies ϵ or $w_1 w_2 \dots w_k$, respectively, and which also satisfies λ or $r''_1 \cdot r''_1 \cdot \dots \cdot r''_1$

\iff There exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies r''_1^*

\iff There exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies r''

4. $r = r_1 \circ n_2$

$w \models r_1 \circ n_2$

\iff There exist w_1, w_2 and b s.t. $w = w_1 b w_2$, $w_1 b \models r_1$ and $b w_2 \models n_2$

- \iff [By the inductive hypothesis]
 There exist w_1, w_2 and b s.t. $w = w_1bw_2$, there exists a run σ_1s_1 of $\mathcal{D}_{N'}$ satisfying w_1b which also satisfies r_1'' , and $bw_2 \models n_2$
- \iff [By the transition relation of N' and the correctness of $\mathcal{D}'_{N'}$ as its representation]
 There exist w_1, w_2 and b s.t. $w = w_1bw_2$, there exists a run σ_1s_1 of $\mathcal{D}_{N'}$ satisfying w_1b which also satisfies r_1'' , and there exists a run $s_2\sigma_2$ of $\mathcal{D}_{N'}$ satisfying bw_2 which also satisfies idle^* and n_2
- \iff [By Lemma 17]
 There exist w_1, w_2 and b s.t. $w = w_1bw_2$ and there exists a run $w' = \sigma_1s_1\sigma_2$ of $\mathcal{D}_{N'}$ satisfying w_1bw_2 which also satisfies $r_1'' \circ n_2$
- \iff There exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies r''
5. $r = n1 \circ r_2$
 Symmetric to the above case.
6. $r = r1 \cap n_2$
 $w \models r1 \cap n_2$
 $\iff w \models r1$ and $w \models n_2$
 \iff [By the inductive hypothesis]
 There exists a run w_1 of $\mathcal{D}_{N'}$ satisfying w which also satisfies r_1'' and $w \models n_2$
 \iff There exists a run w_1 of $\mathcal{D}_{N'}$ satisfying w which also satisfies r_1'' and n_2
 \iff There exists a run w_1 of $\mathcal{D}_{N'}$ satisfying w which also satisfies $r_1'' \cap n_2$
 \iff There exists a run w' of $\mathcal{D}_{N'}$ satisfying w which also satisfies r''
7. $r = r1 \cap n_2$
 Symmetric to the above case.

□

Theorem 10

$$\mathcal{D}_M \models \text{not } r! \iff \mathcal{D}_M \parallel \mathcal{D}_{N'} \models \text{not } r[t \leftarrow t']!$$

Proof.

$$\mathcal{D}_M \not\models \text{not } r!$$

- \iff There exists a finite run w of \mathcal{D}_M s.t. $w \models r$
- \iff [By Lemma 18]
 There exists a finite run w of \mathcal{D}_M s.t. there exists a run of $\mathcal{D}_{N'}$ satisfying w which also satisfies r''
- \iff [By definition of r'']
 There exists a finite run w of \mathcal{D}_M s.t. there exists a run of $\mathcal{D}_{N'}$ satisfying w which also satisfies $r[t \leftarrow t']$
- $\iff \mathcal{D}_M \parallel \mathcal{D}_{N'} \not\models \text{not } r[t \leftarrow t']!$

□

Lemma 19. *Let r be a SERE, and $t = r_1 \cap r_2$ a sub-SERE of r . Let N'_1 and N'_2 be the cruising NFAs for r_1 and r_2 respectively. Let $\mathcal{D}_{N'_1}$ and $\mathcal{D}_{N'_2}$ be their corresponding DTSs.*

Let idle_1 and idle_2 denote, respectively, that $D_{N'_1}$ and $D_{N'_2}$ are in a state s such that $s[\text{state}] = q_I$. Let idle denote the conjunction $\text{idle}_1 \wedge \text{idle}_2$. Let t' denote the SERE placeholder $(r_1 \cap r_2, \text{start}, \text{middle}, \text{end})$. Let r''' denote the SERE $(\text{idle} \vee \text{start}) \circ r[t \leftarrow t'] \circ (\text{end} \vee \text{idle})$ and let r'' denote the SERE r''' if $\mathcal{S}(r) = \text{false}$ and the SERE $\lambda \cup r'''$ otherwise. Then

$$w \models r$$

$$\Downarrow$$

there exists a run of $\mathcal{D}_{N'_1} \parallel \mathcal{D}_{N'_2}$ satisfying w which satisfies also r'' .

Proof. We show only the base case ($r = r_1 \cap r_2$). The rest of the proof proceeds similarly to the proof of Lemma 18.

$$w \models r_1 \cap r_2$$

$$\iff w \models r_1 \text{ and } w \models r_2$$

$$\iff \text{[By Lemma 18]}$$

If $\mathcal{S}(r_1) = \text{false}$ then there exists a run w' of $\mathcal{D}_{N'_1}$ satisfying w which also satisfies

$$\{\text{start}_1 \wedge \text{end}_1\} \cup \{\text{start}_1 \cdot \neg \text{start}_1^* \cdot \text{end}_1\}$$

otherwise there exists a run w' of $\mathcal{D}_{N'_1}$ satisfying w which also satisfies

$$\lambda \cup \{\text{start}_1 \wedge \text{end}_1\}_1 \cup \{\text{start}_1 \cdot \neg \text{start}_1^* \cdot \text{end}_1\}$$

And if $\mathcal{S}(r_2) = \text{false}$ then there exists a run w' of $\mathcal{D}_{N'_2}$ satisfying w which also satisfies

$$\{\text{start}_2 \wedge \text{end}_2\} \cup \{\text{start}_2 \cdot \neg \text{start}_2^* \cdot \text{end}_2\}$$

otherwise there exists a run w' of $\mathcal{D}_{N'_2}$ satisfying w which also satisfies

$$\lambda \cup \{\text{start}_2 \wedge \text{end}_2\} \cup \{\text{start}_2 \cdot \neg \text{start}_2^* \cdot \text{end}_2\}$$

\iff If $\mathcal{S}(r_1 \cap r_2) = \text{false}$ then there exists a run w' of $\mathcal{D}_{N'_1} \parallel \mathcal{D}_{N'_2}$ satisfying w which also satisfies

$$\{(\text{start}_1 \wedge \text{start}_2) \wedge (\text{end}_1 \wedge \text{end}_2)\} \cup \{\text{start}_1 \wedge \text{start}_2 \cdot (\neg \text{start}_1 \wedge \neg \text{start}_2)^* \cdot (\text{end}_1 \wedge \text{end}_2)\}$$

otherwise there exists a run w' of $\mathcal{D}_{N'_1} \parallel \mathcal{D}_{N'_2}$ satisfying w which also satisfies

$$\lambda \cup \{(\text{start}_1 \wedge \text{start}_2) \wedge (\text{end}_1 \wedge \text{end}_2)\} \cup \{\text{start}_1 \wedge \text{start}_2 \cdot (\neg \text{start}_1 \wedge \neg \text{start}_2)^* \cdot (\text{end}_1 \wedge \text{end}_2)\}$$

\iff If $\mathcal{S}(r_1 \cap r_2) = \text{false}$ then there exists a run w' of $\mathcal{D}_{N'_1} \parallel \mathcal{D}_{N'_2}$ satisfying w which also satisfies

$$\{\text{start} \wedge \text{end}\} \cup \{\text{start} \cdot \text{middle}^* \cdot \text{end}\}$$

otherwise there exists a run w' of $\mathcal{D}_{N'_1} \parallel \mathcal{D}_{N'_2}$ satisfying w which also satisfies

$$\lambda \cup \{\text{start} \wedge \text{end}\} \cup \{\text{start} \cdot \text{middle}^* \cdot \text{end}\}$$

\iff There exists a run of $\mathcal{D}_{N'_1} \parallel \mathcal{D}_{N'_2}$ satisfying w which satisfies also r'' .

□

Proposition 12

Let \mathcal{D}_M be a DTS, r a SERE and $r_1 \cap r_2$ a sub-SERE of r . Let N'_1 and N'_2 be the cruising NFAs, for r_1 and r_2 respectively. Let $\mathcal{D}_{N'_1}$ and $\mathcal{D}_{N'_2}$ be their corresponding DTSSs. Then

$$\begin{aligned} \mathcal{D}_M \models \text{not } r! \\ \Downarrow \\ \mathcal{D}_M \parallel \mathcal{D}_{N'_1} \parallel \mathcal{D}_{N'_2} \models \text{not } r[r_1 \cap r_2 \leftarrow \text{placeholder}(r_1 \cap r_2, \text{start}, \text{middle}, \text{end})]! \end{aligned}$$

Proof. The proof is similar to the proof of Theorem 10, this time making use of Lemma 19.

□

Proposition 13

The NFA $N_{p[=i..j]}$ accepts $\text{Lng}(\{p[=i..j]\})$.

Proof. Let $w \in \text{Lng}(\{p[=i..j]\})$, we show there exists a run of $N_{p[=i..j]}$ on w which is accepting. Let k be the number of p 's in w . Since $w \in p[=i..j]$, we have that $i \leq k \leq j$. Denote $w = \neg p^{m_0}, p, \neg p^{m_1}, \dots, p, \neg p^{m_k}$ where $\neg p^b$ denotes b consecutive repetitions of $\neg p$. By the transition relation $q_0^{m_0+1}, q_1^{m_1+1}, q_2^{m_2+1}, \dots, q_k^{m_k}$ is an accepting run of $N_{p[=i..j]}$ on w .

Let $w \notin p[=i..j]$ and assume towards contradiction that $s_0 s_1 \dots s_n$ is an accepting run of $N_{p[=i..j]}$ on w . Therefore $s_n \in \{q_i, \dots, q_j\}$. It is easy to see that for all $0 \leq t \leq n$ and $\forall 1 \leq m \leq j$, if $s_t = q_m$ then there are exactly m states in $s_0 s_1 \dots s_t$ in which p is asserted. Therefore there are between i to j states in $s_0 s_1 \dots s_n$ in which p is asserted. That is, $w \in p[=i..j]$, in contradiction to the assumption. □

Proofs of Section 4.3

Definition 20 (Semantics of CTL formulas). *The semantics of CTL formulas are defined with respect to a model (DTS) and a state in the model. Let $\mathcal{D}_M = \langle V_M, \Theta_M, \rho_M, \mathcal{A}_M \rangle$ be a DTS. The notation $\mathcal{D}_M, s \models \varphi$ means that formula φ holds in state s of model \mathcal{D}_M . The notation $\mathcal{D}_M \models \varphi$ is equivalent to $\mathcal{D}_M, s \models \varphi$ for all s such that $s \models \Theta$. In other words, φ is valid for every initial state of \mathcal{D}_M . The semantics of a CTL formula over \mathcal{D}_M are defined as follows, where b denotes a boolean expression and φ, φ_1 , and φ_2 denote CTL formulas.*

- $\mathcal{D}_M, s \models b \iff s \models b$
- $\mathcal{D}_M, s \models \neg \varphi \iff \mathcal{D}_M, s \not\models \varphi$
- $\mathcal{D}_M, s \models \varphi_1 \wedge \varphi_2 \iff \mathcal{D}_M, s \models \varphi_1 \text{ and } \mathcal{D}_M, s \models \varphi_2$
- $\mathcal{D}_M, s \models \text{EX } \varphi \iff \exists \text{ run } \sigma \text{ of } \mathcal{D}_M \text{ s.t. } |\sigma| > 1, \sigma^0 = s, \text{ and } \mathcal{D}_M, \sigma^1 \models \varphi$
- $\mathcal{D}_M, s \models \text{E}[\varphi_1 \text{ U } \varphi_2] \iff \exists \text{ run } \sigma \text{ of } \mathcal{D}_M \text{ s.t. } \sigma_0 = s \text{ and } \exists k < |\sigma| \text{ s.t. } \mathcal{D}_M, \sigma^k \models \varphi_2 \text{ and } \forall j \text{ s.t. } j < k: \mathcal{D}_M, \sigma^j \models \varphi_1$

- $\mathcal{D}_M, s \models EG\varphi \iff \exists \text{run } \sigma \text{ of } \mathcal{D}_M \text{ s.t. } \sigma^0 = s \text{ and } \forall j \text{ s.t. } 0 \leq j < |\sigma|: \mathcal{D}_M, \sigma_j \models \varphi$

Definition 21 Let \mathcal{D}_M be a DTS and $s \in \Sigma_V$. Let r be a SERE such that $\epsilon \not\models r$.

$$\mathcal{D}_M, s \models \text{not } \{r\}! \iff \text{for every finite run } \sigma \text{ of } \mathcal{D}_M \text{ s.t. } \sigma^0 = s, \sigma \not\models r.$$

Proposition 15

Let \mathcal{D}_M be a discrete system and r a SERE with no starred sub-SERES, such that $\epsilon \not\models r$. Then

$$\mathcal{D}_M \models \text{not } r! \iff \mathcal{D}_M \models \neg T(r).$$

Proof. First we note that the translation procedure (Definition 14) covers all SERES r such that $\epsilon \not\models r$ and r does not contain any starred sub-SERES. Second, we note that Definition 21 is legitimate, since although SERES are defined over non-empty words, the definition $\mathcal{D}_M \models \text{not } r!$ relies on the fact that runs are by definition non-empty and further assumes $\epsilon \not\models r$. From the same reason we can first show that for every $s \in \Sigma_V$,

$$\mathcal{D}_M, s \models \text{not}\{r\}! \iff \mathcal{D}_M, s \models \neg T(r)$$

We show this by induction on the structure of r . Let b be a boolean expression and let r_1, r_2 be SERE's.

- Base case.

$$\begin{aligned} 1. \mathcal{D}_M, s \models \text{not } \{b\}! \\ \iff \mathcal{D}_M, s \models \neg b \\ \iff \mathcal{D}_M, s \models \neg T(b) \end{aligned}$$

- Induction step.

$$\begin{aligned} 2. \mathcal{D}_M, s \models \text{not } \{r \cdot b^*\}! \\ \iff \mathcal{D}_M, s \models \text{not } \{r\}! \\ \iff [\text{by the induction hypothesis}] \mathcal{D}_M, s \models \neg T(r) \\ 3. \mathcal{D}_M, s \models \text{not } \{r_1 \cup r_2\}! \\ \iff \mathcal{D}_M, s \models \text{not } \{r_1\}! \wedge \text{not } \{r_2\}! \\ \iff [\text{by the induction hypothesis}] \mathcal{D}_M, s \models \neg T(r_1) \wedge \neg T(r_2) \\ \iff \mathcal{D}_M, s \models \neg T(r_1 \cup r_2) \\ 4. \mathcal{D}_M, s \models \text{not } \{b \cdot r_1\}! \\ \iff \text{for every finite run } \sigma = ss'\sigma', s \models \neg b \text{ or } \mathcal{D}_M, s' \models \text{not } \{r_1\}! \\ \iff [\text{by the induction hypothesis}] \\ s \models \neg b \text{ or for every } \sigma = ss'\sigma', \mathcal{D}_M, s' \models \neg T(r_1) \\ \iff \mathcal{D}_M, s \models \neg b \vee \text{AX} \neg T(r_1) \\ \iff \mathcal{D}_M, s \models \neg T(b \cdot r_1) \\ 5. \mathcal{D}_M, s \not\models \text{not } \{b^* \cdot r_1\}! \end{aligned}$$

$$\begin{aligned}
&\iff \text{there exists a run } \sigma \text{ s.t. either } \sigma \models r_1 \text{ or } \sigma = s^0 s^1 \dots s^n \sigma' \text{ (where } s^0 = s) \\
&\quad \text{and } \forall 0 \leq j < n : \sigma^j \models b \text{ and } \sigma^n \sigma' \models r_1 \\
&\iff \mathcal{D}_M, s \not\models \text{not } \{r\}! \text{ or there exists a run } \sigma = s^0 s^1 \dots s^n \sigma' \text{ (where } s^0 = s) \\
&\quad \text{and } \forall 0 \leq j < n : \sigma^j \models b \text{ and } \mathcal{D}_M, \sigma^n \not\models \text{not } \{r_1\}! \\
&\iff \text{[by the induction hypothesis]} \\
&\quad \mathcal{D}_M, s \models T(r) \text{ or there exists a run } \sigma = s^0 s^1 \dots s^n \sigma' \text{ (where } s^0 = s) \text{ and} \\
&\quad \forall 0 \leq j < n : \sigma^j \models b \text{ and } \mathcal{D}_M, \sigma^n \models T(r_1) \\
&\iff \mathcal{D}_M, s \models T(r_1) \vee \mathbf{E}[b\mathbf{UT}(r_1)] \\
&\iff \mathcal{D}_M, s \models T(b^*.r_1)
\end{aligned}$$

$$\begin{aligned}
6. \quad &\mathcal{D}_M, s \models \text{not } \{r_1 \cup r_2 \cdot r\}! \\
&\iff \mathcal{D}_M, s \models \text{not } \{r_1 \cdot r\}! \wedge \text{not } \{r_2 \cdot r\}! \\
&\iff \text{[by the induction hypothesis]} \\
&\quad \mathcal{D}_M, s \models \neg T(r_1 \cdot r) \wedge \neg T(r_2 \cdot r) \\
&\iff \mathcal{D}_M, s \models \neg T(r_1 \cup r_2 \cdot r)
\end{aligned}$$

In particular for every initial state s , $\mathcal{D}_M, s \models \text{not } \{r\}! \iff \mathcal{D}_M, s \models \neg T(r)$.
Therefore $\mathcal{D}_M \models \text{not } \{r\}! \iff \mathcal{D}_M \models \neg T(r)$. \square