H-0258 (H0506-002) June 8, 2005
Computer Science

IBM Research Report

Automata Construction for On-the-Fly Model Checking PSL Safety Simple Subset

Sitvanit Ruah[1], Dana Fisman[1,2], Shoham Ben-David[3*]

[1]IBM Research Division
Haifa Research Laboratory
Mt. Carmel 31905
Haifa, Israel

[2]Weizmann Institute of Science

[3]University of Waterloo

[*]Work done while at IBM Haifa Research Lab

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

Automata construction for on-the-fly Model Checking PSL Safety Simple Subset*

Sitvanit Ruah[1] Dana Fisman[1,2] Shoham Ben-David[3]**

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

April 23, 2005

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 ([McM93]) 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 temporal language PSL [Acc04], which has been standardized by the Accellera standards organization, augments LTL with new language constructs, including Regular Expressions.

In order to be model-checked, a PSL formula needs to be translated into a verifiable form, usually an automaton. In this paper we present the translation into automata of a subset of PSL called SafetyPSL^{det}. This subset, as can be understood by its name, consists of *safety* properties. Such properties are of special interest, because they can be model checked efficiently, as will be explained in the sequel.

A property is considered to be *safe* if its violation can be detected by a finite path. Formally, consider a language L of finite and infinite words over an alphabet Σ. A finite word u over Σ is a bad prefix for L iff $\forall v \in \Sigma^* \cup \Sigma^\omega$, $uv \notin L$. A language L is a *safety language* if every word not in L has a finite bad prefix.

Model checking of a general linear property φ involves the construction of a Büchi automaton $\mathcal{B}_{\neg\varphi}$, of size exponential in φ, that accepts exactly all the infinite computations violating the property φ. Model checking φ is done by checking the emptiness of the product of the model M and $\mathcal{B}_{\neg\varphi}$ [VW86]. For safety properties, however, we can many times do better. Since computations violating a safety formula are all finite, a finite automaton $\mathcal{A}$ can detect them. Model checking can then be reduced to invariant checking, with the invariant being "$\mathcal{A}$ is not in an accepting state". Invariant checking is typically easier to

* This work was supported in part by the European Commission (FP6 STREP PROSYD contract no. 507219)

** The work of this author was done while working at IBM Haifa Research Lab

perform, as it can be done "on-the-fly" [BBL98], while searching the reachable state-space. A bug, if exists, can be found before computing the whole reachable state-space. The reduction to invariant checking is important also because simulation tools as well as many model-checking engines such as SAT-based engines perform better (if not only) on invariant checking.

This paper presents an automata construction for SafetyPSL^{det}, thus reducing the verification of formulas in this subset into invariant checking. The subset SafetyPSL^{det} of PSL formally defined in section 3.3, is an extension of both the safety simple subset of PSL [Acc04], and the safety common fragment of LTL and ACTL [Mai00]. We give a direct translation from SafetyPSL^{det} into a co-universal automaton, that does not involve an intermediate representation. In the next section we compare our work to related results. Section 3 gives some preliminaries. Section 4.2 is the main section of the paper, where the construction is given. The correctness of the construction is given in Section 5

2 Related Work

In this section we discuss related work on the subject. First we describe other results and then compare to ours.

Kupferman and Vardi [KV99] have shown that in the worst case, the size of an automaton on finite words accepting the set of bad prefixes of a safety LTL formula is doubly-exponential in the size of the formula. Since LTL is a subset of PSL this lower bound holds for PSL safety formulas as well.

Most optimizations used in the literature for automata for LTL formulas use an automata construction that works for a general LTL formula and then perform reductions on the resulting automaton. Such an approach is taken by Bloem et al [BRS99]. They classify Büchi automata into three classes according to the number of fixed-points that have to be computed for each class. When the Büchi automaton is *terminal* it can be composed with the model and used for on-the-fly model checking. Given an LTL formula they construct a Büchi automaton and then check in polynomial time whether it is terminal.

Another approach is based on known characteristics of the formula. Beer et al [BBL98] address RCTL formulas with a restricted syntax, and use the information on the syntax to construct efficient automata. They show how to transform a safety RCTL formula f into a regular expression accepting all bad prefixes of f. Thus producing a linear finite automaton that detects violation of the given property. This automaton can run in parallel to the system and detect states in which the property is violated. The verification is done on-the-fly during computation of the reachable states of the system.

Kupferman and Vardi [KV99] address LTL properties that are known to be safety properties and show how to construct automata for on-the-fly model checking for such formulas. For formulas that are syntactically safe, that is formulas in positive normal form constructed with the temporal connectives X and V, they construct an alternating automaton detecting bad prefixes, that is linear in the size of the formula. The automaton is then translated into a nondeterministic automaton on finite words which is of size exponential in the length of the formula.

Comparison to our work
Our algorithm

- constructs an optimized linear co-universal automata for a subset of syntactically safe LTL properties
- adds support for SERE formulas, which makes the size of the finite automata exponential in the length of the formula.
- constructs the automata without going through an alternating automata.
- is symbolic in the following sense: the alphabet of our automata are boolean expression over the given set of atomic propositions, rather than interpretation of the truth value of the given set of atomic propositions.

We take the approach of [BBL98] and extend it. Our approach is different than [BRS99] and [KV99] in the following aspects:

- We first checks the syntax of the formula falls in the subset we are interested in and then construct the most efficient automaton for it.
- It constructs a co-universal automaton on finite words without going through a Büchi automaton.
- For the LTL subset supported the automaton on finite words we construct is linear in the length of the formula while the automaton of [KV99] and [BRS99] is exponential in the length of the formula.
- In [BRS99,KV99] the alphabet is 2^P while in our automaton the alphabet is the set of boolean expressions over P. This "symbolic" nature makes our automaton more succinct.
- We support SERE formulas that are not addressed in [BRS99,KV99].

Our results extend the results of [BBL98] in several ways:

- Support for LTL instead of ACTL by using the definition of the common fragment of LTL and ACTL [Mai00].
- Support a larger subset of formulas, i.e. the closure of the union of the safety simple subset of PSL and the common fragment of LTL and ACTL.
- [BBL98] go through a regular expression before constructing the automaton. We construct the automaton directly and therefore enjoy the more relative strength advantage of an automaton over a regular expression.

3 Preliminaries

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 $w\epsilon = \epsilon w = w$. If $w = uv$ we say that u is a *prefix* of w, denoted $u \preceq w$, that v is a *suffix* of w, and that w is an *extension* of u, denoted $w \succeq u$. 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 $\{w \mid \exists w_1 \in L_1,\ \exists w_2 \in L_2 \text{ and } w = w_1w_2\}$. 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] The infinite concatenation of L to itself is denoted L^ω.

We denote the length of a word w by $|w|$. The empty word $w = \epsilon$ has length 0, a finite non-empty word $w = (s_0s_1s_2 \cdots s_n)$ has length $n+1$, and an infinite word has length ∞. 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).

Given a set V of typed state variable over finite domains. We define by Σ_V the set of type-consistent interpretations of V (assigning to each variable $p \in V$ a value in its domain). We use $\widehat{\Sigma}_V$ to denote the set $\Sigma_V \cup \{\top, \bot\}$. We use $\overline{w}$ to denote the *dual* of a word w which is the word obtained from w by replacing $\top$ with $\bot$ and vice versa. We denote by $Bool_V$ the set of boolean expression over V, which we identify with 2^{Σ_V}. For a boolean expression $b \in Bool_V$ and a letter $\ell \in \widehat{\Sigma}_V$ we define the boolean satisfaction relation $\Vdash$ as follows. For $\ell \in \Sigma_V$, we define $\ell \Vdash b \iff \ell \in b$. We define $\top \Vdash b$ and $\bot \nVdash b$.

3.1 A co-universal automaton on finite/infinite words (cua)

The finite automata for finite words we work with are co-universal automata. That is non-deterministic automata where acceptance is determined by the fact that *all* possible runs *do not* visit the set of bad states. These automata detect all bad prefixes of safety formulas in the subset we consider. Thus for a property f model checking can be done by verifying the invariant property $\mathsf{AG}\neg$"the automaton for f is in a bad state"

Definition 1 *A co-universal automaton automaton on finite/infinite words (*CUA*) $\mathcal{C}$ is a tuple $\mathcal{C} = \langle V, , Q, Q_0, \delta, B\rangle$ consisting of the following components:*

- *$V = \{p_1, \ldots, p_n\}$: A finite set of typed* state-variables*.*
- *Q: A finite set of automata locations.*
- *$Q_0 \subseteq Q$ - A set of initial locations.*
- *$\delta \subseteq Q \times Bool_V \times Q$ - A transition relation. This is a set of triples (q_1, b, q_2) relating location $q_1 \in Q$ to one of its successor locations $q_2 \in Q$ under an input letter $\ell \in \widehat{\Sigma}_V$ satisfying $b \in Bool_V$ (i.e., $\ell \Vdash b$).*
- *$B \subseteq Q$ - A set of bad locations.*

Let $\mathcal{C}$ be an CUA for which the above components have been defined. The input to $\mathcal{C}$ is a finite/infinite word $w = w^0w^1 \ldots \in (\widehat{\Sigma}_V^* \cup \widehat{\Sigma}_V^\omega)$. We define a *run* of $\mathcal{C}$ over a word $w = w^0w^1 \ldots$ to be a finite or infinite non-empty sequence $\sigma : q_0q_1 \ldots$ of locations in Q satisfying the requirements of *initiality* i.e. that $q_0 \in Q_0$; and of *consecution* i.e. that for each $j = 0, 1, \ldots,$ there exists $b \in Bool_V$ such that $(q_j, b, q_{j+1}) \in \delta$ and $w^j \Vdash b$. A run satisfying the requirement

[1] Here ω denotes the first transfinite ordinal number.

of *maximality* i.e. that it is either infinite, or terminates at a location q_k which has no successors is termed a *maximal run.* A word w is accepted by an CUA iff a run of CUA over w never reaches a state in B. A run $\sigma : q_0 q_1 q_2 \ldots q_{n+1} \ldots$ of $\mathcal{C}$ over a word w is said to be *accepting* w iff $q_i \notin B, 0 \leq i < |w|$. The CUA $\mathcal{C}$ accepts a word w iff every run $\sigma = q_0, q_1 \ldots$ of $\mathcal{C}$ over w is accepting. The CUA $\mathcal{C}$ is *deterministic* iff $\forall b \in Bool_V$ and $\forall q, q_1, q_2 \in Q$ such that $q_1 \neq q_2$: $(q, b, q_1) \in \delta$ implies $(q, b, q_2) \notin \delta$ (i.e., for each state q and each label b there is at most one transition labeled by letter b exiting the state q).

3.2 Regular Expressions

Regular expressions (REs) over a given alphabet Γ are defined as follows. For our purpose the alphabet will be $Bool_V$, the definition below, however is for a general alphabet Γ.

Definition 2 (Regular expressions (REs) over alphabet B).

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

The language defined by a regular expression is defined as follows [HU79].

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

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

The semantics given by PSL to REs relates REs over $Bool_V$ with words over $\widehat{\Sigma}_V$ rather than words over $Bool_V$ as done in the traditional semantics (Definition 3).

Definition 4 (Semantics of REs over boolean expressions) *Let V be a set of state variables. Let r, r_1, r_2 be regular expressions over the alphabet $Bool_V$. The semantics relates regular expressions over $Bool_V$ with finite words over the alphabet $\widehat{\Sigma}_V$. The notation $v \models r$, where r is an* RE *and v a finite word means that v models tightly r. The semantics of* RE*s are defined as follows, where b denotes a boolean expression in $Bool_V$, and r, r_1, and r_2 denote* RE*s over $Bool_V$.*

- $v \models b \iff |v| = 1$ *and* $v^0 \Vdash b$
- $v \models r_1 \cdot r_2 \iff \exists v_1, v_2$ *s.t.* $v = v_1 v_2$, $v_1 \models r_1$, *and* $v_2 \models r_2$
- $v \models r_1 \cup r_2 \iff v \models r_1$ *or* $v \models r_2$
- $v \models r^* \iff$ *either* $v = \epsilon$ *or* $\exists v_1, v_2$ *s.t.* $v_1 \neq \epsilon$, $v = v_1 v_2$, $v_1 \models r$ *and* $v_2 \models r^*$

We use $\mathbb{S}(r)$ *to denote the set* $\{w \in \widehat{\Sigma}_V^* \mid w \Vdash\!\equiv r\}$

PSL extends regular expression with operations such as *fusion* ($\circ$) and *intersection* ($\cap$) that do not add expressive power but add succinctness. The extended expressions are referred to as SEREs. It is well known that REs (and SEREs) are as expressive as automata on finite words, and in particular non-deterministic automata on finite words (NFAs). Moreover, for any RE r one can construct an NFA N_r linear in the size of r accepting the same language [HU79,BFR04a]. Transformation of SEREs into equivalent NFA appear in [BFR04b]. In the following we do not distinguish between REs and SEREs, and refer to both simply as regular expressions or REs.

The distinction between the langauge of regular expression (as given in Definition 3) and its semantics (as given in Definition 4) involves some subtleties. The language of a regular expression is defined over a syntactic alphabet while the semantics is defined over a semantic alphabet. In particular, the language of the intersection of the letters a and b over the alphabet of boolean expressions is given by $\mathbb{L}(a \cap b) = \mathbb{L}(a) \cap \mathbb{L}(b) = \emptyset$ (because a syntactic letter can be either a or b, but not both), the semantics of the intersection of a and b is given by $\mathbb{S}(a \cap b) = \{w \in \widehat{\Sigma}_V^* \mid |w| = 1 \text{ and } w \Vdash a \text{ and } w \Vdash b\} \neq \emptyset$ (because a semantic letter ℓ can satisfy both a and b at the same time).

3.3 The Logic

The logic SafetyPSL^{det} is the closure of the union of the safety common fragment of LTL and ACTL [Mai00] and the safety simple subset of PSL [Acc04]. It is formally defined as follows.

Definition 5 (SafetyPSL^{det} formulas) *If b is a boolean expression, r is an* RE *and f, f_1 and f_2 are SafetyPSL^{det} then the following are in SafetyPSL^{det}:*

1. $b!$
2. r
3. $f_1 \wedge f_2$
4. $\mathsf{X!}\ f$
5. $(b \wedge f_1)\ \vee\ (\neg b \wedge f_2)$
6. $[(b \wedge f_1)\ \mathsf{W}\ (\neg b \wedge f_2)]$
7. $r \mapsto f$

The safety common fragment of LTL and ACTL, denoted SafetyLTL^{det} is the subset of formulas of SafetyPSL^{det} involving no regular expressions.

The safety part of the PSL simple subset defined in the PSL language reference manual [Acc04] is a subset of SafetyPSL^{det}. This since the restrictions on the operands of $\vee$ and W in [Acc04] are stronger then the restrictions in Definition 5. Therefore the algorithm we give here applies to the PSL safety simple subset.

Definition 6 (Semantics of SafetyPSL^{det}) *Let w be a finite or infinite word, b be a boolean expression, r, r_1, r_2 REs, and f, f_1, f_2 SafetyPSL^{det} formulas. We* *use $\models$ to define the semantics of SafetyPSL^{det} formulas: If $w \models f$ we say that w models (or satisfies) f.*[2]

1. $w \models b! \iff |w| > 0$ *and* $w^0 \Vdash b$
2. $w \models r \iff \forall$ *finite non-empty* $v \preceq w$, $\exists$ *non-empty* $u \preceq v\top^\omega$ *s.t.* $u \models\!\!\equiv r$.
3. $w \models f_1 \wedge f_2 \iff w \models f_1$ *and* $w \models f_2$
4. $w \models \mathsf{X!}\ f \iff |w| > 1$ *and* $w^{1..} \models f$
5. $w \models (b \wedge f_1)\ \vee\ (\neg b \wedge f_2) \iff$
   either ($w^0 \Vdash b$ *and* $w \models f_1$) *or* ($w^0 \Vdash \neg b$ *and* $w \models f_2$)
6. $w \models [(b \wedge f_1)\ \mathsf{W}\ (\neg b \wedge f_2)] \iff$
   either ($\exists k < |w|$ *s.t.* $w^k \Vdash \neg b$, $w^{k..} \models f_2$, *and* $\forall j < k$, $w^j \Vdash b$ *and* $w^{j..} \models f_1$)
   or ($\forall j < |w|$, $w^j \Vdash b$ *and* $w^{j..} \models f_1$)
7. $w \models r \mapsto f \iff \forall j < |w|$ *s.t.* $\overline{w}^{0..j} \models\!\!\equiv r$, $w^{j..} \models f$

We use $[\![\varphi]\!]$ *to denote the set* $\{w \in \widehat{\Sigma}^\infty \mid w \models \varphi\}$

Note that the semantics of a weak RE r is such that r may be assumed non-empty. That is $[\![r]\!] = [\![r']\!]$ where r' is such that $\mathbb{L}(r') = \mathbb{L}(r) \setminus \{\epsilon\}$. Similarly, by the semantics of suffix implication $[\![r \mapsto f]\!] = [\![r' \mapsto f']\!]$ where r' and f' are such that $\mathbb{L}(r') = \mathbb{L}(r) \setminus \{\epsilon\}$ and $[\![f']\!] = [\![f]\!] \setminus \{\epsilon\}$. We therefore assume without loss of generality that $\epsilon \notin \mathbb{L}(r)$ and that $\epsilon \notin [\![f]\!]$ for any sub-formula r or $r \mapsto f$.

4 Automata Construction for SafetyPSL^{det} Formulas

In this section we show how to construct a CUA for any SafetyPSL^{det} formula. This CUA shall be used for checking whether it holds on a given model as follows.

In order to check that the model M (given as a discrete transition system DTS D_M, see Section A.1) satisfies a SafetyPSL^{det} formula f we perform the following:

1. Construct a CUA for the formula f denoted CUA(f) (as described in Section 4.2 below).
2. Construct the DTS corresponding to CUA(f) denoted $\mathcal{D}_f$ (as described in Section A.2 of the appendix).
3. Verify that $\mathcal{D}_M \,|||\, \mathcal{D}_f \models \mathsf{AG}\ \neg($ at bad state of $\mathcal{D}_f)$.

For most of the operators we build a non-deterministic CUA. An exception is the weak RE operator r, where we need determinization. Intuitively, the reason weak REs need determinization is as follows. The semantics of a weak RE r

[2] The semantics given here is equivalent to the semantics given in PSL [Acc04]. The semantics given in PSL are defined directly for a set of core operators, and by syntactic sugaring for the other operators. Here, we gave a direct semantics to some operators which are given as syntactic sugaring in PSL. The proof of equivalence appears in [HFE04].

stipulates that either there exists a prefix of the word that matches tightly r or every prefix of the given word, can be extended with some number of $\top$'s so that the resulting word matches tightly r. That is, an automaton for r should accept a word if *there exists* an accepting run on a prefix of a word, possibly extended by some $\top$'s. However, the construction works with co-universal automata that accepts a word iff *all* runs are accepting. Applying determinization to the NFA for r we obtain an automaton with a single run, thus treating it as a universal automaton makes no difference.

First we describe the construction for a weak RE, then we describe the construction for the other SafetyPSL^{det} operators.

4.1 Constructing a deterministic cua accepting r

Our construction makes use of non-deterministic automata on finite words (NFA) accepting traditional regular expression. That is, given a regular expression r we assume $N_r = \langle Bool_V, Q, Q_0, \delta, F\rangle$ is an NFA accepting $\mathbb{L}(r)$ [HU79].[3] We say that N_r is a "syntactic NFA" since it accepts $\mathbb{L}(r)$ rather than $\mathbb{S}(r)$. That is the alphabet of N_r is a subset of $Bool_V$ and N_r does not check the semantics of letters in its input, but rather checks that they are syntactically the same as a letter in δ. For example, let $r = a \cdot b * \cdot c$ then the word $w_1 = abc \in \mathbb{L}(r)$ and therefore is accepted by N while $w_2 = (a \wedge b) \cdot b * \cdot c \notin \mathbb{L}(r)$ and is not accepted by N (though $w_2 \in \mathbb{S}(r)$).

As noted in Section 3.2, there are some subtleties between the language of a regular expression and its semantics. A thorough study of the precise nature of these differences is beyond the scope of this paper. The differences spill over into the theory of the relationship between regular expressions and finite automata and the theory of finite automata themselves. For instance, consider the automaton of Figure 1. In traditional automata theory (i.e. judging by its language) it is deterministic. In our usage (i.e. judging by its semantics) it is non-deterministic, and its deterministic version is given in Figure 2.

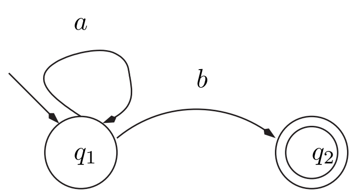

Fig. 1. A non-deterministic automaton accepting the language of $a^* \cdot b$

[3] We assume N has no ϵ transitions.

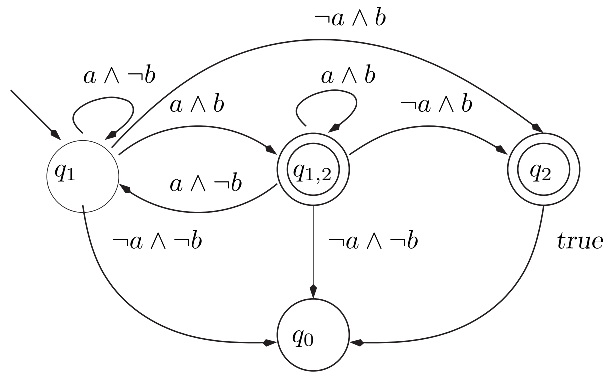

Fig. 2. Deterministic version of Figure 1, accepting the semantics of $a^* {\cdot} b$

The determinization construction

Proposition 7. *Let r be an* RE *over the state variables V and let w be a word over Σ_V. There exists a deterministic co-automaton $\mathcal{C}_r$ of size $2^{O(r)}$ such that*

$$\mathcal{C}_r \textit{ accepts } w \textit{ iff } w \models r.$$

Below we provide the construction, we give the proof in Section 5.

Let $N = \langle Bool, Q, Q_0, \delta, F\rangle$ be a non-deterministic finite automata on finite words. We build a deterministic co-automaton corresponding to N, $\text{DCA}(N) = \langle V, Q^d, Q_0^d, \delta^d, B^d\rangle$ using the subset construction procedure with some modifications to capture the "semantic" nature of the alphabet *Bool*.

Let V be the set of state variables over which *Bool* is defined. Let q_{sink} be a new state. States in Q^d are subsets of $Q \cup \{q_{sink}\}$. The initial state of $\text{DCA}(N)$ is $Q_0^d = \{q \mid q \in Q_0\}$. Let S be a subset of $Q \cup \{q_{sink}\}$. To construct all the out going edges from S in the deterministic co-automaton we need to enumerate the set of mutually-disjoint conditions that can hold at state S. In case we fall of N we add a corresponding transition to the sink state q_{sink}. For example, if $S = \{1, 2\}$ and in N we have the edges $(1, a, 2)$ and $(2, b, 3)$ then in the DCA we will have the edges $(\{1, 2\}, a \wedge b, \{2, 3\})$, $(\{1, 2\}, \neg a \wedge b, \{q_{sink}, 3\})$, $(\{1, 2\}, a \wedge \neg b, \{2, q_{sink}\})$, $(\{1, 2\}, \neg a \wedge \neg b, \{q_{sink}\})$.

Formally, a state S in the DCA is expanded as follows. Denote the set of outgoing transitions from states in S in N by

$$\text{outgoing}(S) = \{(s_1, \ell, s_2) \in \delta | s_1 \in S\}.$$

Denote by conditions(S) the set

$$\{\ell \in Bool | (s_1, \ell, s_2) \in \text{outgoing}(S)\}.$$

Every subset P of conditions in conditions(S) forms one edge in the DCA. The condition on this edge is

$$\text{det\_edge\_cond}(P) = (\bigwedge_{\ell \in P} \ell) \wedge (\bigwedge_{\ell \in \text{conditions}(S) \setminus P} \neg \ell).$$

Denote by succ(S, P) the DCA state reached from S when det_edge_cond(P) holds. That is

$$\text{succ}(S,P) = \begin{cases} \{s_2 \in Q | (s_1, \ell, s_2) \in \text{outgoing}(S), \ell \in P\} & \text{if } P = \text{conditions}(S) \\ \{s_2 \in Q | (s_1, \ell, s_2) \in \text{outgoing}(S), \ell \in P\} \cup \{q_{sink}\} & \text{otherwise} \end{cases}$$

Define $\delta^d(S) = \bigcup_{P \subseteq \text{conditions}(S)} (S, \text{det\_edge\_cond}(P), \text{succ}(S, P))$. Define

$$\hat{\delta} = \bigcup_{S \in 2^{Q \cup \{q_{sink}\}}} \delta^d(S)$$

The automaton given by the above set of state $(2^{Q \cup \{q_{sink}\}})$, the above initial state $(\{q | q \in Q_0\})$, the transition relation $\hat{\delta}$ and the set of bad states $\{q_{sink}\}$ is a deterministic co-automaton accepting the language of the given RE. However, we are interested in the minimal deterministic co-automaton. We thus build the set of reachable states, and restrict the set of states as well as the other components accordingly.

Denote by Reach$(Q \cup \{q_{sink}\})$ the set of states in $2^{Q \cup \{q_{sink}\}}$ that are reachable from Q_0^d. That is

$$\begin{aligned} \text{Reach}(Q \cup \{q_{sink}\}) = \{S \in 2^{Q \cup \{q_{sink}\}} | & S = Q_0^d \text{ or} \\ & \text{there exist } S_0, \ldots, S_{n-1} \in 2^{Q \cup \{q_{sink}\}}, S_0 = Q_0^d \\ & \ell_0, \ldots, \ell_{n-1} \in Bool, \ell_i \not\equiv \mathit{false}, \\ & (S_i, \ell_i, S_{i+i}) \in \hat{\delta}, 0 \leq i < n-1, \\ & (S_i, \ell_{n-1}, S) \in \hat{\delta}\} \end{aligned}$$

DCA$(N) = \langle V, Q^d, Q_0^d, \delta^d, B^d \rangle$ where

- $Q^d = \text{Reach}(Q \cup \{q_{sink}\})$
- $Q_0^d = \{q | q \in Q_0\}$
- $\delta^d = \bigcup_{S \in Q^d} \delta^d(S)$
- $B^d = \{q_{sink}\}$.

4.2 The full construction

Theorem 8. *Let f be a SafetyPSL^{det} formula over the state variables V and let w be a word over Σ_V. Then there exists an* CUA $\mathcal{C}_f$ *such that*

$$w \models f \iff \mathcal{C}_f \text{ accepts } w$$

and $\mathcal{C}_f$ is of size $O(|f|)$ if f contains no sub-formulas of the form r (weak regular expressions) and is of size $2^{O(|f|)}$ otherwise.

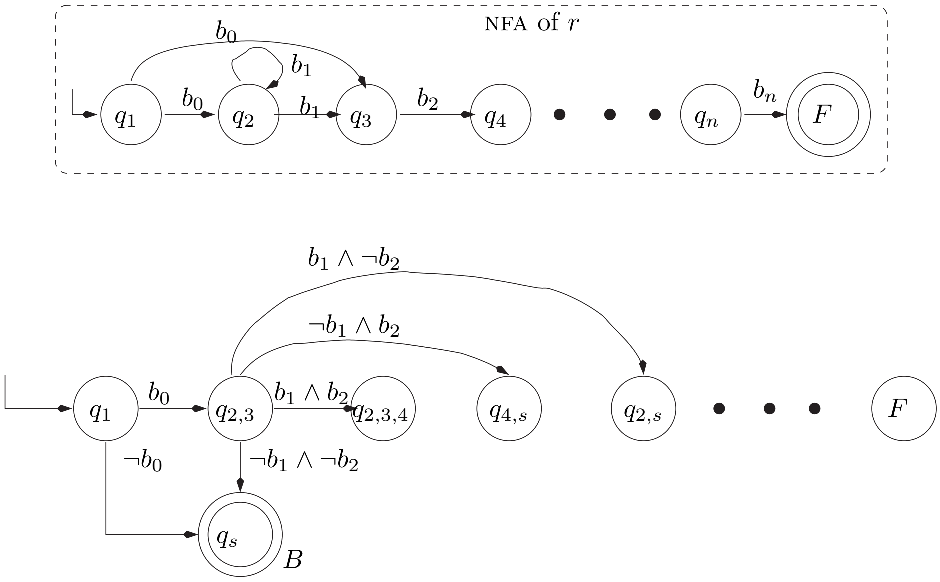

Fig. 3. A DCA for r

Below we provide the construction, we give the proof in Section 5.

Let r be an RE such that $\epsilon \notin \mathbb{L}(r)$, b a boolean expression, f_1, f_2, f formulas in SafetyPSL^{det}. For the induction hypothesis, let CUA$(f_1) = \langle V^1, Q^1, Q_0^1, \delta^1, B^1\rangle$ and CUA$(f_2) = \langle V^2, Q^2, Q_0^2, \delta^2, B^2\rangle$.

1. CUA$(b!) = \langle V^{\mathcal{C}}, \{q_0, q_1, q_2\}, \{q_1\}, \delta^{\mathcal{C}}, \{q_0\}\rangle$ where $V^{\mathcal{C}}$ is the set of state variables in b and $\delta^{\mathcal{C}} = \{(q_1, b, q_2), (q_1, \neg b, q_0)\}$.
2. r
   The on-the-fly DTS for r is constructed as follows:
   (a) Construct an NFA $N = \langle Bool_V, Q, Q_0, \delta, F\rangle$ accepting $\mathbb{L}(r)$.
   (b) CUA$(r) =$ DCA(N) as constructed in Section 4.1.
3. $f_1 \wedge f_2$.
   A run of CUA$(f_1 \wedge f_2)$ has a non-deterministic choice between a run of Q_1 and a run of $\mathcal{C}_2$. In any choice it should not reach a state in either B_1 or B_2.
   Formally,
   CUA$(f_1 \wedge f_2) = \langle V^{\mathcal{C}}, Q^{\mathcal{C}}, Q_0^{\mathcal{C}}, \delta^{\mathcal{C}}, B^{\mathcal{C}}\rangle$, where
   (a) $V^{\mathcal{C}} = V^1 \cup V^2$.
   (b) $Q^{\mathcal{C}} = Q^1 \cup Q^2$.
   (c) $Q_0^{\mathcal{C}} = Q_0^1 \cup Q_0^2$.
   (d) $\delta^{\mathcal{C}} = \delta^1 \cup \delta^2$.
   (e) $B^{\mathcal{C}} = B^1 \cup B^2$
   The resulting CUA is described in Figure 4.

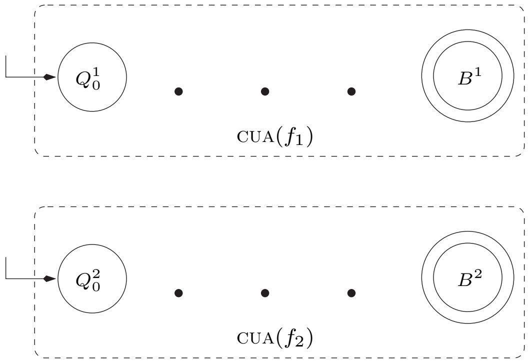

Fig. 4. An CUA for $f_1 \wedge f_2$

4. X!f.
   Let CUA$(f) = \langle V, Q, Q_0, \delta, B\rangle$.
   CUA(X!f) $= \langle V, Q \cup \{s_0\}, \{s_0\}, \delta^{\mathcal{C}}, B\rangle$ where s_0 is a new state and
   $\delta^{\mathcal{C}} = \delta \cup \bigcup_{q \in Q_0}(s_0, true, q)$ The resulting CUA is described in Figure 5

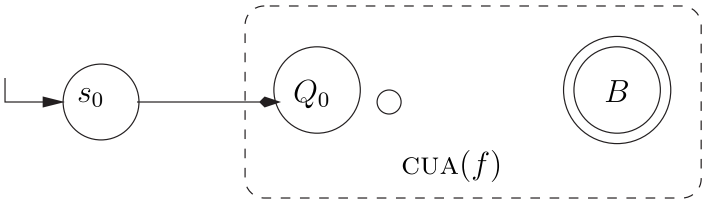

Fig. 5. An CUA for X!f

5. $(b \wedge f_1) \vee (\neg b \wedge f_2)$.
   A run of CUA$(b \wedge f_1 \vee \neg b \wedge f_2)$ starts in a new state s_0, if b holds it continues on $\mathcal{C}_1$ otherwise it continues on $\mathcal{C}_2$.
   CUA$(b \wedge f_1 \vee \neg b \wedge f_2) = \langle V^{\mathcal{C}}, Q^{\mathcal{C}}, Q_0^{\mathcal{C}}, \delta^{\mathcal{C}}, B^{\mathcal{C}}\rangle$, where
   (a) $V^{\mathcal{C}} = V^1 \cup V^2$.
   (b) $Q^{\mathcal{C}} = \{s_0\} \cup Q^1 \cup Q^2$.
   (c) $Q_0^{\mathcal{C}} = \{s_0\}$.
   (d)
$$\begin{aligned}\delta^{\mathcal{C}} = {} & \delta^1 \cup \delta^2 \cup \\ & \textstyle\bigcup_{q_1 \in Q_0^1} \bigcup_{(q_1,\ell,q_2) \in \delta^1} (s_0, b \wedge \ell, q_2) \\ & \textstyle\bigcup_{q_1 \in Q_0^2} \bigcup_{(q_1,\ell,q_2) \in \delta^2} (s_0, \neg b \wedge \ell, q_2)\end{aligned}$$
   (e) $B^{\mathcal{C}} = B^1 \cup B^2$
   The resulting CUA is described in Figure 6.

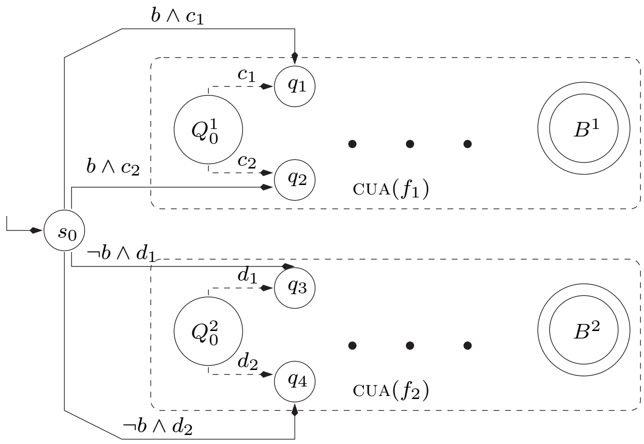

Fig. 6. An CUA for $(b \wedge f_1) \vee (\neg b \wedge f_2)$

6. $(p \wedge f_1)\mathsf{W}(\neg p \wedge f_2)$.
   (a) $V^{\mathcal{C}} = V^1 \cup V^2$.
   (b) $Q^{\mathcal{C}} = \{s_0\} \cup Q^1 \cup Q^2$.
   (c) $Q_0^{\mathcal{C}} = \{s_0\}$.
   (d)
$$\begin{aligned}\delta^{\mathcal{C}} = &(s_0, p, s_0) \cup \delta^1 \cup \delta^2 \cup \\ &\bigcup_{q_1 \in Q_0^1} \bigcup_{(q_1,\ell,q_2) \in \delta^1} (s_0, b \wedge \ell, q_2) \\ &\bigcup_{q_1 \in Q_0^2} \bigcup_{(q_1,\ell,q_2) \in \delta^2} (s_0, \neg b \wedge \ell, q_2)\end{aligned}$$
   (e) $B^{\mathcal{C}} = B^1 \cup B^2$

   The resulting CUA is described in Figure 7.
7. $r \mapsto f$

   Let $N = \langle Bool_V Q, Q_0, \delta, F\rangle$ accepting $\mathbb{L}(r)$. Let $\mathcal{C}_1 = \langle V^1, Q^1, Q_0^1, \delta^1, B^1\rangle$ be defined as follows:
   - V^1 is the set of state variables in r
   - $Q^1 = Q \cup \{q_{sink}\}$
   - $Q_0^1 = Q_0$
   - $\delta^1 = \bigcup_{(q_1,\ell,q_2) \in \delta} \{(q_1, \ell, q_2), (q_1, \neg\ell, q_{sink})\}$
   - $B^1 = F$.

   The resulting CUA is described in Figure 8, as standalone it accepts $[\![r \mapsto \mathit{false}]\!]$.
   Let $\mathcal{C}_2 = \text{CUA}(f) = \langle V^2, Q^2, Q_0^2, \delta^2, B^2\rangle$ be the CUA, as constructed by induction for f.
   Then $\text{CUA}(r \mapsto f) = \langle V^{\mathcal{C}}, Q^{\mathcal{C}}, Q_0^{\mathcal{C}}, \delta^{\mathcal{C}}, B^{\mathcal{C}}\rangle$ is constructed by concatenation of $\mathcal{C}_1$ and $\mathcal{C}_2$ as follows:
   (a) $V^{\mathcal{C}} = V^1 \cup V^2$.
   (b) $Q^{\mathcal{C}} = Q^1 \cup Q^2 \cup \{q_{bad}\}$, where q_{bad} is a new state.
   (c) $Q_0^{\mathcal{C}} = Q_0^1$.

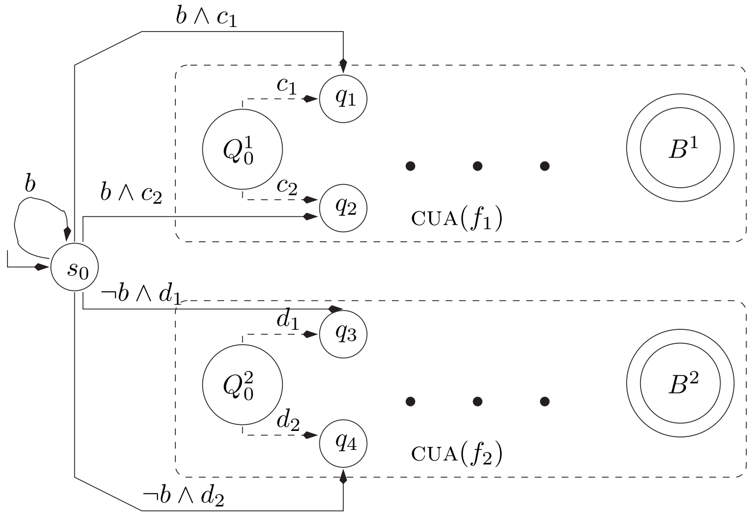

Fig. 7. An CUA for $(b \wedge f_1)$ W $(\neg b \wedge f_2)$

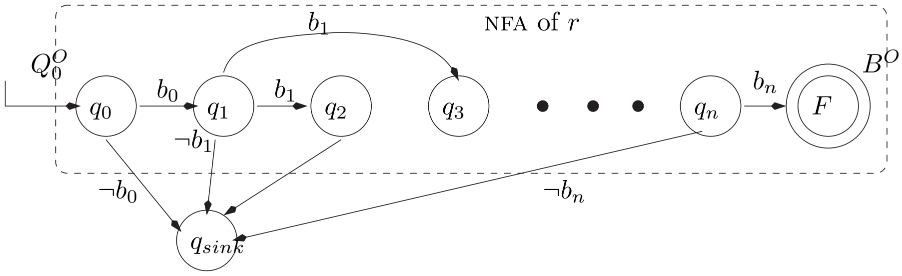

Fig. 8. An CUA for $r \mapsto$ *false*

(d) $\delta^{\mathcal{C}} = \begin{array}{l} \delta^1 \cup \delta^2 \cup \{(q_1, \ell_1 \wedge \neg\ell_2, q_{bad}) \mid q_2 \in B^1, (q_1, \ell_1, q_2) \in \delta^1\} \cup \\ \{(q_1, \ell_1 \wedge \ell_2, q_4) \mid q_2 \in B^1, q_3 \in Q_0^2, (q_1, \ell_1, q_2) \in \delta^1, (q_3, \ell_2, q_4) \in \delta^2\} \end{array}$.

(e) $B^{\mathcal{C}} = B^2 \cup \{q_{bad}\}$

The resulting CUA is described in Figure 9.

5 Correctness of the Construction

The proofs make use of the following Lemma [BFR04a, Lemma 10].

Lemma 9. *Let V be a set of state variables, w a word over Σ_V and r an* RE *over $Bool_V$. Then $w \models\!\equiv r$ iff either $\epsilon \in \mathbb{L}(r)$ and $w = \epsilon$ or there exists a word $\beta = b_0 \ldots b_n \in \mathbb{L}(r)$ such that $w_i \Vdash b_i$ for every $0 \leq i \leq n$.*

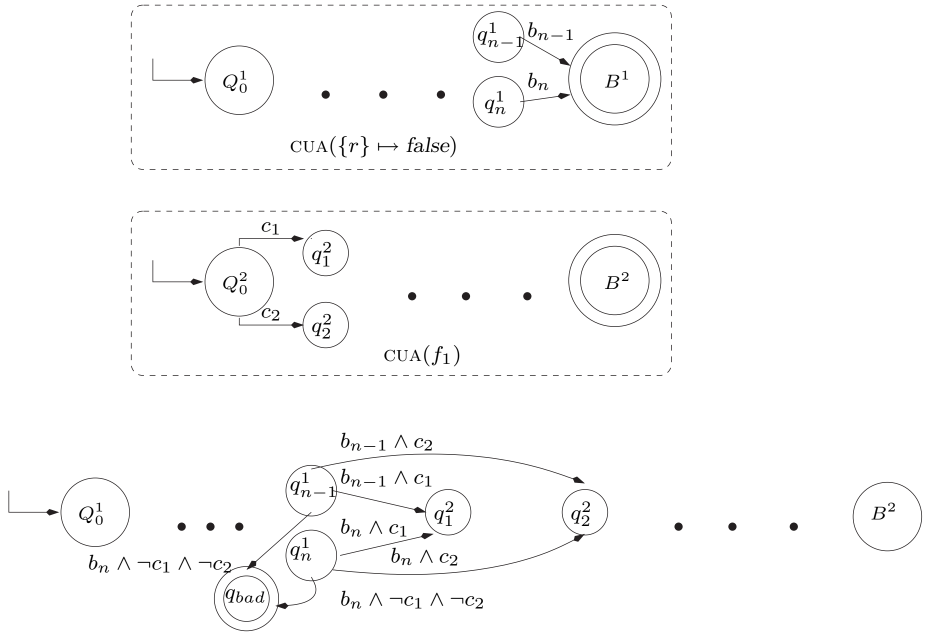

Fig. 9. An CUA for $r \mapsto f_1$

Proposition 7. *Let r be an* RE *over the state variables V and let w be a word over Σ_V. There exists a deterministic co-automaton $\mathcal{C}_r$ of size $2^{O(r)}$ such that*

$$\mathcal{C}_r \textit{ accepts } w \textit{ iff } w \models r.$$

Proof. Let $N = \langle Bool, Q, Q_0, \delta, F \rangle$ be an NFA accepting $\mathbb{L}(r)$. Let $\mathcal{C}_r = \text{DCA}(N) = \langle V, Q^d, Q_0^d, \delta^d, B^d \rangle$ be the deterministic co-autoamton as described in Section 4.2

If.

$w \not\models r$
$\iff$ there exists a minimal $j < |w|$ such that $w^{0..j}\top^\omega \not\models r!$.
$\implies w^{0..j-1}\top^\omega \models r!$
$\iff \exists u \preceq w^{0..j-1}\top^\omega$ such that $u \models r$
Let $S_0 \ldots S_{j+1}$ be a run of DCA(N) over $w^{0..j}$. By the construction, and the definition of a run of an CUA, for every $0 \leq i < j$, There exist $s_i \in S_i, s_{i+1} \in S_{i+1}, v^i \in Bool$ such that $(s_i, v^i, s_{i+1}) \in \delta$, and $w^i \models v_i$.
$w^j \not\models v^j$ for every $v^j \in Bool$ such that $\exists s' \in S_j, s'' \in Q$ such that $(s', v^j, s'') \in \delta$. Assume otherwise, that is, $\exists v^j \in Bool, \exists s' \in S_j, s'' \in Q$ such that $(s', v^j, s'') \in \delta$ and $w^j \models v^j$.
$\implies S_i \notin B$ for $0 \leq i < j$. Since N is reduced (every run can be extended to an accepting run), $s_0 s_1 \ldots s_j$ can be extended to a run accepting $w^{0..j}\top^k$ for

some k. Therefore $w^{0..j}\top^k \not\models r$ contradicting the assumption on w. Therefore $w^j \not\models v^j$ for every $v^j \in \mathit{Bool}$ such that $\exists s' \in S_j, s'' \in Q$ and $(s', v^j, s'') \in \delta$. By the construction of $\textsc{dca}(r)$, $S_{j+1} = \{q_{sink}\}$
$\Longrightarrow$ $\textsc{dca}(N)$ does not accept w.

Only if.

Assume $\textsc{dca}(N)$ does not accept w.
$\Longrightarrow$ the run $S_0 S_1 \ldots$ of $\textsc{dca}(N)$ over w satisfies that there exists a minimal j such that $S_{j+1} = \{q_{sink}\}$. Let $\sigma : s_0 s_1 \ldots s_j$ be such that $s_i \in S_i, 0 \leq i < j$ and there exist $v^0 \ldots v^{j-1}$ such that $(s_i, v^i, s_{i+1}) \in \delta$ and $w^i \models v^i, 0 \leq i < j$. Then for every extension of σ to an accepting run

$$s_0 \xrightarrow{v^0} s_1 \xrightarrow{v^1} s_2 \quad \cdots \quad s_j \xrightarrow{v^j} s_{j+1} \quad \cdots \quad s_k$$

of N, it holds that $w^j \not\models v^j$.
$\Longrightarrow$ $\forall v = v^0 \ldots v^k \in \mathbb{L}(r)$ such that $w^i \models v^i, 0 \leq i < j$ it holds that $w^j \not\models v^j$
$\Longrightarrow$ (by Lemma 9) $w^{0..j}\top^k \not\models r$ for every k
$\Longrightarrow$ $w \not\models r$.

□

Lemma 1. *Let f be an SafetyPSL^{det} formula and let* $\textsc{cua}(f) = \langle V^{\mathcal{C}}, Q^{\mathcal{C}}, Q_0^{\mathcal{C}}, \delta^{\mathcal{C}}, B^{\mathcal{C}} \rangle$ *be as constructed in Section 4.2. Then $Q_0^{\mathcal{C}} \cap B^{\mathcal{C}} = \emptyset$.*

Proof. By induction on the structure of f.

- $f = b!$ where $b \in \mathit{Bool}$.
  $Q_0^{\mathcal{C}} = \{q_1\}, B^{\mathcal{C}} = \{q_0\} \Longrightarrow Q_0^{\mathcal{C}} \cap B^{\mathcal{C}} = \emptyset$.
- $f = r$.
  $Q_0^{\mathcal{C}} = \{Q_0\}, B^{\mathcal{C}} = \{q_{sink}\}$
  $\Longrightarrow Q_0^{\mathcal{C}} \cap B^{\mathcal{C}} = \emptyset$

Assume the Lemma holds for f_1 and f_2.
Let $\textsc{cua}(f_1) = \langle V^1, Q^1, Q_0^1, \delta^1, B^1 \rangle$ and $\textsc{cua}(f_2) = \langle V^2, Q^2, Q_0^2, \delta^2, B^2 \rangle$.

- $f = f_1 \wedge f_2$.
  $Q_0^{\mathcal{C}} \cap B^{\mathcal{C}} = (Q_0^1 \cup Q_0^2) \cap (B^1 \cup B^2)$.
  (By the induction hypothesis and since $Q^1 \cap Q^2 = \emptyset$) $Q_0^{\mathcal{C}} \cap B^{\mathcal{C}} = \emptyset$.
- $f = \mathsf{X}! f_1$
  $Q_0^{\mathcal{C}} = \{s_0\}$ $s_0 \notin B^{\mathcal{C}}$.
- $f = (b \wedge f_1) \vee (\neg b \wedge f_2)$
  $Q_0^{\mathcal{C}} = \{s_0\}$, $s_0 \notin B^1 \cup B^2 = B^{\mathcal{C}}$.
- $f = (b \wedge f_1)\mathsf{W}(\neg b \wedge f_2)$
  $Q_0^{\mathcal{C}} = \{s_0\}$, $s_0 \notin B^1 \cup B^2 = B^{\mathcal{C}}$.
- $f = r \mapsto f_1$
  $Q_0^{\mathcal{C}} \cap B^{\mathcal{C}} = Q_0^1 \cap (B^2 \cup \{q_{bad}\}) = \emptyset$.

Lemma 2. *Let w be a word over Bool,r an* RE *and j minimal such that $w^{0..j}\top^\omega \not\models r!$. For every $v = v^0 \ldots v^k \in \mathbb{L}(r)$ $(\forall i < j : w^i \models v^i) \rightarrow w^j \not\models v^j$*

Proof. Let w such that $w^{0..j}\top^\omega \not\models r!$, and let $v = v^0 \dots v^k \in \mathbb{L}(r)$ such that $\forall i < j : w^i \models v^i$. Assume towards contradiction that $w^j \models v^j$. But then $\exists k :$ $w^{0..j}\top^k \models\!\equiv r$. Therefore $w^{0..j}\top^\omega \models r!$ contradiction.

Theorem 8. *Let f be a SafetyPSL^{det} formula over the state variables V and let w be a word over Σ_V. Then there exists an* CUA $\mathcal{C}_f$ *such that*

$$w \models f \quad \Longleftrightarrow \quad \mathcal{C}_f \text{ accepts } w$$

and $\mathcal{C}_f$ is of size $O(|f|)$ if f contains no sub-formulas of the form r (weak regular expressions) and is of size $2^{O(|f|)}$ otherwise.

Proof. Let $\mathcal{C}_f$ be CUA(f) as described in Section 4.2. The proof is by induction on the structure of f.

– $f = b$ where $b \in Bool$.
  Let w be such that $w \models b$ and let $s_0, s_1, s_2, \dots$ be a run of CUA(b) over w. By initiality $s_0 = q_1$. By consecution, there exists ℓ such that $(s_0, \ell, s_1) \in \delta$ and and $w^0 \Vdash \ell$. By the transition relation there are two option for ℓ: either b or $\neg b$. Since $w \models b$ the first option holds. Thus $\ell = b$ and $s_2 = q_2$, for $i \geq 1$. Since both $q_1, q_2 \notin B$, any state of the run does not visit B therefore CUA(b) accepts w.
  Assume CUA(b) accepts w and let $s_0, s_1, s_2, \dots$ be an accepting run of CUA(b) on w. By initiality $s_0 = q_1$. Since the run is accepting, $\forall i > 0$, $s_i \notin B = \{q_0\}$. By the transition relation, the only possibility is that $s_i = q_2$ for every $i > 0$. Thus $w_0 \Vdash b$. Therefore $w \models b$.
– $f = r$.
  Follows from Proposition 7 since CUA(r) = DCA(N).

Assume the theorem holds for $f_1, f_2 \in$ SafetyPSL^{det} and every word w over $Bool$.

– $f = f_1 \wedge f_2$
  $w \models f_1 \wedge f_2$ iff $w \models f_1$ and $w \models f_2$ iff (by the induction hypothesis) CUA(f_1) accepts w and CUA(f_2) accepts w iff for every run $q^1_0 \dots q^1_{|w|}$ of CUA(f_1) over w $q^1_i \notin B^1$ for $0 \leq i \leq |w|$ and for every run $q^2_0 \dots q^2_{|w|}$ of CUA(f_2) over w $q^2_i \notin B^2$ for $0 \leq i \leq |w|$ iff for every run $q_0 \dots q_{|w|}$ of CUA$(f_1 \wedge f_2)$ over w $q_i \notin B$ for $0 \leq i \leq |w|$ (since every run of CUA$(f_1 \wedge f_2)$ over w is either a run of CUA(f_1) or a run of CUA(f_2) over w. iff
  CUA$(f_1 \wedge f_2)$ accepts w.
– $f = \mathsf{X}! f_1$
  Denote $w = w_0 w_1 \dots$.
  $w \models \mathsf{X}! f_1$ iff $w^{1..} \models f_1$ iff (by the induction hypothesis) CUA(f_1) accpets $w^{1..}$. Let $s_0 q_0 q_1 \dots$ be a run of CUA$(\mathsf{X}! f_1)$ over w. By the construction $q_0 q_1 \dots$ is a run of CUA(f_1) on $w^{1..}$. Therefore $q_i \notin B^1$ for $i \geq 0$. By Lemma 1, $s_0 \not{i}nB$ therefore CUA$(\mathsf{X}! f_1)$ accepts w.
  Assume CUA$(\mathsf{X}! f)$ accpets w. Let $s_0 q_0 q_1 \dots$ be a run of CUA$(\mathsf{X}! f_1)$ over w.

$q_i \not{i}nB = B^1, i \geq 0$. By the construction $q_0q_1\ldots$ is a run of $\textsc{cua}(f_1)$ over $w^{1..}$. Therefore $\textsc{cua}(f_1)$ accepts $w^{1..}$. By the induction hypothesis $w^{1..} \models f_1$. Therefore $w \models \mathsf{X!}f_1$.

- $(b \wedge f_1) \;\vee\; (\neg b \wedge f_2)$
  $w \models (b \wedge \varphi_1) \;\vee\; (\neg b \wedge \varphi_2)$ iff
  $w \models b$ and $w \models f_1$ or $w \models \neg b$ and $w \models f_2$ iff (by the induction hypothesis)
  $w \models b$ and $\textsc{cua}(f_1)$ accepts w or $w \models \neg b$ and $\textsc{cua}(f_2)$ accepts w.
  Let $s_0q_1q_2\ldots$ be a run of $\textsc{cua}((b \wedge \varphi_1) \;\vee\; (\neg b \wedge \varphi_2))$ over w.
  If $w \models b$ then $\textsc{cua}(f_1)$ accepts w and $\exists q_0 \in Q_0^1$ such that $q_0q_1q_2\ldots$ is a run of $\textsc{cua}(f_1)$ over w. Therefore $q_i \notin B^1$ and therefore $q_i \notin B$ for $i \geq 0$. In addition by Lemma 1 $s_0 \notin B$. It follows that $\textsc{cua}((b \wedge \varphi_1) \;\vee\; (\neg b \wedge \varphi_2))$ accepts w in the case that $w \models b$. In a similar way $\textsc{cua}((b \wedge \varphi_1) \;\vee\; (\neg b \wedge \varphi_2))$ accepts w in the case $w \models \neg b$.
  For the other direction, assume $\textsc{cua}((b \wedge \varphi_1) \;\vee\; (\neg b \wedge \varphi_2))$ accepts w. Let $s_0q_1q_2\ldots$ be a run of $\textsc{cua}((b \wedge \varphi_1) \;\vee\; (\neg b \wedge \varphi_2))$ over w.
  If $w \models b$ then $\exists q_0 \in Q_0^1$ such that $q_0q_1q_2\ldots$ is a run of $\textsc{cua}(f_1)$ on w. $q_i \notin B^1$ for $i \geq 1$ since $s_0q_1q_2\ldots$ is an accepting run over w and $q_0 \not{i}nB^1$ by Lemma 1. Therefore $q_0q_1q_2\ldots$ is an accepting run of $\textsc{cua}(f_1)$ over w. Since the choice of the run $q_0q_1q_2\ldots$ of $\textsc{cua}(f_1)$ on w was arbitrary it follows that $\textsc{cua}(f_1)$ accepts w. By the induction hypothesis $w \models f_1$. In a similar way if $w \models \neg b$ then $w \models f_2$. Therefore $w \models (b \wedge \varphi_1) \;\vee\; (\neg b \wedge \varphi_2)$.
- $f = [(b \wedge \varphi_1)\; \mathsf{W}\; (\neg b \wedge \varphi_2)]$
  $w \models [(b \wedge \varphi_1)\; \mathsf{W}\; (\neg b \wedge \varphi_2)]$ iff
  $\exists k < |w|$ such that $w^{k..} \models \neg b \wedge f_2$ and $\forall j < k, w^{j..} \models b \wedge f_1$, or
  $w^{m..} \models b \wedge f_1$ for $m \geq 0$.

  - If $\exists k < |w|$ such that $w^{k..} \models \neg b \wedge f_2$ and $\forall j < k, w^{j..} \models b \wedge f_1$ then a run of $\textsc{cua}(f)$ on w is in one of the following forms:
    1. $s_0^jq_1q_2\ldots$ where $0 \leq j < k$ and $\forall 0 \leq j < k$ there exists $q_0 \in Q_0^1$ such that $q_0q_1q_2\ldots$ is a run of $\textsc{cua}(f_1)$ on $w^{j..}$.
    2. $s_0^kq_1q_2\ldots$ and there exists $q_0 \in Q_0^2$ such that $q_0q_1q_2\ldots$ is a run of $\textsc{cua}(f_2)$ on $w^{k..}$.

    For a run of type 1. $\forall 0 \leq j < k, w^{j..} \models b \wedge f_1$, so $\forall 0 \leq j < k, w^{j..} \models f_1$. By the induction hypothesis $\forall 0 \leq j < k, \textsc{cua}(f_1)$ accepts $w^{j..}$. Therefore $q_1q_2\ldots \notin B^1$ implying $q_1, q_2, \ldots \notin B$. By lemma 1 $s_0 \notin B$ so the run is accepting.
    For a run of type 2. $w^{k..} \models \neg b \wedge f_2$ so $w^{k..} \models f_2$. By the induction hypothesis $\textsc{cua}(f_2)$ accepts $w^{k..}$. Therefore $q_1q_2\ldots \notin B^2$ implying $q_1, q_2, \ldots \notin B$. $s_0 \notin B$ so the run is accepting.
  - Otherwise $w^{m..} \models b \wedge f_1$ for $m \geq 0$, and all the runs of $\textsc{cua}(f)$ on w are of the form $s_0^mq_1q_2\ldots$ where $m \geq 0$ and there exist $q_0 \in Q_0^1$ such that $q_0q_1q_2\ldots$ is a run of $\textsc{cua}(f_1)$ on $w^{m..}$. $w^{m..} \models f_1$, by the induction hypothesis $\textsc{cua}(f_1)$ accepts $w^{m..}$ therefore $q_1, q_2, \ldots \notin B^1$ implying $q_1, q_2, \ldots \notin B$. $s_0 \notin B$ therefore $\textsc{cua}(f)$ accpets w. Therefore all runs of $\textsc{cua}(f)$ on w are accepting.

  Other direction: Assume $\textsc{cua}(f)$ accepts w.

- If there exists k such that $w^k \models \neg b$ and $w^j \models b$ for $0 \leq j < k$ then there exist:
  1. a run of $\textsc{cua}(f)$ on w of the form $s_0^k q_1 q_2 \ldots$ where there exists $q_0 \in Q_0^2$ such that $q_0 q_1 q_2 \ldots$ is a run of $\textsc{cua}(f_2)$ on $w^{k..}$.
  2. runs of $\textsc{cua}(f)$ on w of the form $s_0^j q_1 q_2 \ldots$ where $0 \leq j < k$ and there exists $q_0 \in Q_0^1$ such that $q_0 q_1 q_2 \ldots$ is a run of $\textsc{cua}(f_1)$ on $w^{j..}$.

  Since all runs of $\textsc{cua}(f)$ on w are accepting, from 1 it follows there exists k such that $w^j \models b$ for $0 \leq j < k$, $w^k \models \neg b$, and $\textsc{cua}(f_2)$ accepts $w^{k..}$. By the induction hypothesis $w^j \models b$ for $0 \leq j < k$ and $w^{k..} \models \neg b \wedge f_2$. If $k > 0$, from 2 it follows $\textsc{cua}(f_1)$ accepts $w^{j..}$ for $0 \leq j < k$. By the induction hypothesis $w^{j..} \models f_1$ for $0 \leq j < k$. It follows that $w \models (b \wedge f_1)\mathsf{U}(\neg b \wedge f_2)$.
- If there does not exist a k as above, that is $w^j \models b$ for every $b \geq 0$ then all runs of $\textsc{cua}(f)$ on w are of the form $s_0^j q_1 q_2 \ldots$ where $0 \leq j < k$ and there exists $q_0 \in Q_0^1$ such that $q_0 q_1 q_2 \ldots$ is a run of $\textsc{cua}(f_1)$ on $w^{j..}$. Similarly to 2 we get that $w \models \mathsf{G}(b \wedge f_1)$ in this case.

So $w \models ((b \wedge f_1)\mathsf{U}(\neg b \wedge f_2)) \vee \mathsf{G}(b \wedge f_1)$

– $f = r \mapsto f_1$

Let $N = \langle Bool_V Q, Q_0, \delta, F\rangle$ be an $\textsc{nfa}$ accepting $\mathbb{L}(r)$. Let $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}$ be as defined in Section 4.2. First we show that $\mathcal{C}_1$ accepts w iff $w \models r \mapsto \mathit{false}$, then we show that $\mathcal{C}$ accepts w iff $w \models r \mapsto f$.

> $w \models r \mapsto \mathit{false}$ iff $\forall j < |w|,\ w^{0..j} \not\models r$ iff $\forall j < |w|$ all runs of N on $w^{0..j}$ are not accepting iff (by the construction of $\mathcal{C}_1$) $\mathcal{C}_1$ accepts w.

Now we prove the construction for $r \mapsto f_1$:
$w \models r \mapsto f_1$ iff
$\forall j$ such that $w^{0..j} \models r$ it holds that $w^{j..} \models f_1$ iff
$\forall j$ such that $w^{0..j} \not\models r \mapsto \mathit{false}$, $w^{j..} \models f_1$ iff
(by the above claim and by the induction hypothesis) $\forall j$ such that there exists a run $q_0 q_1 \ldots q_{j+1}$ of $\mathcal{C}_1$ on $w^{0..j}$ such that $q_{j+1} \in B^1$, $\mathcal{C}_2 = \textsc{cua}(f_1)$ accepts $w^{j..}$ iff
$\forall j$ such that there exists a run $q_0 q_1 \ldots q_{j+1}$ of $\mathcal{C}_1$ on $w^{0..j}$ where $q_{j+1} \in B^1$, for every run $s_0 s_1 \ldots$ of $\mathcal{C}_2$ on $w^{j..}$, $s_i \notin B^2 = B$ for $i \geq 0$.
Every run of $\mathcal{C}_1$ over w is in one of the following forms:
1. $q_0 q_1 \ldots q_{j+1} s_1 s_2 \ldots$ where $q_0 q_1 \ldots q_{j+1}$ is a run of $\mathcal{C}_1$, over $w^{0..j}$ such that $q_{j+1} \in B^1$ and $s_1 s_2 \ldots$ is a run of $\textsc{cua}(f_1)$ on $w^{j..}$.
2. $q_0 q_1 \ldots$ where $\forall i \geq 0, q_i \in Q^1 \setminus B^1$.

For a run of type 1, it holds that $q_i \notin B, 0 \leq i \leq j+1$ and $s_i \notin B, i \geq 1$, so the run is accepting. For a run of type 2, it holds that $\forall i \geq 0, q_i \in Q^1 \setminus B^1$, $Q^1 \cap B = \emptyset$ therefore the run is accepting. It follows that $\textsc{cua}(r \mapsto f_1)$ accepts w.
Other direction: Assume $\mathcal{C}_1$ accepts w.
If $w^{0..j} \models r$, then $w^{0..j} \not\models r \mapsto \mathit{false}$. By the claim above, there exists a run $q_0 q_1 \ldots q_{j+1}$ of $\mathcal{C}_1$ over $w^{0..j}$ such that $q_{j+1} \in B^1$.
By the construction, for every run $s_0 s_1 \ldots$ of $\mathcal{C}_2 = \textsc{cua}(f_1)$ over $w^{j..}$, $q_0 q_1 \ldots q_{j+1} s_1 s_2 \ldots$

is a run of $\mathcal{C}$ over w, therefore $s_i \notin B = B^2$ for $i \geq 1$. By Lemma 1, $s_0 \notin B^2$ so $s_0 s_1 \ldots$ is an accepting run of $\textsc{cua}(f_1)$ over $w^{j..}$.

Since this holds for every run of $\textsc{cua}(f_1)$ over $w^{j..}$, it follows $\textsc{cua}(f_1)$ accepts $w^{j..}$. By the induction hypothesis, $w^{j..} \models f_1$.

By definition $w \models r \mapsto f_1$.

6 Acknowledgments

We would like to thank Cindy Eisner for interesting discussion about syntactic vs. semantic alphabets. We would like to thank Avigail Orni for her helpful comments on an early draft of this paper.

References

[Acc04] Accellera. Accellera property language reference manual. In *http://www.eda.org/vfv/docs/PSL-v1.1.pdf*, June 2004.

[BBL98] 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.

[BFR04a] S. Ben-David, D. Fisman, and S. Ruah. Automata construction for regular expressions in model checking, June 2004. IBM research report H-0229. http://reswats1.watson.ibm.com/library/cyberdig.nsf/papers/A14E9FE3829B557785256EE6005006A5/I 0229.pdf.

[BFR04b] S. Ben-David, D. Fisman, and S. Ruah. Embedding finite automata within regular expressions. In *Proceeding of the 1st Symposium on Leveraging Applications of Formal Methods*, pages 173–180, 2004.

[BRS99] Roderick Bloem, Kavita Ravi, and Fabio Somenzi. Efficient decision procedures for model checking of linear time logic properties. In *11th International Conference on Computer Aided Verification (CAV'99), LNCS 1633*, pages 222–235, 1999.

[HFE04] John Havlicek, Dana Fisman, and Cindy Eisner. Basic results on the semantics of accellera psl 1.1. Technical Report 2004.02, Accellera, May 2004.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Series in Computer Science. Addison-Wesley, 1979.

[KPR98] Y. Kesten, A. Pnueli, and L. Raviv. Algorithmic verification of linear temporal logic specifications. In K.G. 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.

[KV99] O. Kupferman and M. Y. Vardi. Model checking of safety properties. In *11th International Conference on Computer Aided Verification (CAV'99), LNCS 1633*, pages 172–183, 1999.

[Mai00] Monika Maidl. The common fragment of CTL and LTL. In *IEEE Symposium on Foundations of Computer Science*, pages 643–652, 2000.

[McM93] K. McMillan. Symbolic model checking, 1993.

[VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344. IEEE Computer Society, 1986.

A

A.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 of a DTS is derived from the definition of a fair discrete system (FDS) [KPR98]. A DTS $\mathcal{D} : \langle V, \Theta, \rho, \mathcal{A}\rangle$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 $Bool_V$ the set of all boolean expressions over the state-variables in V (when V is understood from the context we write simply Σ and $Bool$, 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.

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 $Bool_U$ 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 $Bool_U$ iff for every $i \geq 0$, $s_i \Vdash b_i$.

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

A.2 Constructing a DTS from an cua

This section describes step 2 in the outline of the method. Given $\text{CUA}(f) = \langle V, Bool_V, Q, Q_0, \delta, B\rangle$ accepting a formula f we construct $\text{DTS}(f) = \langle V_D, \Theta, \rho, \mathcal{A}\rangle$

accepting bad prefixes of f, denoted $\textsc{dts}(f)$. That is $\textsc{cua}(f)$ accepts a word w iff $\textsc{dts}(f)$ does not accept w.

Let *state* be a new variable (not in V) whose domain is Q. Then

$$V_D = V \cup \{state\}; \quad \Theta = \bigvee_{q_0 \in Q_0} state = q_0; \quad \mathcal{A} = \bigvee_{q \in B} state = q;$$

$$\rho = \bigvee\nolimits_{(q_1,\ell,q_2)\in\delta} (state = q_1 \wedge \ell \wedge state' = q_2)$$

Proposition 1. *$\textsc{cua}(f)$ accepts w iff $\nexists j < |w|$ such that $\textsc{dts}(f)$ accepts $w^{0..j}$.*