H-0229 (H0407-001) July 3, 2004
Computer Science

IBM Research Report

Automata Construction for Regular Expressions in Model Checking

Shoham Ben-David, Dana Fisman*, Sitvanit Ruah
IBM Research Division
Haifa Research Laboratory
Mt. Carmel 31905
Haifa, Israel

*and Weizmann Institute of Science

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Automata Construction for Regular Expressions in Model Checking

Shoham Ben-David[1] Dana Fisman[1,2] Sitvanit Ruah[1]

[1] IBM Haifa Research Lab [2] Weizmann Institute of Science

June 30, 2004

Abstract. Industrial temporal languages like PSL/Sugar and ForSpec have augmented the language with Regular Expressions (REs). An RE specification represents a sequence of Boolean events a model may or may not exhibit. A common way of using REs for specification is in a negative way: a not RE! property describes an undesirable behavior of the model. 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 modeled by a non-deterministic finite automaton (NFA) N_r, which accepts sequences satisfying r, and which is linear in the size of r.
In this paper we discuss the translation of a not RE! into an NFA. While many translation methods exist in the literature ([12, 11]), to the best of our knowledge, the adoption of such a method to model-checking has never been explicitly discussed before. We present our method, which adopts that of Glushkov [11] to better suit model checking needs, and discuss its advantages.

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 ([15]) 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 [1], as well as other industry oriented languages (e.g. [2]), augment the logic with the use of Regular Expressions (REs using the formulation of [1]).

An RE specification can be viewed as a sequence of Boolean events describing a desirable behavior of the model. For example, the RE 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, the formula describes an $undesireable$ behavior of the model. Thus the formula 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 of the form not RE!. 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 us to model a not r! formula 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 AG $\neg$(accepting state of N_r). The reduction to an invariant property is very important, since invariant properties are easier to verify by different model checking engines [6]. In fact, several engines can only verify invariant formulas [3, 16]. As shown in [5] many CTL and SERE-based properties can be translated into not r! properties. Since those CTL properties are in the common fragment of ACTL and LTL [14], we get that many LTL formulas can also be translated to not RE! properties. Because of these two advantages, not RE! s properties have become a major component of the IBM model checking tool-set RuleBase [4].

Many algorithms exist for the translation of an RE into an NFA$\dot{\text{H}}$owever, the adoption of it to model checking needs several adjustments which were never discussed before. Copty et al. in [9] mention they compile a ForSpec formula into an invariant, but do not explain which formulas or how it is done. Beer et al. in [5] sketch the idea of translating an RCTL formula into an NFA, but do not elaborate any further.

In this paper we present the translation of a not r! formula into an NFA which is linear in the size of r. Our construction follows that of Glushkov [11], to build a $position$ NFA (See section 3). This construction is considered the $natural$ NFA of r in the sense that every letter in r corresponds to a state in N_r. Several differences exist between our construction and Glushkov's, which adjust the NFA to better suit model checking needs.

The rest of the paper is organized as follows. Section 2 covers some preliminaries. In section 3 we give our NFA translation, and discuss its unique characteristics. Section 4 concludes the paper.

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) [13]. A DTS $\mathcal{D} : \langle V, \Theta, \rho, \mathcal{A}, \mathcal{J} \rangle$ consists of the following components:

- $V = \{u_1, \ldots, u_n\}$: 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).
- 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 $\langle s, s' \rangle \Vdash \rho(V, V')$, 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.

– $\mathcal{J} = \{J_1, \ldots, J_k\}$: 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 $\sigma : s_0 s_1 s_2 \ldots$ satisfying the requirements of *initiality* i.e. that $s_0 \Vdash \Theta$; and of *consecution* i.e. that for each $j = 0, 1, \ldots,$ 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 $\sigma : s_0 s_1 s_2 \ldots s_n \ldots$ is said to be *satisfying a finite word* $w = b_0 b_1 \ldots b_n$ over B_U iff for every i, $0 \leq i \leq n$, $s_i \Vdash b_i$. A run $\sigma : s_0 s_1 s_2 \ldots s_{n+1} \ldots$ satisfying a finite word $w = b_0 b_1 \ldots b_n$ is said to be *accepting* w iff s_{n+1} satisfies $\mathcal{A}$. An infinite run $\sigma : s_0 s_1 s_2 \ldots$ is said to be *satisfying an infinite word* $w = b_0 b_1 \ldots$ over B_U iff for every $i \geq 0$, $s_i \Vdash b_i$. 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 $\sigma = s_0 s_1 s_2 \ldots$ we denote by $\sigma|_U$ the run $s_0|_U \; s_1|_U \; s_2|_U \; \ldots$.

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 $\mathcal{D}_1 \;|||\; \mathcal{D}_2$ and define it to be $\mathcal{D}_1 \;|||\; \mathcal{D}_2 = \langle V_1 \cup V_2,\ \Theta_1 \wedge \Theta_2,\ \rho_1 \wedge \rho_2,\ \mathcal{A}_1 \wedge \mathcal{A}_2, \mathcal{J}_1 \cup \mathcal{J}_2 \rangle$. 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) [12] 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) [17].

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 $Q \cup \{q_{sink}\}$. 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

$$V_N = V \cup \{state\}; \quad \Theta_N = \bigvee_{q_0 \in Q_0} state = q_0; \quad \mathcal{A}_N = \bigvee_{q \in A} state = q; \quad \mathcal{J}_N = \emptyset;$$

$$\rho_N = \bigvee\nolimits_{(q_1, \sigma, q_2) \in \delta} \begin{array}{l} (state = q_1 \wedge \sigma \wedge state' = q_2) \bigvee \\ (state = q_1 \wedge \neg\sigma \wedge state' = q_{sink}) \end{array}$$

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 $\mathcal{F} = \{F_1, \ldots, F_k\}$. 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:

$$V_G = V \cup \{state\}; \quad \Theta_G = \bigvee_{q_0 \in Q_0} state = q_0; \quad \mathcal{A}_G = \emptyset;$$
$$\mathcal{J}_G = \{J_1, \ldots, J_k\} \text{ where for each } 1 \leq i \leq k: \; J_i = \bigvee_{q \in F_i} state = q;$$
$$\rho_G = \bigvee_{(q_1,\sigma,q_2)\in\delta} \begin{array}{l}(state = q_1 \wedge \sigma \wedge state' = q_2) \bigvee \\ (state = q_1 \wedge \neg\sigma \wedge state' = q_{sink}) \bigvee \\ (state = q_{sink} \wedge state' = q_{sink})\end{array}$$

In this paper, given an NFA $N = \langle B, Q, Q_0, \delta, A\rangle$ we first construct a terminal Büchi automaton [8] 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 $\sigma = \sigma_0 s_1 \ldots$ 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 $(q_{j_1}, \sigma, q_{j_2}) \in \delta$ of N iff $(s_i, s_{i+1}) \models (state = q_1 \wedge \sigma \wedge state' = q_2)$.

2.2 The logic

The logic considered in this paper is the fragment of the industry-standard temporal logic PSL/Sugar [1] that consists of only not r! formulas where r is a regular expression (RE). Its formal definition is given 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 $s \in \Sigma$, $s \Vdash$ *true* and $s \nVdash$ *false*.

Definition 1 (REs).

- *The empty set* $\emptyset$ *and the empty regular expression* λ *are* RE*s.*
- *Every boolean expression* $b \in B$ *is an* RE*.*
- *If* r, r_1, *and* r_2 *are* RE*s, then the following are also* RE*s:*
  1. $\{r\}$ *(encapsulation)* 2. $r_1 \cup r_2$ *(union)*
  3. $r_1 \cdot r_2$ *(concatenation)* 4. r^* *(Kleene closure)*

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 $w\epsilon = \epsilon w = w$. Let L_1 and L_2 be sets of words. The *concatenation* of L_1 and L_2, denoted $L_1 L_2$ is the set $\{w \mid \exists w_1 \in L_1,\ \exists w_2 \in L_2 \text{ and } w = w_1 w_2\}$. Define $L^0 = \{\epsilon\}$ and $L^i = LL^{i-1}$ for $i \geq 1$. The *Kleene closure* of L denoted L^* is the set $\bigcup_{i<\omega} L^i$.[1]

We denote the length of a word w by $|w|$. An empty word $w = \epsilon$ has length 0, and a finite word $w = (s_0 s_1 s_2 \cdots s_n)$ 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

[1] Where ω denotes the cardinality of the non-negative integers.

$s|_U$ the restriction of the letter s to the state-variables in U. For a word $w = s_0 s_1 s_2 \ldots$ we denote by $w|_U$ the restriction of every letter in w to the state-variables in U (i.e, $w|_U = s_0|_U\ s_1|_U\ s_2|_U\ \ldots$).

Definition 2. *The semantics of* RE *s are defined using the relation* $\models\!\!\!\equiv$ *between* RE*s over* B *and (possibly empty) finite words over* Σ*. When* $w \models\!\!\!\equiv r$ *we say that* w tightly satisfies r*. The semantics of* RE *s are defined as follows, where* w *is a finite (possibly empty) word over* Σ*,* b *denotes a boolean expression in* B*, and* r*,* r_1*, and* r_2 *denote* RE*s over* B*.*

- $w \not\models\!\!\!\equiv \emptyset$
- $w \models\!\!\!\equiv \lambda \Longleftrightarrow w = \epsilon$
- $w \models\!\!\!\equiv b \Longleftrightarrow |w| = 1$ *and* $w^0 \Vdash b$
- $w \models\!\!\!\equiv r_1 \cup r_2 \Longleftrightarrow w \models\!\!\!\equiv r_1$ *or* $w \models\!\!\!\equiv r_2$
- $w \models\!\!\!\equiv r_1 \cdot r_2 \Longleftrightarrow \exists w_1, w_2$ *s.t.* $w = w_1 w_2$, $w_1 \models\!\!\!\equiv r_1$, *and* $w_2 \models\!\!\!\equiv r_2$
- $w \models\!\!\!\equiv r^* \Longleftrightarrow w = \epsilon$ *or* $\exists w_1, w_2$ *s.t.* $w_2 \neq \epsilon$, $w = w_1 w_2$, $w_1 \models\!\!\!\equiv r^*$ *and* $w_2 \models\!\!\!\equiv r$

We note that despite the surface similarities to traditional regular expressions (defined below), there are some subtleties. In particular, the set of words satisfying a traditional regular expression is defined over the same alphabet as the regular expression itself (while here the alphabet of the regular expression is B while the alphabet of the words satisfying it is Σ). Moreover, the traditional semantics of REs, assumes *letters* (the finest elements, other than λ and $\emptyset$, appearing in an RE) of the alphabet are mutually exclusive. This assumption does not hold here since the RE-letters are boolean expressions which may hold simultaneously.

Definition 3 (The Language of RE**s)** *Let* Γ *be a finite set of symbols (an alphabet). Let* b *be a letter in* Γ *and* r*,* r_1*, and* r_2 SERE*s over* Γ*. The set* $Lng(r)$*, defined below, denotes the set of words over* Γ *satisfying* r *according to the traditional semantics of regular expressions.*

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

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

We use the syntax not $r!$ to be compliant with PSL [1]. The semantics given here to not $r!$ is equivalent to the one given in PSL to negating a strong SERE, only that we give it directly for the composed construct over the given model.

3 Automata Construction for Regular Expressions

Below we describe the construction of an NFA from an RE. Our construction adjusts that of Glushkov [11] (which was popularized by Berry and Sethi [7]) to better suit the task of verification. The construction works on linear REs, where an RE is said to be *linear* iff no letter appears in it more than once. This construction is considered the *natural* NFA of r [7], in the sense that every letter in r corresponds to a state in N_r. In the sequel, we elaborate on the differences between the original Glushkov construction and the construction given here.

In order to linearize the given RE, we add a subscript to each letter appearing in the RE. The subscripting is done such that every letter in r gets a natural number subscript, and the subscripts create an increasing tight sequence of natural numbers. For example, the result of subscripting the RE $\{\{a{\cdot}b\}\} \cup \{b{\cdot}c^*\}{\cdot}a\}$ is $\{\{a_1{\cdot}b_2\} \cup \{b_3{\cdot}{c_4}^*\}{\cdot}a_5\}$. We denote by $\widetilde{r}$ the the result of subscripting the RE r. With this approach the subscripted symbols a_i and b_j are called *positions* and the set of positions in $\widetilde{r}$ is denoted $pos(r)$. We use x, y, z as variables for positions.

We note that when we work with subscripted REs we consider their traditional semantics, i.e. the set $Lng(\widetilde{r})$ of words over the alphabet consisting of their positions. Later, to connect to the semantics of an RE (as given in Definition 2) we strip away the position and move from the alphabet of boolean expressions to the alphabet of interpretation of state variables by considering the FDS representation of the NFA.

Before we apply the construction we remove all occurrences of λ and $\emptyset$. This can be done by substituting each occurrence of $\emptyset$ with *false*, and each occurrence of λ with *false**, due to the following claim.

Claim 5. *Let w be a word over Σ. Then,*

$$w \models\!\!\!\equiv \emptyset \iff w \models\!\!\!\equiv \mathit{false} \qquad \textit{and} \qquad w \models\!\!\!\equiv \lambda \iff w \models\!\!\!\equiv \mathit{false}^*$$

Proof.

– $w \models\!\!\!\equiv \mathit{false}$
$\iff$ $|w| = 1$ and $w^0 \models \mathit{false}$
$\iff$ FALSE
$\iff$ $w \models\!\!\!\equiv \emptyset$

– $w \models\!\!\!\equiv \mathit{false}^*$
$\iff$ either $w = \epsilon$ or $\exists w_1, w_2$ s.t. $w_2 \neq \epsilon$, $w = w_1 w_2$, $w_1 \models\!\!\!\equiv \mathit{false}^*$ and $w_2 \models\!\!\!\equiv \mathit{false}$
$\iff$ either $w = \epsilon$ or $\exists w_1, w_2$ s.t. $w_2 \neq \epsilon$, $w = w_1 w_2$, $w_1 \models\!\!\!\equiv \mathit{false}^*$ and [by the item above] FALSE
$\iff$ $w = \epsilon$
$\iff$ $w \models\!\!\!\equiv \lambda$

□

After the substitution we get an RE whose finest components are boolean expressions.

Definition 6 (Position Functions). *We use the following function to capture the notion of positions in an* RE*, where $Lng(r)$ denotes the set $\{w \mid w \models\!\!\!\equiv r\}$.*

- $\mathcal{F}(r)$ *- the set of positions that match the* first *letter of some word in $Lng(\widetilde{r})$. Formally, $\mathcal{F}(r) = \{x \in pos(r) \mid \exists v \in pos(r)^*$ s.t. $xv \in Lng(\widetilde{r})\}$.*
- $\mathcal{L}(r)$ *- the set of positions that match the* last *letter of some word in $Lng(\widetilde{r})$. Formally, $\mathcal{L}(r) = \{x \in pos(r) \mid \exists v \in pos(r)^*$ s.t. $vx \in Lng(\widetilde{r})\}$.*

– $\mathcal{N}(r,x)$ - *the set of positions that can* follow *position x in a path through $\widetilde{r}$. Formally,* $\mathcal{N}(r,x) = \{y \in pos(r) \mid \exists u,v \in pos(r)^* \text{ s.t. } uxyv \in Lng(\widetilde{r})\}$.
– $\mathcal{P}(r,x)$ - *the set of positions that can* precede *position x in a path through $\widetilde{r}$. Formally,* $\mathcal{P}(r,x) = \{y \in pos(r) \mid \exists u,v \in pos(r)^* \text{ s.t. } uyxv \in Lng(\widetilde{r})\}$.

Below we give an inductive definition of these functions. The definitions are based on a predicate $\mathcal{S}(r)$ that returns *true* if $\epsilon \models r$, and *false* otherwise. This predicate can be defined inductively as follows: $\mathcal{S}(\emptyset) = \textit{false}$; $\mathcal{S}(\lambda) = \textit{true}$; $\mathcal{S}(b) = \textit{false}$; $\mathcal{S}(r_1 \cdot r_2) = \mathcal{S}(r_1) \wedge \mathcal{S}(r_2)$; $\mathcal{S}(r_1 \cup r_2) = \mathcal{S}(r_1) \vee \mathcal{S}(r_2)$; and $\mathcal{S}(r^*) = \textit{true}$. We use r, r_1, r_2 to denote REs, s_1, s_2 starred REs (REs such that $\mathcal{S}(s_1) = \mathcal{S}(s_2) = \textit{true}$) and n_1, n_2 non-starred REs:[2]

- $\mathcal{F}(\emptyset) = \emptyset$ - $\mathcal{F}(\lambda) = \emptyset$ - $\mathcal{F}(x) = \{x\}$ - $\mathcal{F}(r_1 \cup r_2) = \mathcal{F}(r_1) \cup \mathcal{F}(r_2)$ - $\mathcal{F}(n_1 \cdot r_2) = \mathcal{F}(n_1)$ - $\mathcal{F}(s_1 \cdot r_2) = \mathcal{F}(s_1) \cup \mathcal{F}(r_2)$ - $\mathcal{F}(r^*) = \mathcal{F}(r)$	- $\mathcal{N}(x,x) = \emptyset$ - $\mathcal{N}(r_1 \cup r_2, x) = \begin{cases} \mathcal{N}(r_1,x) & \text{if } x \in pos(r_1) \\ \mathcal{N}(r_2,x) & \text{if } x \in pos(r_2) \end{cases}$ - $\mathcal{N}(r_1 \cdot r_2, x) = \begin{cases} \mathcal{N}(r_1,x) & \text{if } x \in pos(r_1) \setminus \mathcal{L}(r_1) \\ \mathcal{N}(r_1,x) \cup \mathcal{F}(r_2) & \text{if } x \in \mathcal{L}(r_1) \\ \mathcal{N}(r_2,x) & \text{if } x \in pos(r_2) \end{cases}$ - $\mathcal{N}(r^*, x) = \begin{cases} \mathcal{N}(r,x) & \text{if } x \in pos(r) \setminus \mathcal{L}(r) \\ \mathcal{N}(r,x) \cup \mathcal{F}(r) & \text{if } x \in \mathcal{L}(r) \end{cases}$
- $\mathcal{L}(\emptyset) = \emptyset$ - $\mathcal{L}(\lambda) = \emptyset$ - $\mathcal{L}(x) = \{x\}$ - $\mathcal{L}(r_1 \cup r_2) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2)$ - $\mathcal{L}(r_1 \cdot n_2) = \mathcal{L}(n_2)$ - $\mathcal{L}(r_1 \cdot s_2) = \mathcal{L}(s_2) \cup \mathcal{L}(r_1)$ - $\mathcal{L}(r^*) = \mathcal{L}(r)$	- $\mathcal{P}(x,x) = \emptyset$ - $\mathcal{P}(r_1 \cup r_2, x) = \begin{cases} \mathcal{P}(r_1,x) & \text{if } x \in pos(r_1) \\ \mathcal{P}(r_2,x) & \text{if } x \in pos(r_2) \end{cases}$ - $\mathcal{P}(r_1 \cdot r_2, x) = \begin{cases} \mathcal{P}(r_2,x) & \text{if } x \in pos(r_2) \setminus \mathcal{F}(r_2) \\ \mathcal{P}(r_2,x) \cup \mathcal{L}(r_1) & \text{if } x \in \mathcal{F}(r_2) \\ \mathcal{P}(r_1,x) & \text{if } x \in pos(r_1) \end{cases}$ - $\mathcal{P}(r^*, x) = \begin{cases} \mathcal{P}(r,x) & \text{if } x \in pos(r) \setminus \mathcal{F}(r) \\ \mathcal{P}(r,x) \cup \mathcal{L}(r) & \text{if } x \in \mathcal{F}(r) \end{cases}$

Based on these functions we can build an NFA N that recognizes the set of words tightly satisfying r. Denote $S = \mathcal{F}(r)$, $E = \mathcal{L}(r)$, $N = \{xy \mid x \notin \mathcal{P}(r,y)\}$, and $B_r = \{b \mid b \in pos(r)\} \cup \{\neg b \mid b \in pos(r)\}$. Define D to be the NFA $\langle B_r, Q, Q_0, \delta, A\rangle$ where $Q = \{q_\sigma \mid \sigma \in pos(r)\} \cup \{q_\infty, q_{sink}\}$; $Q_0 = \{q_\sigma \mid \sigma \in S\}$ if $\mathcal{S}(r) = \textit{false}$ and $Q_0 = \{q_\sigma \mid \sigma \in S\} \cup \{q_\infty\}$ otherwise; $A = \{q_\infty\}$; and

$$\delta = \begin{array}{l} \{(q_{\sigma_1}, \sigma_1, q_{\sigma_2}) \mid \sigma_1\sigma_2 \notin N\} \cup \{(q_\sigma, \neg\sigma, q_{sink}) \mid \sigma \in pos(r)\} \cup \\ \{(q_\sigma, \sigma, q_\infty) \mid \sigma \in E\} \cup \{(q_{sink}, \sigma, q_{sink}) \mid \sigma \in B_r\} \end{array}$$

The Satellite's Characteristics We call the NFA which results from our construction a *satellite*, since it runs in parallel to the model, looks at its state-variables, but does not interfere with the run. We note that our satellite has the special nature, that *outgoing* edges from a given state are labeled by a single Boolean expression $-$ the corresponding position in the RE, or its negation (when the transition is to the sink state). This is different than a regular Glushkov automata where all *incoming* edges to a given state are labeled by the corresponding position.

[2] Since λ and $\emptyset$ have no positions, $\mathcal{N}()$ and $\mathcal{P}()$ are not defined for $r = \emptyset$ or $r = \lambda$.

One may claim that this difference results in more non-determinism. For instance, that the satellite for the RE $\{a \cdot b^* \cdot c\}$ will be non-deterministic (since the state q_a corresponding to position a has two outgoing edges with the same label a, one that enters the state q_b corresponding to b and one that enters the state q_c corresponding to c) while the Glushkov automata will be deterministic (since by definition every edge entering a state q_x corresponding to position x is labeled x, thus, when the SERE is linear, it cannot be that there are two outgoing edges from the same states with the same label reaching two distinct states). However, since (as noted in subsection 2.2) the REs alphabet is not mutual exclusive, the original Glushkov automata will not be deterministic either, since the fact that two outgoing edges has different labels, does not mean they cannot both be taken.

Another difference between our construction and Glushkov's is that our satellite has a sink state, while Glushkov's automaton does not. This characteristic is useful for model checking *weak* regular expressions [10]. The sink state can be used to distinguish between runs that have *failed* (no extension of them will satisfy the given RE), and thus do not satisfy the weak RE and runs on words not satisfying r (whose extension may eventually satisfy the given RE) and so may satisfy the weak RE.

For an RE r we denote by $\mathcal{D}_r$ the discrete transition system representing the satellite of r. The following proposition states that $\mathcal{D}_r$ recognizes words that tightly satisfy r.

Proposition 7. *Let r be an RE over B and w a word over some $\Sigma' \supseteq \Sigma$. Then*

$$w \models r \text{ iff there exists a finite accepting run of } \mathcal{D}_r \text{ satisfying } w.$$

The proof of Proposition 7 makes use of the following three lemmas.

Lemma 8. *Let $S, E \subseteq \Sigma$, $N \subseteq \Sigma^2$ and $L = (S\Sigma^* \cap \Sigma^* E) \setminus \Sigma^* N \Sigma^*$. Let D be the automaton $\langle B, Q, Q_0, \delta, A\rangle$ where:*

- $B = \{\sigma \mid \sigma \in \Sigma\} \cup \{\overline{\sigma} \mid \sigma \in \Sigma\}$
- $Q = \{q_\sigma \mid \sigma \in \Sigma\} \cup \{q_\infty, q_{sink}\}$
- $Q_0 = \{q_\sigma \mid \sigma \in S\}$
- $\delta = \begin{array}{l} \{(q_{\sigma_1}, \sigma_1, q_{\sigma_2}) \mid \sigma_1\sigma_2 \notin N\} \cup \{(q_\sigma, \overline{\sigma}, q_{sink}) \mid \sigma \in \Sigma\} \cup \\ \{(q_\sigma, \sigma, q_\infty) \mid \sigma \in E\} \cup \{(q_{sink}, \sigma, q_{sink}) \mid \sigma \in B\} \end{array}$
- $A = \{q_\infty\}$

Then $w \in L$ iff there exists a run of D on w that terminates in a state $s \in A$

Proof. Note that $\epsilon \notin L$.

Let $r = s_1, \ldots, s_{n+1}$ be a run of D on $w = \sigma_1\sigma_2\ldots\sigma_n$ s.t. $s_{n+1} \in A$. Then since $s_{n+1} \in A$ we get $s_{n+1} = q_\infty$. Therefore, by the transition relation, we get that $s_n = q_{\sigma_n}$. Therefore by the transition relation, we get that $s_{n-1} = q_{\sigma_{n-1}}$ and so on. Thus the run r of D on w looks as follows:

$$q_{\sigma_1} \xrightarrow{\sigma_1} q_{\sigma_2} \xrightarrow{\sigma_2} q_{\sigma_3} \quad \cdots \quad q_{\sigma_n} \xrightarrow{\sigma_n} q_\infty$$

Since q_{σ_1} is an initial state we get that $\sigma_1 \in S$ and from q_∞ being the accepting state we get $\sigma_n \in E$. Also, for every i, $1 \leq i < n$ the transition $q_{\sigma_i} \xrightarrow{\sigma_i} q_{\sigma_{i+1}}$ implies $\sigma_i\sigma_{i+1} \notin N$. Thus $\sigma_1\sigma_2\ldots\sigma_n \in (S\Sigma^* \cap \Sigma^* E) \setminus \Sigma^* N \Sigma^*$. That is $w \in L$.

Conversely, if $w = \sigma_1\sigma_2 \ldots \sigma_n \in L$, it follows, $\sigma_1 \in S$, $\sigma_n \in E$ and for $1 \leq i < n$, $\sigma_i\sigma_{i+1} \notin N$. Therefore $q_{\sigma_1} \xrightarrow{\sigma_1} q_{\sigma_2} \xrightarrow{\sigma_2} q_{\sigma_3} \quad \cdots \quad q_{\sigma_n} \xrightarrow{\sigma_n} q_\infty$ is a run of D terminating in a state in A.

□

Lemma 9. *Let $S, E \subseteq \Sigma$, $N \subseteq \Sigma^2$ and $L = (S\Sigma^* \cap \Sigma^* E) \setminus \Sigma^* N \Sigma^*$, and let $L' = \{\epsilon\} \cup L$. Let D' be the automaton $\langle B, Q, Q'_0, \delta, A\rangle$ where:*

- $B = \{\sigma \mid \sigma \in \Sigma\} \cup \{\overline{\sigma} \mid \sigma \in \Sigma\}$
- $Q = \{q_\sigma \mid \sigma \in \Sigma\} \cup \{q_\infty, q_{sink}\}$
- $Q_0 = \{q_\sigma \mid \sigma \in S\} \cup \{q_\infty\}$
- $\delta = \begin{array}{l} \{(q_{\sigma_1}, \sigma_1, q_{\sigma_2}) \mid \sigma_1\sigma_2 \notin N\} \cup \{(q_\sigma, \overline{\sigma}, q_{sink}) \mid \sigma \in \Sigma\} \cup \\ \{(q_\sigma, \sigma, q_\infty) \mid \sigma \in E\} \cup \{(q_{sink}, \sigma, q_{sink}) \mid \sigma \in B\} \end{array}$
- $A = \{q_\infty\}$

Then $w \in L'$ iff there exists a run of D' on w that terminates in a state $s \in A$

Proof. Note that D' is equivalent to D in all components but the initial states, which include also q_∞. Thus clearly D' recognizes a word w iff w is recognized by D or $w = \epsilon$. Thus, by lemma 8, $w \in L'$ iff there exists a run of D' on w that terminates in a state $s \in A$. □

Lemma 10. *Let w be a word over Σ, r an* RE *over B. Then $w \models r$ iff either $\epsilon \in Lng(r)$ and $w = \epsilon$ or there exists a word $\beta = b_0 \ldots b_n \in Lng(r)$ such that $w_i \models b_i$ for $0 \leq i \leq n$.*

Proof. By induction on the structure of r.

1. $r = \emptyset$
   $w \models \emptyset$ iff **False** iff there exists a word $\beta \in Lng(r)$ such that $w_i \models b_i$ for $0 \leq i \leq n$.
2. $r = \lambda$
   $\Longrightarrow \epsilon \in Lng(r) \Longrightarrow w \models \lambda$ iff $w = \epsilon$.

Assume the claim holds for the RE's r_1, r_2

1. $r = r_1 \cup r_2$
   $w \models r_1 \cup r_2$ iff $w \models r_1$ or $w \models r_2$ iff by the induction hypothesis either $\epsilon \in Lng(r_1)$ and $w = \epsilon$ or $\exists\beta = b_0 \ldots b_n \in Lng(r_1)$ such that $w_i \models b_i$ for $0 \leq i \leq n$ or $\epsilon \in Lng(r_2)$ and $w = \epsilon$ or $\exists\beta = b_0 \ldots b_n \in Lng(r_2)$ such that $w_i \models b_i$ for $0 \leq i \leq n$ iff either $\epsilon \in Lng(r_1) \cup Lng(r_2)$ and $w = \epsilon$ or $\exists\beta \in Lng(r_1) \cup Lng(r_2)$ such that $w_i \models b_i$ for $0 \leq i \leq n$ iff either $\epsilon \in Lng(r_1 \cup r_2)$ and $w = \epsilon$ or $\exists\beta \in Lng(r_1 \cup r_2)$ such that $w_i \models b_i$ for $0 \leq i \leq n$.
2. $r = r_1 \cdot r_2$
   $w \models r_1 \cdot r_2$ iff $\exists u_1, u_2$ such that $w = u_1 u_2$, $u_1 \models r_1$ and $u_2 \models r_2$. By the induction hypothesis iff $\exists u, v, b_1 = b_0^1 \ldots b_{n_1}^1 \in Lng(r_1), b_2 = b_0^2 \ldots b_{n_2}^2 \in Lng(r_2)$ such that $w = uv$ and either $u = \epsilon$ and $\epsilon \in Lng(r_1)$ or $u_i \models b_i^1$ for $0 \leq i \leq n_1$ and either $v = \epsilon$ and $\epsilon \in Lng(r_2)$ or $u_i \models b_i^2$ for $0 \leq i \leq n_2$. iff $\exists u, v, \beta = b_1 b_2$ where $b_1 \in Lng(r_1), b_2 \in Lng(r_2)$ such that $w = uv$ and either $w = \epsilon$ and $\epsilon \in Lng(r_1 \cdot r_2)$ or $w_i \models b_i$ for $0 \leq i \leq n_1 + n_2$ iff either $w = \epsilon$ and $\epsilon \in Lng(r_1 \cdot r_2)$ or $\exists\beta \in Lng(r_1 \cdot r_2)$ such that $w_i \models b_i$ for $0 \leq i \leq n_1 + n_2$.

3. $r = {r_1}^*$.
   By induction on the length of w. For $w = \epsilon$, $w \models r$ since $\epsilon \in Lng(r)$. Assume $|w| > 0$ and the claim holds for u such that $|u| < |w|$ and $r = {r_1}^*$. For $w \neq \epsilon$, $w \models {r_1}^*$ iff $\exists u_1, u_2$ such that $u_2 \neq \epsilon$, $w = u_1 u_2$, $u_1 \models {r_1}^*$ and $u_2 \models r_1 \iff$ (By the induction hypothesis on $|w|$) $\exists u_1, u_2$ such that $u_2 \neq \epsilon$, $w = u_1 u_2$, either $u_1 = \epsilon$ or there exists a word $b_1 \in Lng(r)$ such that $\forall 0 \leq i \leq |u_1| : u_1^i \models b_1^i$ and $u_2 \models r_1$. $\iff$ (By the induction hypothesis on the structure of r) iff $\exists u_1, u_2$ such that $u_2 \neq \epsilon$, $w = u_1 u_2$, either $u_1 = \epsilon$ or there exists a word $b_1 \in Lng({r_1}^*)$ such that $u_1^i \models b_1^i$ for $0 \leq i < |u_1|$ and there exists $b_2 \in Lng(r_1)$ such that $u_2^i \models b_2^i$ for $0 \leq i < |u_2| \iff$ there exists a word $\beta = b_1 b_2$ such that $b_1 \in Lng({r_1}^*)$, $b_2 \in Lng(r_1)$ and $w_i \models b_i$ for $0 \leq i < |\beta| \iff$ there exists a word $\beta \in Lng(r)$ such that $w_i \models b_i$ for $\forall 0 \leq i < |\beta|$.

Proof of Proposition 7

Proof. Let V_r be the set of variables over which $pos(r)$ ranges. Let Σ_r be the set of states providing interpretations to V_r and let B_r be the set of boolean expressions over V_r. Denote $S = \mathcal{F}(r)$, $E = \mathcal{L}(r)$, $N = \{xy \mid x \notin \mathcal{P}(y, r)\}$ and $L = (SB_r^* \cap B_r^* E) \setminus B_r^* N B_r^*$. Let $L' = L$ if $\mathcal{S}(r) = \mathit{false}$ and $L' = L \cup \{\epsilon\}$, otherwise. Let D be the NFA constructed for r as in subsection 2.1. Then D is the NFA constructed in Lemma 8 or Lemma 9 (depending if $\mathcal{S}(r) = \mathit{false}$ or not) modulo the use of $\neg\sigma$ instead of $\overline{\sigma}$. Thus by Lemma 8 or Lemma 9, a word β over B_r belongs to L' iff there exists an accepting run of D on β. Let $w = s_0 s_1 \ldots s_n$ be a word over Σ_r. Then, by Lemma 10, $w \models r$ iff either $\epsilon \in L'$ and $w = \epsilon$ or there exists a word $\beta = b_0 \ldots b_n \in L'$ such that $s_i \models b_i$ for $0 \leq i \leq n$. Thus $w \models r$ iff there exists an accepting run of $\mathcal{D}_r$ satisfying w. And this is true for any Σ and B defined over some $V \supseteq V_r$. $\square$

To verify a $\mathsf{not}\ r!$ formula, we can run $\mathcal{D}_r$ in parallel to the given model, and check that the joint run does not reach a finite accepting state of $\mathcal{D}_r$, i.e. a state satisfying $\mathcal{A}_r$.

Proposition 11. *Let $\mathcal{D}_M$ be a* DTS*, r an* RE*, and $\mathcal{D}_r$ the* DTS *of r constructed as above. Then,*

$$\mathcal{D}_M \models \mathsf{not}\ r! \iff \mathcal{D}_M \,|||\, \mathcal{D}_r \models \mathsf{AG}\ \neg\mathcal{A}_r$$

Proof.

- If direction.
  $\mathcal{D}_M \models \mathsf{not}\ r!$
  $\implies$ $\forall\sigma$ a finite run of $\mathcal{D}_M$, $\sigma \not\models r$ [by Proposition 7]
  $\implies$ $\forall\sigma$ a finite run of $\mathcal{D}_M$, every run σ_r of $\mathcal{D}_r$ satisfying $\sigma|_{V_r}$ does not reach a state satisfying $\mathcal{A}_r$
  $\implies$ $\forall$ finite run σ' of $\mathcal{D}_M \,|||\, \mathcal{D}_r$, does not reach a state satisfying $\mathcal{A}_r$
  $\implies$ $\mathcal{D}_M \,|||\, \mathcal{D}_r \models \mathsf{AG}\ \neg\mathcal{A}_r$.
- Only if direction.
  $\mathcal{D}_M \,|||\, \mathcal{D}_r \models \mathsf{AG}\ \neg\mathcal{A}_r$.
  $\implies$ Any finite run σ of $\mathcal{D}_M \,|||\, \mathcal{D}_r$ does not reach a state satisfying $\mathcal{A}_r$

$\Longrightarrow$ Any finite run σ_r of $\mathcal{D}_r$ satisfying a word $\sigma_M|_{V_r}$ which is a finite run of $\mathcal{D}_M$ does not reach a state satisfying $\mathcal{A}_r$ [by Proposition 7]
$\Longrightarrow$ For any finite run σ_M of $\mathcal{D}_M$, $\sigma_M \not\models r$
$\Longrightarrow$ $\mathcal{D}_M \models \textsf{not}\, r!$

$\square$

This proposition 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 $\neg\textsf{EF}\, A$ (see [8]).

4 Conclusions

Verification of not RE! formula over a given model can be reduced to verification of an invariant formula over an extended model, consisting of a parallel composition of the given model with a non-deterministic finite automata (NFA). In this paper we have shown how to generate an NFA and an invariant formula from a given not RE! formula.

The importance of this reduction stems from the fact that (1) verification of invariant properties is extremely efficient compared to other properties and (2) a large subset of temporal logic properties can be transformed into not SERE! properties ([5, 14]) and thus enjoy this reduction.

The reduction presented here is the main translation path in the IBM model checking tool-set Rulebase [4].

Industrial temporal logics such as PSL/Sugar ([1]) contain extended regular expressions (EREs or SEREs) which augments the traditional regular expressions with additional operators. Translating not SERE! properties can be done by first transforming a SERE to an RE(this is possible since SEREs are expressible as REs) and then using the procedure given here. We are currently working on more efficient algorithms for translating general SERE s to automata.

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. Accellera. Accellera property language reference manual. In *http://www.eda.org/vfv/docs/PSL-v1.1.pdf*, pages 109–117, June 2004. Appendix B.
2. 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.
3. S. Barner and Y.Rodeh. Searching for counter-examples adaptively. In *The Sixth International Workshop in Formal Methods (IWFM'03)*, July 2003.

4. 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.* 9^{th} *International Conference on Computer Aided Verification (CAV)*, LNCS 1254, pages 480–483. Springer-Verlag, 1997.
5. I. Beer, S. Ben-David, and A. Landver. On-the-fly model checking of RCTL formulas. In *Proc.* 10^{th} *International Conference on Computer Aided Verification (CAV'98)*, LNCS 1427, pages 184–194. Springer-Verlag, 1998.
6. S. Ben-David, C. Eisner, D. Geist, and Y. Wolfsthal. Model checking at ibm. *Formal Methods in System Design*, 22(2):101–108, 2003.
7. G. Berry and R. Sethi. From regular expression to deterministic automata. *Theoretical Computer Science*, 48:117–126, 1986.
8. 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.
9. F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Y. Vardi. Benefits of bounded model checking at an industrial setting. In *CAV'01*, july 2001.
10. D. Fisman, C. Eisner, and J. Havlicek. Weak regular expression. to appear.
11. V. M. Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16:1–53, 1953.
12. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Series in Computer Science. Addison-Wesley, 1979.
13. 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.
14. M. Maidl. The common fragment of CTL and LTL. In *IEEE Symposium on Foundations of Computer Science*, pages 643–652, 2000.
15. K. McMillan. Symbolic model checking, 1993.
16. R. Tzoref, M. Matusevich, E. Berger, and I. Beer. An optimized symbolic bounded model checking engine. In *CHARME*, 2003.
17. 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.