From Sequential Extended Regular Expressions to NFA with Symbolic Labels*

Alessandro Cimatti, Sergio Mover, Marco Roveri, and Stefano Tonetta
Fondazione Bruno Kessler - IRST

Abstract. Practical property specification languages such as the IEEE standard PSL use at their core Sequential Extended Regular Expressions (SERE). In order to enable the reuse of traditional verification techniques, it is necessary to translate SEREs into automata. SERE are regular expressions built over alphabets resulting from the state variables of the design under analysis. Thus, a traditional approach to generate the automaton would suffer from the fact that the size of the alphabet is exponential in the number of symbols in the design.
In this work, we tackle this problem by proposing non-deterministic finite automata with symbolic representation of transitions labels, by way of propositional formulas, while states and transitions are explicitly represented. We provide a symbolic version of the algorithms for all the major operations over non-deterministic finite automata. The approach has been implemented in the AUTLIB library, with Binary Decision Diagrams (BDD) used to represent transition labels. We carried out a thorough experimental evaluation over a set of realistic benchmarks, comparing our library against MONA (which uses deterministic finite automata with BDD-based symbolic transitions), and against GRAZ (which features non-deterministic finite automata with a DNF-based representation of the labels). Experimental results over a realistic set of benchmarks show that both features of AUTLIB (the ability to deal with non-determinism, and a BDD-based treatment of labels) are fundamental to achieve acceptable performance.

1 Introduction

Property specification languages (e.g. the IEEE standard PSL [1] and SVA [20]) are increasingly used to represent requirements of hardware and software components. Such languages extend the power of temporal operators [14] by featuring at their core an extended form of regular expressions, called SERE. In order to generalize well-established model checking techniques [9] from traditional temporal logics to such languages, recent approaches [8] require the ability to generate a finite automaton accepting the language of a given SERE.

There exist well-known solutions to the automata construction from regular expression (cfr., e.g., [11, 6, 4, 21]). These approaches are in principle valid in the context of SEREs, which also represent regular languages. However, they are inefficient in the context of SEREs mainly for the following reasons.

* S. Tonetta is supported by the Provincia Autonoma di Trento (project ANACONDA). The other authors are supported by EU grant FP7-2007-IST-1-217069 COCONUT.

First, the alphabet of a SERE over a set of atomic propositions $\mathcal{P}$ is $\Sigma_{\mathcal{P}} = 2^{\mathcal{P}}$, i.e. the powerset of the set of atomic propositions. This means that the size of the alphabet grows exponentially in the number of atomic propositions. In real PSL formulas, the number of atomic propositions can be large, thus inducing a huge alphabet.

Second, SEREs are concise using Boolean formulas over atomic propositions as atomic regular expressions. Boolean formulas represent a set of letters of alphabet, a classical automata construction would force an explicit enumeration of the letters (i.e. the models of a formula), which is exponential.

We tackle this problem by proposing non-deterministic finite automata with a symbolic representation of transitions labels (NFASL). The idea is to represent explicitly states and transitions, and to represent the set of all transitions between two states with just one transition labeled with a Boolean formula. This representation is more concise, since multiple transitions can be collapsed together. Moreover, it avoids an explicit enumerations of the alphabet letters, at the price of applying symbolic transformations when combining transition labels. To this extent, we provide a symbolic version of the algorithms for all the major operations over NFASL, and a procedure to map a SERE into an NFASL. The approach has been implemented in the AUTLIB library, using Binary Decision Diagrams (BDDs) to represent transition labels.

We carried out a thorough experimental evaluation over a set of realistic benchmarks, comparing our AUTLIB library against MONA and GRAZ. MONA is a well known and optimized BDD-based procedure that uses deterministic finite automata with a symbolic representation of the transition relation. GRAZ is a library that features non-deterministic finite automata with a semi-symbolic representation of transition labels, based on Disjunctive Normal Form (DNF). Experimental results over a realistic set of benchmarks show substantial advantages over either competitor, and substantiate the claim that both features of AUTLIB (the ability to deal with non-determinism, and a fully symbolic treatment of labels) are fundamental to achieve acceptable performance.

The paper is structured as follows. In Sec. 2 we present some background on SERE. In Sec. 3 we formalize the notion of NFASLs. In Sec. 4 we compare our approach with related work, and in Sec. 5 we experimentally evaluate the AUTLIB library. In Sec. 6 we draw some conclusions and outline directions for future research.

2 Regular Expressions for Property Specification

PSL is an IEEE-standard language [1] for the specification of hardware requirements, based on a combination of Linear Temporal Logic [14] with SERE, a variant of classic regular expressions [11]. A key difference of SEREs with respect to classic regular expressions is that a letter in the alphabet of a SERE is a truth assignment to a set of atomic propositions, since SEREs are defined over Boolean formulas. In this section we formally define syntax and semantics of SEREs.

Notation We consider regular languages parameterized by a set of atomic propositions $\mathcal{P}$. The alphabet of such languages is given by the set $\Sigma_{\mathcal{P}} = 2^{\mathcal{P}}$ of truth assignments to the propositions of $\mathcal{P}$. We will use $\mathcal{B}_{\mathcal{P}}$ to denote the set of Boolean formulas (obtained applying conjunction $\wedge$, disjunction $\vee$ and negation $\neg$) over the elements of $\mathcal{P}$. We use

ℓ,ℓ', ℓ_1,. . .,ℓ_n to refer to a letter in the alphabet $\Sigma_{\mathcal{P}}$. A finite word is a finite sequence of letters. We use v, w, w_1, w_2 to denote finite words and $\Sigma_{\mathcal{P}}^*$ to denote the set of all words in $\Sigma_{\mathcal{P}}$. Given $v = \ell_0, \ell_1, \ldots, \ell_{n-1} \in \Sigma_{\mathcal{P}}^*$ and $w = \ell'_0, \ell'_1, \ldots, \ell'_{n-1} \in \Sigma_{\mathcal{P}}^*$, $vw = \ell_0, \ell_1, \ldots, \ell_{n-1}, \ell'_0, \ell'_1, \ldots, \ell'_{n-1}$ is the concatenation of words v and w. w^i denotes the $(i+1)^{th}$ letter of w, where the first letter is w^0 while $w^{i\cdot\cdot}$ is the suffix of w starting at w^i.

Syntax and semantics The syntax of SEREs is defined as follows:

Definition 1 (SEREs **syntax).** *An atomic expression is either a Boolean formula $\phi \in \mathcal{B}_{\mathcal{P}}$ or the empty word denoted with ϵ.* SEREs *are obtained by applying recursively the following operators to the atomic expressions:*

- *if r_1, r_2 are* SEREs*, then r_1 **;** r_2, r_1 **:** r_2, r_1 **|** r_2, r_1 **&** r_2 and r_1 **&&** r_2 are* SEREs*;*
- *if r is a* SERE*, then r**[*]** and r**[+]** are* SEREs*.*

Boolean formulas are interpreted over letters in $\Sigma_{\mathcal{P}}$: a propositional atom p is true in ℓ iff $p \in \ell$, false otherwise; Boolean connectives are interpreted in the standard way. If ℓ is a letter and b a Boolean formula, we denote with $\ell \models_{\mathcal{B}} b$ by the fact that ℓ is a model of b.

Definition 2 (SERE **semantics).** *Let w be a finite word over $\Sigma_{\mathcal{P}}$, ϕ a Boolean formula, r, $r1$, $r2$* SEREs*, then the satisfaction relation $w \models r$ is defined as follows:*

- $w \models \epsilon$ iff $|w| = 0$
- $w \models \phi$ iff $|w| = 1$ and $w^0 \models_{\mathcal{B}} \phi$
- $w \models r_1$ **;** r_2 iff $\exists w_1, w_2$ s.t. $w = w_1 w_2$, $w_1 \models r_1$, $w_2 \models r_2$
- $w \models r_1$ **:** r_2 iff $\exists w_1, w_2, \ell$ s.t. $w = w_1 \ell w_2$, $w_1 \ell \models r_1$, $\ell w_2 \models r_2$
- $w \models r_1$ **|** r_2 iff $w \models r_1$ or $w \models r_2$
- $w \models r_1$ **&** r_2 iff *either* $w \models r_1$ and $\exists w_1, w_2$ s.t. $w = w_1 w_2$, $w_1 \models r_2$, or $w \models r_2$ and $\exists w_1, w_2$ s.t. $w = w_1 w_2$, $w_1 \models r_1$
- $w \models r_1$ **&&** r_2 iff $w \models r_1$ and $w \models r_2$
- $w \models r$**[*]** iff $|w| = 0$ or $\exists w_1, w_2$ s.t. $|w_1| \neq 0$, $w = w_1 w_2$, $w_1 \models r$, $w_2 \models r$**[*]**
- $w \models r$**[+]** iff $\exists w_1, w_2$ s.t. $w = w_1 w_2$, $w_1 \models r$, $w_2 \models r$**[*]**

Definition 3 (Language of SEREs**).** *The language of a* SERE *r is the set $\mathcal{L}(r) := \{w \in \Sigma_{\mathcal{P}}{}^* \mid w \models r\}$.*

Example 1. The SERE $\{start$**;**$\{busy\}$**[*];**$end\}$**&&**$\{\{\neg abort\}$**[*]**$\}$ over $\mathcal{P} = \{start, busy, end, abort\}$ may represent the sequences of a potential hardware procedure that lasts for an uncertain number of cycles while never aborted. The SERE $\{req$**;**$\{\{read$**;**$\{\neg cancel\_r \wedge \neg done\}$**[*]**$\}$ **|** $\{write$**;**$\{\neg cancel\_w \wedge \neg done\}$**[*]**$\}\}$**;**$done\}$ over $\mathcal{P} = \{req, read, write, cancel\_r, cancel\_w, done\}$ may represent the sequences of a request of read or write which is accomplished without being canceled.

3 Non-Deterministic Finite Automata with Symbolic Labels

3.1 Symbolic Representation

Definition 4 (NFASL). *A Non-deterministic Finite-state Automaton with Symbolic Labels (NFASL) is a tuple $A = \langle \mathcal{P}, Q, q^0, \rho, F \rangle$, where*

- *$\mathcal{P}$ is the set of atomic propositions,*
- *Q is a finite set of states,*
- *$q^0 \in Q$ is the initial state,*
- *$\rho : Q \to 2^{\mathcal{B}_\mathcal{P} \times Q}$ is the symbolic transition function,*
- *$F \subseteq Q$ is the set of final states.*

By definition, an NFASL can move from a state q non-deterministically choosing a pair of label-state. The definition of transition function differs from the classic one, where a move from state q is determined by a single letter of the alphabet, namely $\rho_C : Q \times \Sigma_\mathcal{P} \to 2^Q$. Given, ρ, we can define ρ_C as $\rho_C(q, l) := \{q' \mid \langle \phi, q' \rangle \in \rho(q)$ for some ϕ s.t. $l \models \phi\}$ A tuple $\langle q, \phi, q' \rangle$, where $\langle \phi, q' \rangle \in \rho(q)$, is called a symbolic transition. A symbolic transition $\langle q, \phi, q' \rangle$ is said feasible iff ϕ is satisfiable.

Definition 5 (NFASL language). *An NFASL $A = \langle \mathcal{P}, Q, q^0, \rho, F \rangle$ accepts a word $l_1, \ldots, l_n \in \Sigma^*_\mathcal{P}$ iff there exists a sequence of states $\pi = q_0, q_1, \ldots, q_n$ such that $q_0 = q^0$, $q_n \in F$, and, for all i, $1 \leq i \leq n$, there exists $\phi_i \in \mathcal{B}_\mathcal{P}$ such that $\langle \phi_i, q_i \rangle \in \rho(q_{i-1})$ and $l_i \models \phi_i$. The set $L(A) \subseteq \Sigma^*_\mathcal{P}$ of words accepted by A is called language of A.*

Definition 6 (NFASL path). *A path of an NFASL $A = \langle \mathcal{P}, Q, q^0, \rho, F \rangle$ is a sequence of states $\pi = q_0, q_1, \ldots, q_n$ such that $q_0 = q^0$ and, for all i, $1 \leq i \leq n$, there exists $\phi \in \mathcal{B}_\mathcal{P}$ such that $\langle \phi, q_i \rangle \in \rho(q_{i-1})$. π is accepting iff $q_n \in F$.*

We say that a state q is reachable in A iff there exists a path $\pi = q_0, q_1, \ldots, q_n = q$ of A. An automaton is *trim* iff all its states are contained in some accepting path. Finally, an automaton is said to be *normalized* iff for each pair of states q_0, q_1 there exists at most one transition from q_0 to q_1.

3.2 From SERE to NFASL

In the following, we explain the procedure to convert a SERE into the corresponding NFASL. The translation is recursively based on the structure of the SERE and can be seen as a variant of the Berry-Sethi construction (cfr. [21]). The base cases are the automata to recognize ϵ and the atomic expressions. The step cases map directly on the corresponding operations on automata.

Let us consider the NFASLs $A_1 = \langle \mathcal{P}_1, Q_1, q^0_1, \rho_1, F_1 \rangle$ and $A_2 = \langle \mathcal{P}_2, Q_2, q^0_2, \rho_2, F_2 \rangle$.

Definition 7 (Intersection). *The intersection $A_1 \cap A_2$ of A_1 and A_2 is the NFASL $A = \langle \mathcal{P}_1 \cup \mathcal{P}_2, Q_1 \times Q_2, q^0_1 \times q^0_2, \rho, F_1 \times F_2 \rangle$, where*
$\rho(q_1 \times q_2) = \{\langle \phi_1 \wedge \phi_2, q'_1 \times q'_2 \rangle \mid \langle \phi_1, q'_1 \rangle \in \rho_1(q_1), \langle \phi_2, q'_2 \rangle \in \rho_2(q_2)\}$.

Definition 8 (Union). *The union $A_1 \cup A_2$ of A_1 and A_2 is the NFASL $A = \langle \mathcal{P}_1 \cup \mathcal{P}_2, \{q^0\} \cup Q_1 \cup Q_2, q^0, \rho, F\rangle$, where q^0 is a new state , $F = q^0 \cup F_1 \cup F_2$ if $q_1^0 \in F_1$ or $q_2^0 \in F_2$ else $F = F_1 \cup F_2$, and*

$$\rho(q) = \begin{cases} \rho_1(q_1^0) \cup \rho_2(q_2^0) & \text{if } q = q^0 \\ \rho_1(q) & \text{if } q \in Q_1 \\ \rho_2(q) & \text{if } q \in Q_2 \end{cases}$$

Definition 9 (Concatenation). *The concatenation $A_1; A_2$ of A_1 and A_2 is the NFASL $A = \langle \mathcal{P}_1 \cup \mathcal{P}_2, Q_1 \cup Q_2, q_1^0, \rho, F\rangle$, where $F = F_2$ if $q_2^0 \notin F_2$ else $F = F_1 \cup F_2$, and*

$$\rho(q) = \begin{cases} \rho_1(q) \cup \rho_2(q_2^0) & \text{if } q \in F_1 \\ \rho_1(q) & \text{if } q \in Q_1 \setminus F_1 \\ \rho_2(q) & \text{if } q \in Q_2 \end{cases}$$

Definition 10 (Fusion). *The fusion $A_1 : A_2$ of A_1 and A_2 is the NFASL $A = \langle \mathcal{P}_1 \cup \mathcal{P}_2, Q = Q_1 \cup Q_2, q_1^0, \rho, F_2\rangle$, where*

$$\rho(q) = \begin{cases} \rho_1(q) \cup \{\langle \phi_1 \wedge \phi_2, q_2'\rangle \mid \langle \phi_1, q_1'\rangle \in \rho_1(q_1), q_1' \in F_1, \langle \phi_2, q_2'\rangle \in \rho_2(q_2^0)\} & \text{if } q \in Q_1 \\ \rho_2(q) & \text{if } q \in Q_2 \end{cases}$$

Definition 11 (Star). *The star A_1* of A_1 is the NFASL $A = \langle \mathcal{P}_1, Q_1, q_1^0, \rho, F_1 \cup \{q^0\}\rangle$, where*

$$\rho(q) = \begin{cases} \rho_1(q) \cup \rho_1(q_1^0) & \text{if } q \in F_1 \\ \rho_1(q) & \text{if } q \in Q_1 \setminus F_1 \end{cases}$$

Definition 12 (Plus). *The plus A_1+ of A_1 is the NFASL $A = \langle \mathcal{P}_1, Q_1, q_1^0, \rho, F_1\rangle$, where*

$$\rho(q) = \begin{cases} \rho_1(q) \cup \rho_1(q_1^0) & \text{if } q \in F_1 \\ \rho_1(q) & \text{if } q \in Q_1 \setminus F_1 \end{cases}$$

The conversion algorithm is detailed in Figure 1. The function $construct$ builds the automaton starting from the initial state and adding only feasible transitions and reachable states. Moreover, it merges the transitions connecting the same pair of states producing a normalized automaton (see [15] for further details). Finally, the function $trim$ is used to remove states which do not reach the accepting states due to inconsistent combination of labels (see Definition of $A_1 \cap A_2$ and $A_1 : A_2$).

Theorem 1. *If $A = SERE2NFASL(r)$, then $L(A) = L(r)$.*

The construction matches the complexity of the standard algorithms, in that it builds an automaton with a linear number of states if the SERE does not contain any intersection operator, while it is in general exponential.

3.3 Determinization

Definition 13 (Determinization). *Given an NFASL A_1, the determinization A_1^D of A_1 is the NFASL $A = \langle \mathcal{P}, Q, q^0, \rho, F\rangle$, where $\mathcal{P} = \mathcal{P}_1$, $Q = 2^{Q_1}$, $q^0 = \{q_1^0\}$ and*

- $\rho(q) = \{\langle \phi, q'\rangle \mid \phi = \bigwedge_{\langle \phi_1, q_1'\rangle \in S} \phi_1 \wedge \bigwedge_{\langle \phi_1, q_1'\rangle \in S'} \neg\phi_1, q' = \{q_1'\}_{\langle \phi_1, q_1'\rangle \in S}$, *for some* $S \subseteq \bigcup_{q_1 \in q} \rho_1(q_1)$ *and* $S' = \bigcup_{q_1 \in q} \rho_1(q_1) \setminus S\}$;
- $F = \{q \mid q \cap F_1 \neq \emptyset\}$.

Theorem 2. *A_1^D is deterministic and $L(A_1^D) = L(A_1)$*

Algorithm *SERE2NFASL*(r)

1. **switch** r
2. **case** ϵ: $A = \langle \mathcal{P}, \{q\}, q, \rho, \{q\} \rangle$ with $\rho(q) = \emptyset$
3. **case** ϕ: $A = \langle \mathcal{P}, \{q_0, q_1\}, q_0, \rho, \{q_1\} \rangle$ with $\rho(q_0) = \{\langle \phi, q_1 \rangle\}$ and $\rho(q_1) = \emptyset$
4. **case** r_1 **;** r_2
5. $A_1 =$ *SERE2NFASL*(r_1); $A_2 =$ *SERE2NFASL*(r_2)
6. $A = construct(A_1; A_2)$
7. **case** r_1 **:** r_2
8. $A_1 =$ *SERE2NFASL*(r_1); $A_2 =$ *SERE2NFASL*(r_2)
9. $A = trim(construct(A_1 : A_2))$
10. **case** r_1 **|** r_2
11. $A_1 =$ *SERE2NFASL*(r_1); $A_2 =$ *SERE2NFASL*(r_2)
12. $A = construct(A_1 \cup A_2)$
13. **case** r_1 **&&** r_2
14. $A_1 =$ *SERE2NFASL*(r_1); $A_2 =$ *SERE2NFASL*(r_2)
15. $A = trim(construct(A_1 \cap A_2))$
16. **case** r_1 **&** r_2
17. $A =$ *SERE2NFASL*$(\{r_1$**&&**$\{r_2$**;**$\{\top\}$**[*]**$\}\}$ **|** $\{\{r_1$**;**$\{\top\}$**[*]**$\}$**&&**$r_2\})$
18. **case** r_1**[*]**
19. $A_1 =$ *SERE2NFASL*(r_1)
20. $A = construct(A_1*)$
21. **case** r_1**[+]**
22. $A_1 =$ *SERE2NFASL*(r_1)
23. $A = construct(A_1+)$
24. **return** A

Fig. 1. Algorithm to compile a SERE into a NFASL.

3.4 NFASL reduction based on bi-simulation

We use the quotient graph with regard to the bi-simulation relation to reduce the state space of the NFASL. This is a standard technique (cfr., e.g., [12]), but we adapt the definition of simulation to the symbolic labels.

Given an NFASL $A = \langle \mathcal{P}, Q, q^0, \rho, F \rangle$, the bi-simulation relation $\equiv_R$ is defined as the coarsest equivalence relation over Q that satisfies:

$P1$) $\equiv_R \cap (F \times (Q \setminus F)) = \emptyset$,

$P2_C$) for any $p, q \in Q$, $a \in \Sigma_{\mathcal{P}}$, if $p \equiv_R q$ then for all $q' \in \rho_C(q, a)$, there exists $p' \in \rho(p, a)$ such that $q' \equiv_R p'$.

We define an equivalent condition:

$P2$) for any $p, q \in Q$, if $p \equiv_R q$ then for all $\langle \phi, q' \rangle \in \rho(q)$, then $\phi \rightarrow \bigvee_{\langle \phi', p' \rangle \in \rho(p), q' \equiv_R p'} \phi'$.

Theorem 3. *$P2_C$ is equivalent to $P2$.*

4 Related Work

Several works focus on the construction of automata from regular expressions. Most of them (e.g. [13, 17, 5]) use classic automata representations, where states and labels

are represented explicitly. Other implementations (e.g., AUTOMATA[1] and LIBFA[2]) have a partially-symbolic representation of labels (e.g., intervals of letters). However, these approaches are inefficient when considering Boolean formulas as atomic expressions.

Symbolic representations of finite state automata have been investigated in several works. As in our approach, MONA [10] uses an explicit representation for states, but a symbolic representation for the entire transition relation. The symbolic representation is achieved using a variant of BDDs, called shared multi-terminal BDDs (SMBDDs). Roots and leaves in a SMBDD are states of the automaton, while the internal nodes are atomic propositions. A transition is represented with a path from a root to a leaf. Unlike our approach, MONA cannot represent NFAs: given a state and a letter, there is a unique leaf node. Moreover, MONA does not implement regular languages operations such as concatenation and Kleene closure. STRANGER [22] is a library developed in the context of static strings analysis for Web applications. It is implemented on top of MONA, and extends it with more operations on automata.

Other approaches represent automata states and transitions explicitly and use a symbolic representation of labels, as in our case. The GRAZ library [16] represents NFAs where transitions are labelled with DNF formulas. Pairs of labels are combined by multiplying all the disjuncts of the first label with the disjuncts of the second label. This library was previously used in the NUSMV [7] model checker to manage the construction of automata from SEREs [8]. Also in FSA [18] a predicate is used as label for a transition. The library uses Prolog and not BDDs to represent a predicate and to check its satisfiability. FSA describes the algorithms that we use to perform intersection and determinization. In [2] the authors give an efficient implementation of minimization for NFAs with large alphabets. The representation of NFA is explicit for states, and symbolic (BDD-based) for labels. This work does not take into account the construction of automata from regular expressions. Also REX [19] uses an approach similar in spirit to ours, where states are explicit and labels are symbolic. The key difference is that reasoning on symbolic labels is done using a Satisfiability Modulo Theory (SMT) solver instead of BDDs. As ours, the approach aims at dealing with extended regular expressions. However not all SEREs operations are covered (e.g. the fusion operator is missing).

In the context of circuit synthesis for PSL monitoring, a construction of the automata from SEREs is described in [3]. The approach is very similar to the one presented here, using the same approach of handling symbolic labels on automata transitions. However, the automata and the translation are not formally presented. Unfeasible transitions are not removed, and no detail is given on how formulas are manipulated (BDD, DNF, or strings). The goal of the approach indeed is not the automata construction, but the generation of the circuit and the evaluation regards only the final hardware circuits. In particular, there is no comparison with standard libraries for automata manipulation.

[1] `http://www.brics.dk/automaton/`

[2] `http://augeas.net/libfa/`

5 Experimental Evaluation

The approach decribed in previous sections has been implemented in the AUTLIB library. The library is written in C, using adjacency lists to represent transitions outgoing from a state, and BDDs from the CUDD package (`http://vlsi.colorado.edu/~fabio`) for transition labels. The architecture is extensible to other forms of Boolean reasoning, such as propositional satisfiability (SAT), and to SMT. AUTLIB is used at the core of an extension of the NUSMV [7] model checker able to deal with the PSL language, and, as explained in [8], it is used to generate the automata necessary for PSL verification.

Set up The proposed algorithms are evaluated in terms of *construction time* of an automaton corresponding to a SERE, and *number of states* of the resulting automaton. The AUTLIB library was evaluated in two modes, with and without reduction. In the first mode, the activation of reduction is controlled by a simple heuristic, namely reduction is run only after **|**, **&&** and **&** operators. For the comparison, we use a test suite of 1200 SEREs, obtained by randomly modifying patterns extracted from industrial case studies. The SEREs are combinations of concatenations where the top level operators are randomly chosen in $\{|, \&\&, \&, [*], [+]\}$. The concatenations combine atomic Boolean expressions, or repetition of Boolean expressions using **[*]** or **[+]**. The number of concatenated SEREs is randomly chosen in the range $[2, 10]$. We generated 12 different families of benchmarks, choosing a possible configuration of parameters. The parameters are the number of top-level operators (which ranges in $\{1, 2\}$), the depth of Boolean expressions (which ranges in $\{2, 3\}$) and the number of atomic propositions (which ranges in $\{8, 10, 15\}$). For each family we generated 100 random SEREs.

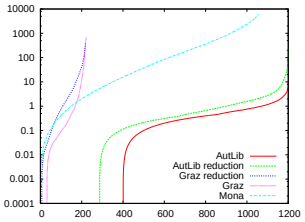
AUTLIB is compared against the GRAZ library [16] and MONA [10]. Also for GRAZ we considered two operating modes: with and without NFA reduction. We compared with MONA through the STRANGER library [22], that provides Concatenation and Star as additional functions, and minimizes the DFA after such operations.

We ran the experiments on a Linux machine equipped with a 2.66GHz Intel(R) Core(TM)2 Quad Core, and 4GB of RAM with a time out of 120 seconds and memory limit of 3Gb. All results, together with the binaries and test cases necessary to reproduce them, are available at `http://es.fbk.eu/people/mover/tests/CIAA10/`.

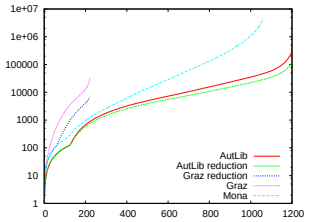
Results The results are presented in two different forms. Survival plots are used to provide a global view of the results: for each competitor, the "snake" shows the cumulative time required to solve a fixed number of instances. Pairwise comparison is obtained by means of scatter plots, where the x and y coordinates for each point represent the performance of the compared solvers on a given sample.

Figure 2(a) shows the cumulative plot for automata construction times for all the evaluated libraries. The GRAZ library, with and without reduction, shows poor performances, solving about 200 over 1200 examples. MONA can solve about 1050 examples while AUTLIB solves all the random generated SEREs. AUTLIB, with and without minimization, is much faster than MONA, and almost immediate on some instances.

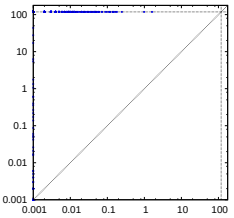
AUTLIB and GRAZ can be compared from the scatter plots shown in Figure 2(c) and 2(d). All the examples where GRAZ can construct the automaton before timeout are



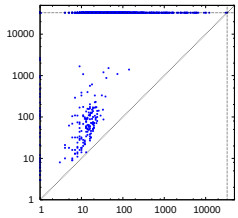
(a) Construction time (sec.)



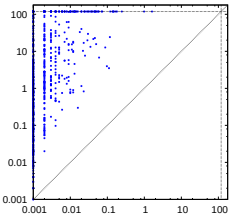
(b) Number of states



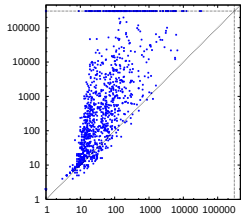
(c) Construction time (sec.): AUTLIB (X axes) vs GRAZ (Y axes)



(d) # of states: AUTLIB (X axes) vs GRAZ (Y axes)



(e) Construction time (sec.): AUTLIB (X axes) vs MONA (Y axes)



(f) # of states: AUTLIB (X axes) vs MONA (Y axes)

trivial for AUTLIB. The bottleneck in the GRAZ approach is due to operations on labels performed on the DNF structure of formulas. Comparing the number of states, GRAZ generates bigger automata than AUTLIB. This is due to the management of labels in GRAZ, where for performance reasons the satisfiability check of a conjunction of labels is not complete. Transitions with inconsistent labels are thus created, possibly avoiding the pruning of unreachable states.

Figure 2(e) shows the scatter plot that compares construction times for MONA, on x axes, and AUTLIB, on y axes. AUTLIB outperforms MONA on every example. These results can be explained looking at Figure 2(f), that shows the number of states for the constructed automata. It is not surprising that DFAs generated by MONA are much bigger than NFASL of AUTLIB, since non-deterministic automata have a much succinct representation, which better adapts to common SERE expressions.

We also tested the effect of reductions for AUTLIB and GRAZ. For AUTLIB the benefits deriving from reductions, i.e., a reduced number of states, seem to be modest with respect to an increased construction time. As for GRAZ, although the reduction is a costly operation in terms of time, the benefits in the number of states are more evident. For lack of space, the scatter plots of the reductions are reported in appendix.

6 Conclusions and Future Work

In this paper we have addressed the problem of providing automata-based techniques suitable for the manipulation of regular expressions arising in specification languages such as PSL. We propose an approach where non-deterministic finite automata are equipped with fully symbolic labels, represented by means of BDDs, over a given set of variables. We implemented an efficient library where all the standard functionalities are provided. The experiments demonstrate the need for the compactness of non-deterministic finite automata (compared to approaches based on deterministic finite automata), and the efficiency of a fully symbolic approach to label representation (with respect to an approach based on sets of partial assignments).

In the future, we plan to extend the experimentation with additional benchmarks, and to pinpoint possible bottlenecks of the current implementation. We will also investigate the use of alternative symbolic technique (e.g. SAT and SMT solvers), and will develop fully symbolic minimization procedures.

Acknowledgments We thank I. Pill for support with the GRAZ library, and O. Ibarra, T. Bultan, F. Yu and M. Alkhalaf for providing us with the STRANGER library.

References

1. IEEE Standard for Property Specification Language (PSL). *IEEE Std 1850-2005*, 2005.
2. P. Aziz Abdulla, J. Deneux, L. Kaati, and M. Nilsson. Minimization of non-deterministic automata with large alphabets. In *CIAA*, pages 31–42, 2005.
3. M. Boule and Z. Zilic. Efficient Automata-Based Assertion-Checker Synthesis of SEREs for Hardware Emulation. In *ASP-DAC*, pages 324–329, 2007.
4. J.-M. Champarnaud. Evaluation of Three Implicit Structures to Implement Nondeterministic Automata From Regular Expressions. *Int. J. Found. Comput. Sci.*, 13(1):99–113, 2002.
5. J. M. Champarnaud and G. Hansel. Automate, a computing package for automata and finite semigroups. *J. Symb. Comput.*, 12(2):197–220, 1991.
6. J.-M. Champarnaud, J.-L. Ponty, and D. Ziadi. From regular expressions to finite automata. *International Journal of Computer Mathematics*, 72(4):415–431, 1999.
7. A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: A New Symbolic Model Checker. *STTT*, 2(4):410–425, 2000.
8. A. Cimatti, M. Roveri, and S. Tonetta. Symbolic Compilation of PSL. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1737–1750, (2008).
9. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, 1999.
10. J.G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. 1995.
11. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
12. L. Ilie, G. Navarro, and S. Yu. On NFA Reductions. In *Theory Is Forever*. pp 112-124, 2004.
13. V. Kell, A. Maier, A. Potthoff, W. Thomas, and U. Wermuth. Amore: a system for computing automata, monoids and regular expressions. In *STACS 89*, pages 537–538. Springer, 1989.
14. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems, Specification*. Springer Verlag, New York, 1992.
15. S. Mover. A Robust Automata Library for Symbolic Model Checking Expressive Temporal Logics, November 2009.

16. I. Pill. *Requirements Engineering and Efficient Verification of PSL properties*. PhD thesis, Graz Univeristy of Technology, 2008.
17. D. Raymond and D. Wood. Grail: a c++ library for automata and expressions. *J. Symb. Comput.*, 17(4):341–350, 1994.
18. G. van Noord and D. Gerdemann. Finite State Transducers with Predicates and Identities. *Grammars*, 4(3):263–286, 2001.
19. M. Veanes, P. Grigorenko, P. de Halleux, and N. Tillmann. Rex: Symbolic regular expression explorer. In *ICST*, 2010.
20. S. Vijayaraghavan and M. Ramanathan. *A Practical Guide for SystemVerilog Assertions*. Springer, 2005.
21. B. W. Watson. A taxonomy of finite automata construction algorithms. Technical report, Eindohoven University of Technology – Mathematics and Computing Science, 1994.
22. F. Yu, T. Bultan, M. Cova, and O. H. Ibarra. Symbolic string verification: An automata-based approach. In *SPIN '08*, pages 306–324. Springer, 2008.

A Proofs

The correctness of Theorem 1 can be directly deduced by the following Lemma.

Lemma 1.

- $L(A_1 \cap A_2) = L(A_1) \cap L(A_2)$
- $L(A_1 \cup A_2) = L(A_1) \cup L(A_2)$
- $L(A_1; A_2) = L(A_1); L(A_2)$
- $L(A_1 : A_2) = L(A_1) : L(A_2)$
- $L(A_1*) = L(A_1)*$
- $L(A_1+) = L(A_1)+$

where we used the following notation on regular languages:

- $L_1; L_2 = \{w \mid \exists w_1, w_2 \text{ s.t. } w = w_1 w_2, w_1 \in L_1, w_2 \in L_2\}$
- $L_1 : L_2 = \{w \mid \exists w_1, w_2, \ell \text{ s.t. } w = w_1 \ell w_2,\ w_1\ell \in L_1,\ \ell w_2 \in L_2\}$
- $L* = \{w \mid |w| = 0 \text{ or } \exists w_1, w_2, \ldots, w_n \text{ s.t. } w = w_1 w_2 \ldots w_n, \text{ for all } i, 1 \leq i \leq n, w_i \in L\}$
- $L+ = \{w \mid \exists w_1, w_2, \ldots, w_n \text{ s.t. } w = w_1 w_2 \ldots w_n, \text{ for all } i, 1 \leq i \leq n, w_i \in L\}$

The proof of the Lemma is quite straightforward and we detail the intersection case (the other cases are similar).

Proof ($L(A_1 \cap A_2) = L(A_1) \cap L(A_2)$). From the definition of $A_1 \cap A_2$, $w = l_1, \ldots, l_n \in L(A_1 \cap A_2)$ iff there exists a sequence of states $\pi = q_{0,1} \times q_{0,2}, q_{1,1} \times q_{1,2}, \ldots, q_{n,1} \times q_{n,2}$ such that $q_{0,1} = q_1^0$, $q_{0,2} = q_2^0$, $q_{n,1} \in F_1$, $q_{n,2} \in F_2$ and, for all i, $1 \leq i \leq n$, there exist $\phi_{i,1} \in \mathcal{B}_{\mathcal{P}_1}$ and $\phi_{i,2} \in \mathcal{B}_{\mathcal{P}_2}$ such that $\langle \phi_{i,1}, q_{i,1} \rangle \in \rho_1(q_{i-1,1})$, $\langle \phi_{i,2}, q_{i,2} \rangle \in \rho_2(q_{i-1,2})$, $l_i \models \phi_{i,1}$, and $l_i \models \phi_{i,2}$. This is true iff $w \in L(A_1)$ and $w \in L(A_2)$.

With regard to Theorem 2, the proof of the language equivalence is quite straightforward. Instead we detail the proof of the determinism.

Proof (A^D is deterministic). Given a state q of A^D, q corresponds to a set X of states of A. Consider two transitions $\langle q, \phi_1, q_1 \rangle$ and $\langle q, \phi_2, q_2 \rangle$ outgoing from the same state q. Then, $\phi_1 = \bigwedge_{\langle \phi'_1, q'_1 \rangle \in S} \phi'_1 \wedge \bigwedge_{\langle \phi'_1, q'_1 \rangle \in S'} \neg \phi'_1$, $q_1 = \{q'_1\}_{\langle \phi'_1, q'_1 \rangle \in S_1}$, for some $S_1 \subseteq \bigcup_{q' \in X} \rho(q')\}$ and $S'_1 = \bigcup_{q' \in q} \rho(q') \backslash S_1\}$ and $\phi_2 = \bigwedge_{\langle \phi'_2, q'_2 \rangle \in S} \phi'_2 \wedge \bigwedge_{\langle \phi'_2, q'_2 \rangle \in S'} \neg \phi'_2$, $q_2 = \{q'_2\}_{\langle \phi'_2, q'_2 \rangle \in S_2}$, for some $S_2 \subseteq \bigcup_{q' \in X} \rho(q')\}$ and $S'_2 = \bigcup_{q' \in q} \rho(q') \setminus S_2\}$. If q_1 and q_2 are different, then also S_1 and S_2 must be different. Thus, there exists $\langle \phi_d, q_d \rangle$ belonging to one set and not to the other. For any letter l either $l \models \phi_d$ or $l \models \neg \phi_d$. Thus, l cannot be accepted by both $\langle q, \phi_1, q_1 \rangle$ and $\langle q, \phi_2, q_2 \rangle$.

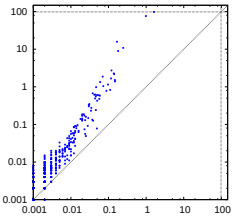
Finally, here is the proof of Theorem 3.

Proof. Let us prove that $P2_C \Rightarrow P2$.

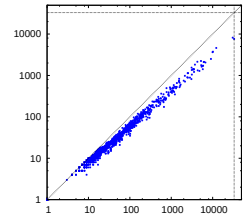
Suppose $p \equiv_R q$. Let us consider $\langle \phi, q' \rangle \in \rho(q)$ and $a \in \Sigma$ such that $a \models \phi$. Then $q' \in \rho_C(q, a)$ and by hypothesis there exists $p' \in \rho(p, a)$ such that $q' \equiv_R p'$. Thus there exists ϕ' such that $a \models \phi'$ and $\langle \phi', q' \rangle \in \rho(p')$. Thus $a \models \bigvee_{\langle \phi', p' \rangle \in \rho(p), q' \equiv_R p'} \phi'$.

The other direction is similar.

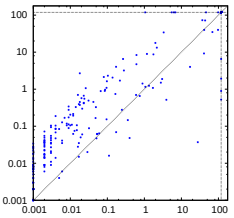
B Reduction plots



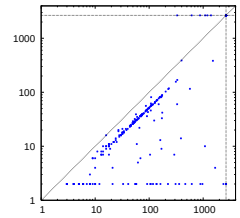
(g) Construction time (sec.): AUTLIB (X axes) vs AUTLIB with reduction (Y axes)



(h) # of states: AUTLIB (X axes) vs AUTLIB with reduction (Y axes)



(i) Construction time (sec.): GRAZ (X axes) vs GRAZ with reduction (Y axes)



(j) # of states: GRAZ (X axes) vs GRAZ with reduction (Y axes)