Regular Model Checking made Simple and Efficient

Parosh Aziz Abdulla[1], Bengt Jonsson[1], Marcus Nilsson[1], and Julien d'Orso[1]

Dept. of Computer Systems, P.O. Box 337, S-751 05 Uppsala, Sweden
{parosh,bengt,marcusn,juldor}@docs.uu.se

Abstract. We present a new technique for computing the transitive closure of a regular relation characterized by a finite-state transducer. The construction starts from the original transducer, and repeatedly adds new transitions which are compositions of currently existing transitions. Furthermore, we define an equivalence relation which we use to merge states of the transducer during the construction. The equivalence relation can be determined by a simple local check, since it is syntactically characterized in terms of "columns" that label constructed states. This makes our algorithm both simpler to present and more efficient to implement, compared to existing approaches. We have implemented a prototype and carried out verification of a number of parameterized protocols.

1 Introduction

Regular model checking has been proposed as a uniform paradigm for algorithmic verification of several classes of infinite-state systems; in particular *parameterized systems* [KMM+97,ABJN99,BJNT00,PS00]. Such systems arise naturally in many applications. For instance, the specification of a protocol may be parameterized by the number of components which may participate in a given session of the protocol. In such a case, it is interesting to verify the correctness of the protocol, regardless of the number of participants in a particular session. The idea of regular model checking is to perform symbolic reachability analysis, using words over a finite alphabet to represent states, and using finite-state transducers to describe transitions between states. Such an approach has been advocated by, e.g., Kesten et al. [KMM+97], Boigelot and Wolper [WB98], and implemented, e.g., in the Mona [HJJ+96], MoSel [KMMG97], or LASH [BFL] packages.

A generic task in most symbolic model checking paradigms is to compute a representation for the transitive closure of the transition relation. Such a characterization can then be used to compute the set of reachable states (e.g. for verifying safety properties), or to find loops when verifying liveness properties [BJNT00,PS00].

A central problem in regular model checking is that the standard iteration-based methods for computing transitive closures, which are used for finite-state systems (e.g., [BCMD92]), are guaranteed to terminate only if there is a bound on the distance (in number of transitions) from the initial configurations to any reachable configuration. In general, a parameterized or infinite-state system does not have such a bound. For instance, consider a transition of a parameterized system in which a process passes a token to its neighbour. The transitive closure of such a transition relation will be to pass the token to any other process through an arbitrary sequence of neighbours. Therefore, an important challenge in the design of algorithms for computing transitive closures, is to invent techniques in order to enhance the performances of iteration-based methods. One such a technique is that of *accelerations*: try to calculate the effect of arbitrarily long sequences of transitions. Although such an effect is in general not computable, accelerations have successfully been applied for several classes of parameterized and infinite-state systems, e.g., systems with unbounded FIFO channels [BG96,BGWW97,BH97,ABJ98], systems with stacks [BEM97,Cau92,FWW97,ES01] systems with counters [BW94,CJ98], and several classes of parameterized systems [ABJN99,PS00].

In our work [JN00], we gave an explicit representation of a finite-state transducer accepting the transitive closure, for the case that the transition relation satisfies a condition of *bounded local depth*. A related automata-based construction was presented in [BJNT00]. Both these works employ a direct construction of some form of "column transducer", whose states are sequences (columns) of states of the original transducer.

In this paper, we present a technique for computing transitive closures, which is more light-weight than our previous automata-based solutions. The technique uses straight-forward post-image computation augmented with identification of "equivalent" states. Roughly, the construction of transitive closure proceeds by starting from the original transducer, then repeatedly adding new transitions by simple matching of already constructed transitions. During the construction, equivalent states are merged, using an equivalence relation which preserves the set of traces of the transducer. More precisely, our equivalence relation is the combination of a forward simulation and a backward simulation relation. This makes sure that no prefix/suffix combinations are added to the set of traces. The technique represents a substantial simplification over the previous approaches [JN00,BJNT00], where several layers of automata-theoretic constructions were used. An important property of the equivalence relation is that it can be syntactically characterized in terms of "columns" that label constructed states, and therefore it can be determined by a simple local check. This allows for a much more efficient implementation of the algorithm. In fact, a first implementation of the new, simplified, technique improves the running times of examples by up to a factor of ten. At the same time, the technique does not substantially sacrifice completeness. Completeness results, similar to those in [JN00,BJNT00] can be proven.

Related Work Previous work on the general aspects of regular model checking, and on analyzing classes of systems, e.g., pushdown systems, parameterized systems, systems with FIFO channels, or with counters, has already been mentioned earlier in this introduction.

In [BJNT00], we present a technique for computing the transitive closure of a regular transducer. The technique relies on several potentially expensive operations on automata such as checking language equivalence, computing post-images of regular sets, and saturating regular sets with respect to members of the alphabet. These operations are not needed in the present algorithm, leading to a much more efficient implementation (see Section 5).

Dams et al. [DLS01] present a related approach, which differs from ours in the way states are merged. Dams et al. use an extensional equivalence, which is computed by a global analysis of the current approximation of the transitive closure. It appears that this calculation is very expensive, and the paper does not report successful application of the techniques to examples of similar complexity as the more complex examples in Section 5. In contrast, we base the equivalence on a relation defined in terms of the "columns" that label constructed states, which can be determined by a simple local check. The technique in our proof of Theorem 2 is inspired by the proof technique of their paper.

Caucal [Cau00] presents a class of rewriting systems, called *right-overlapping systems* (and symmetrically also *left-overlapping systems*) for which the transitive closure can be computed as a transducer. A simple instance is the token-passing example mentioned at the beginning of this introduction. Our algorithm is guaranteed to terminate on all overlapping rewriting systems.

Touili [Tou01] presents a technique for computing transitive closures of regular transducers based on *widening*, and shows that the method is sufficiently powerful to simulate earlier constructions described in [ABJN99] and [BMT01]. However, these are substantially weaker than for the automata-based techniques. Another approach, based on second order monadic logic [PS00], covers some commonly occurring patterns of successive transduction.

Outline In the next section, we present a simple example which we will use to illustrate our algorithm. In Section 3 we describe the algorithm for computing transitive closures. In Section 4, we show soundness and completeness of the algorithm and present sufficient conditions for termination. Section 5 contains a description of an implementation of the algorithm and the result of applying it to a number of mutual exclusion protocols. Concluding remarks and directions for future research are given in Section 6.

2 An Example

In this section, we present informally, through a simple example, an algorithm which computes R^+, for a regular relation R. It computes successively larger under-approximations of R^+, starting from R. The algorithm consists of repeatedly performing a small basic step which combines two matching transitions of the current approximation. It also uses an equivalence relation on states, for on-the-fly identification of newly produced states.

Let Σ be a finite alphabet of symbols. Let R be a regular relation on Σ, represented by a deterministic finite-state *transducer* $T = \langle Q, q_0, \longrightarrow, F \rangle$ where

Q is the set of states, q_0 is the initial state, $\longrightarrow$: $(Q \times (\Sigma \times \Sigma)) \mapsto Q$ is the transition function, and $F \subseteq Q$ is the set of accepting states. We use $q_1 \stackrel{(a,b)}{\longrightarrow} q_2$ to denote that $(q_1, (a, b), q_2) \in \longrightarrow$. We use a similar infix notation also for the other types of transition relations introduced later in the paper.

Our goal is to construct a transducer that recognizes the relation R^+, where $R^+ = \cup_{i>0} R^i$.

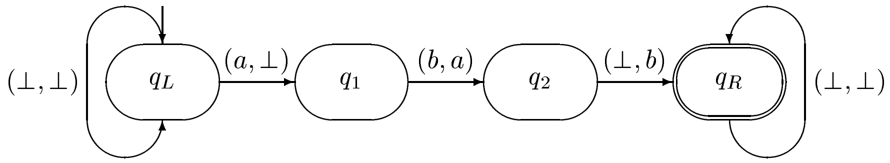
Starting from R, we can in a straight-forward way construct a transducer for R^+ whose states, called *columns*, are sequences of states in Q, where runs of transitions between columns of length i accept pairs of words in R^i. More precisely, define the *column transducer* for T as the tuple $T^+ = \langle Q^+, q_0^+, \Longrightarrow, F^+\rangle$ where

- Q^+ is the set of non-empty sequences of states of T,
- q_0^+ is the set of non-empty sequences of initial states of T,
- $\Longrightarrow$: $(Q^+ \times (\Sigma \times \Sigma)) \mapsto 2^{Q^+}$ is defined as follows: for any columns $q_1 q_2 \cdots q_m$ and $r_1 r_2 \cdots r_m$, and pair (a, a'), we have $q_1 q_2 \cdots q_m \stackrel{(a,a')}{\Longrightarrow} r_1 r_2 \cdots r_m$ iff there are $a_0, a_1, \ldots, a_m$ with $a = a_0$ and $a' = a_m$ such that $q_i \stackrel{(a_{i-1},a_i)}{\longrightarrow} r_i$ for $1 \leq i \leq m$,
- F^+ is the set of non-empty sequences of accepting states of T.

It is easy to see that T^+ accepts exactly the relation R^+: runs of transitions from q_0^i to columns in F^i accept transductions in R^i. The problem is that T^+ has infinitely many columns.

We will use x, y, etc. to denote columns in Q^+, and X, Y, etc. to denote sets of columns (i.e., subsets of Q^+). We use regular expressions notation for representing sets. In this paper, we present a procedure for incrementally generating a transducer which accepts the same relation as T^+. The procedure starts from T; by successively adding transitions of T^+ we compute a sequence of successively larger (in terms of sets of accepted pairs of words) transducers, all of which under-approximate R^+. Each new approximation is generated through performing a basic step. The step constructs transitions by combining already constructed transitions. Furthermore, all the time during this procedure, "equivalent" columns will be merged, in order to hopefully arrive at a finite-state result.

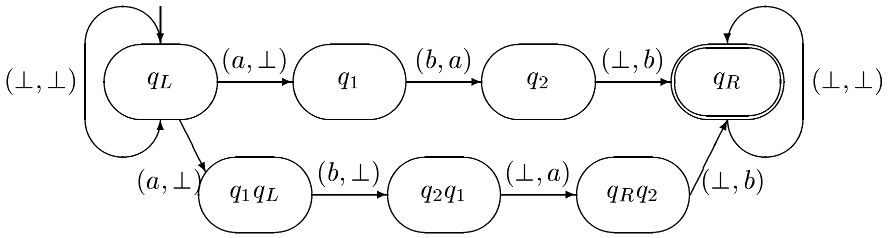
As a running example, consider the transducer below over the alphabet $\{\perp, a, b\}$. It relates a word of the form $\perp^i ab \perp^j$ with $\perp^{i+1} ab \perp^{j-1}$, moving the sequence ab one step to the right.



Our algorithm identifies pairs of transitions and combines them in the following way. When we have a transition from x to x' on (a, b), and a transition from y to y' on (b,c) we add the transition xx' to yy' on (a,c). Furthermore, we define an equivalence relation which enables us to merge columns in the following way. A state in Q is *left-copying* if all words in its prefix consist of pairs of identical symbols. A state in Q is *right-copying* if all words in its suffixes consist of pairs of identical symbols. In the above example, the states q_L and q_R are left- and right-copying, respectively. Now, two columns are *equivalent* if they can be made equal by repeatedly deleting identical neighbours which are either left- or right copying. For instance the columns $q_Lq_Lxq_R$ and $q_Lxq_Rq_R$ are equivalent. Applying this to our example, we get the following transitions.

- $q_L \overset{(a,\perp)}{\Longrightarrow} q_1$ and $q_L \overset{(\perp,\perp)}{\Longrightarrow} q_L$ give us $q_Lq_L \overset{(a,\perp)}{\Longrightarrow} q_1q_L$. We merge q_Lq_L and q_L.
- $q_1 \overset{(b,a)}{\Longrightarrow} q_2$ and $q_L \overset{(a,\perp)}{\Longrightarrow} q_1$ give us $q_1q_L \overset{(b,\perp)}{\Longrightarrow} q_2q_1$.
- $q_2 \overset{(\perp,b)}{\Longrightarrow} q_R$ and $q_1 \overset{(b,a)}{\Longrightarrow} q_2$ give us $q_2q_1 \overset{(\perp,a)}{\Longrightarrow} q_Rq_2$.
- $q_R \overset{(\perp,\perp)}{\Longrightarrow} q_R$ and $q_2 \overset{(\perp,b)}{\Longrightarrow} q_R$ give us $q_Rq_2 \overset{(\perp,b)}{\Longrightarrow} q_Rq_R$. We merge q_Rq_R and q_R.

The new transducer thus becomes:



3 Algorithm

In this section, we will formally present our algorithm. In particular, we define an equivalence relation on the set Q^+ of columns of T^+, which can be used to merge columns during the computation of R^+. The equivalence relation is not just language equivalence: equivalent columns may in general have different left quotients or right quotients. Instead, we shall prove that merging equivalent columns of T^+ does not increase the relation computed by T^+.

Formally, denoting the equivalence relation on Q^+ by $\simeq$, we define the *quotient transducer* $T_\simeq$ as $T_\simeq = \langle Q^+/\simeq, \{q_0\}^+, \Longrightarrow_\simeq, F^+/\simeq\rangle$ where

- $Q^+/\simeq$ is the set of equivalence classes of columns,
- q_0^+ is the initial equivalence class (this will indeed be one equivalence class of $\simeq$),
- $\Longrightarrow_\simeq$: $(Q^+/\simeq \times(\Sigma\times\Sigma)) \mapsto 2^{Q^+/\simeq}$ is defined in the natural way as follows:

$$X \overset{(a,a')}{\Longrightarrow}_\simeq X' \quad\Leftrightarrow\quad \exists x \in X, x' \in X' : x \overset{(a,a')}{\Longrightarrow} x'$$

- $F^+/\simeq$ is the partitioning of F^+ with respect to $\simeq$ (this will be well-defined since, as we shall see later, there are no columns x and y with $x \simeq y$, $x \in F^+$, and $y \notin F^+$).

Now, we define $\simeq$ as follows.

For a set X of columns, let $\mathsf{pref}(X)$ denote the set of prefixes of X, i.e., the set of words $w \in (\Sigma \times \Sigma)^*$ such that $q_0^+ \overset{w}{\Longrightarrow} x$ for some x in X. Analogously, let $\mathsf{suff}(X)$ denote the set of suffixes of X, i.e., the set of words $w \in (\Sigma \times \Sigma)^*$ such that there are $x \in X$ and $y \in F^+$ with $x \overset{w}{\Longrightarrow} y$.

A state $q \in Q$ is *left-copying* if all words in $\mathsf{pref}(\{q\})$ are of form $(a_1, a_1) \cdot (a_2, a_2) \cdots (a_n, a_n)$, i.e., containing only pairs of identical alphabet symbols. A state $q \in Q$ is *right-copying* if all words in $\mathsf{suff}(\{q\})$ are of form $(a_1, a_1) \cdot (a_2, a_2) \cdots (a_n, a_n)$. In other words, prefixes of left-copying states only copy input symbols to output symbols, and similarly for suffixes of right-copying states.

The equivalence classes of $\simeq$ will now be sets of form $e_1 e_2 \cdots e_n$ where each e_i is one of the following:

1. $\{q_L\}^+$, for some left-copying state q_L,
2. $\{q_R\}^+$, for some right-copying state q_R,
3. $\{q\}$, for some state q which is neither left-copying nor right-copying,

and where two consecutive e_i can be identical only if they are neither left-copying nor right-copying. In the following, we often omit the set notation, using q to denote the set $\{q\}$. Thus, equivalence classes in $Q^+/\simeq$ can be represented as columns $e_1 e_2 \cdots e_n$ where each e_i is of form q_L^+, q_R^+, or q, avoiding successive duplicates as described above.

Define the operator $\star$ as the natural concatenation operator on equivalence classes:

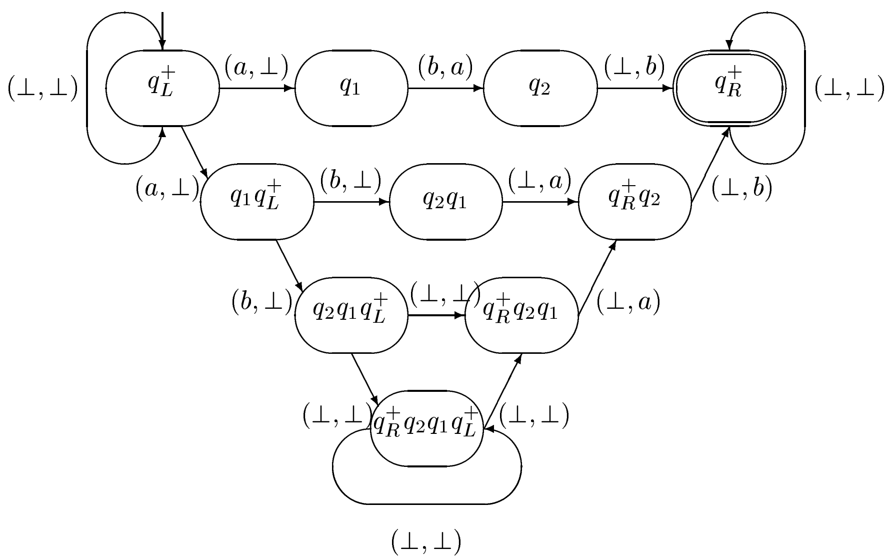
- $e_1 \star e_2$ is defined as
  - e_1 if $e_1 = e_2 = \{q\}^+$ and q is a left- or right-copying state,
  - $e_1 e_2$ otherwise.
- $(E_1 e_1) \star (e_2 E_2)$ is defined as $E_1 (e_1 \star e_2) E_2$ for equivalence classes $E_1 e_1$ and $e_2 E_2$.

Our procedure will now build a sequence $\widetilde{T}_0, \widetilde{T}_1, \widetilde{T}_2, \cdots$ of successively larger (in terms of sets of accepted pairs of words) transducers, all of which under-approximate R^+: the transition relation $\Longrightarrow_i$ of $\widetilde{T}_i$ will be a subset of $\Longrightarrow_\simeq$. The procedure incrementally adds transitions in $\Longrightarrow_\simeq$ between equivalence classes. The initial transducer $\widetilde{T}_0$ is obtained from T by taking all transitions in $\longrightarrow$ and replacing all left-copying states q_L by $\{q_L\}^+$, all right-copying states q_R by $\{q_R\}^+$, and all other states q by $\{q\}$.

In each step of the procedure, $\Longrightarrow_{i+1}$ is obtained from $\Longrightarrow_i$ by adding transitions of form $X \star X' \overset{(a,c)}{\Longrightarrow}_{i+1} Y \star Y'$ with $X \overset{(a,b)}{\Longrightarrow}_i Y$ and $X' \overset{(b,c)}{\Longrightarrow}_0 Y'$.

The algorithm terminates when the relation R^+ is accepted by $\widetilde{T}_i$. This can be tested by checking if the language of $\widetilde{T}_i \circ R$ is included in $\widetilde{T}_i$.

Continuing our example from Section 2, a partial transducer of the algorithm is shown below. Other transitions are also added, but they do not add to the language of the transducer.



4 Correctness

In this section we show correctness (soundness and completeness) of our construction. We do this in two steps. First, we prove (Corollary 1) that $T_{\simeq}$ is equivalent to T^+ in the sense that both transducers accept the same relation on words. Then, we relate the transducer generated by the algorithm in Section 3 to $T_{\simeq}$ proving its soundness (Theorem 3) and completeness (Theorem 4).

We also present sufficient conditions for termination of the algorithm, which implies that our approach is sufficiently general to cover several classes of systems considered in earlier works.

To start with, we need a technical result saying that, since T is deterministic, we can ignore columns containing distinct consecutive left-copying states.

Lemma 1. *Any column x which contains two distinct consecutive left-copying states, i.e., is of form $x = x_1 \cdot q_1 \cdot q_2 \cdot x_2$ where q_1 and q_2 are distinct left-copying states, is unreachable in T^+*

Proof. Follows directly from the fact that T is deterministic, and that the set of initial states of T^+ is q_0^+. □

Lemma 2. *The relation accepted by T^+ remains the same if we remove all columns that contain two distinct consecutive left-copying states. Analogously, the relation accepted by $T_{\simeq}$ remains the same if we remove all equivalence classes that contain two distinct consecutive left-copying states.*

Proof. Follows directly from Lemma 1, and the observation that if some column in an equivalence class contains two distinct consecutive left-copying states, then all columns in this equivalence class will also do so. □

In the rest of this paper, we will thus assume that all columns with two distinct consecutive left-copying states are removed from T^+ and $T_{\simeq}$.

Equivalence of $T_{\simeq}$ and T^{+} We prove equivalence of $T_{\simeq}$ and T^{+} (Corollary 1) by showing (Theorem 1) that the equivalence relation $\simeq$ contains a forward simulation and a backward simulation relation. This ensures that the merging of states will not add prefix/suffix combinations not existing in T^{+}.

A relation $\leq_F$ on the set of columns is a *forward simulation* if whenever $x \leq_F y$ and $x \overset{(a,b)}{\Longrightarrow} x'$ for some pair (a, b) of symbols and column x', there is a column y' such that $y \overset{(a,b)}{\Longrightarrow} y'$ and $x' \leq_F y'$.

Similarly, a relation $\leq_B$ on the set of columns is a *backward simulation* if whenever $x \leq_B y$ and $x' \overset{(a,b)}{\Longrightarrow} x$ for some pair (a, b) of symbols and column x', there is a column y' such that $y' \overset{(a,b)}{\Longrightarrow} y$ and $x' \leq_B y'$.

Theorem 1. *There is a forward simulation $\leq_F$ and a backward simulation $\leq_B$ with $\leq_F \subseteq \simeq$ and $\leq_B \subseteq \simeq$ such that for all columns x and y with $x \simeq y$, there is some column z such that $x \leq_F z$ and $y \leq_B z$.*

Proof. We must define the forward simulation $\leq_F$ and the backward simulation $\leq_B$ on columns. Let $e_1 e_2 \cdots e_n$ and $f_1 f_2 \cdots f_n$ be two equivalent columns, where for each k, we have either $e_k = f_k = q$ for some state q, or that $e_k = q^i$ and $f_k = q^j$ for some left- or right-copying state q. Define

- $e_1 e_2 \cdots e_n \leq_F f_1 f_2 \cdots f_n$ iff in addition $e_k = f_k$ whenever $e_k = q^i$ for some left-copying state q,
- $e_1 e_2 \cdots e_n \leq_B f_1 f_2 \cdots f_n$ iff in addition $e_k = f_k$ whenever $e_k = q^i$ for some right-copying state q.

We must prove that $\leq_F$ is a forward simulation, and that $\leq_B$ is a backward simulation. Let $x = e_1 e_2 \cdots e_n$ and $y = f_1 f_2 \cdots f_n$ be as above.

$\leq_F$: Assume $x \leq_F y$. If $x \overset{(a,b)}{\Longrightarrow} x'$, then x' is of form $x' = e'_1 e'_2 \cdots e'_n$. We can choose y' as $f'_1 f'_2 \cdots f'_n$, where $f'_k = e'_k$, except when e_k is of the form q^i for a right-copying state q. However, in this case, since T is deterministic, e'_k will be of form ${q'}^i$ for a right-copying state q', whence we can choose f'_k as ${q'}^j$.

$\leq_B$: Assume $x \leq_B y$. If $x' \overset{(a,b)}{\Longrightarrow} x$, then x' is of form $x' = e'_1 e'_2 \cdots e'_n$. We can choose y' as $f'_1 f'_2 \cdots f'_n$, where $f'_k = e'_k$, except when e_k is of the form q^i for a left-copying state q. However, in this case, by lemma 2, e'_k will be of form ${q'}^i$ for a left-copying state q', whence we can choose f'_k as ${q'}^j$.

For each pair $x = e_1 e_2 \cdots e_n$ and $y = f_1 f_2 \cdots f_n$ of equivalent columns, we can now find a z with $x \leq_F z$ and $y \leq_B z$ by taking z as $g_1 g_2 \cdots g_n$, where g_k is

- e_k if $e_k = f_k$,
- $g_k = e_k$ whenever $e_k = q^i$ for some left-copying state q,
- $g_k = f_k$ whenever $e_k = q^i$ for some right-copying state q. □

We are now ready to prove the main theorem of this section, namely that the set of traces of $T_{\simeq}$ is included in the set of traces of T^{+}.

Theorem 2. *$T_{\simeq}$ and T^+ have the same set of traces.*

Proof. Notice that since $T_{\simeq}$ is a collapsed version of T^+, it will obviously have more traces. We just need to show the inclusion in the other direction.

We will show that for each sequence of transitions of $T_{\simeq}$

$$X_0 \overset{(a_1,b_1)}{\Longrightarrow_{\simeq}} X_1 \overset{(a_2,b_2)}{\Longrightarrow_{\simeq}} \cdots \overset{(a_{n-1},b_{n-1})}{\Longrightarrow_{\simeq}} X_{n-1} \overset{(a_n,b_n)}{\Longrightarrow_{\simeq}} X_n$$

there is a corresponding sequence of transitions of T^+.

$$x_0 \overset{(a_1,b_1)}{\Longrightarrow} x_1 \overset{(a_2,b_2)}{\Longrightarrow} \cdots \overset{(a_{n-1},b_{n-1})}{\Longrightarrow} x_{n-1} \overset{(a_n,b_n)}{\Longrightarrow} x_n$$

where $x_i \in X_i$ for $i = 0, \ldots, n$. We show this by induction on the length n of the sequence.

•**Base case:** The empty trace is trivially in both $T_{\simeq}$ and T^+.

•**Inductive case:** Assume that the property is true for n. Let $w = w_1 \cdot (a_{n+1}, b_{n+1})$ be a trace of length $n+1$. Then w_1 is a trace of length n, and is thus, by the induction hypothesis, also accepted by T^+ as in the display above. w is accepted in a sequence of transitions

$$X_0 \overset{(a_1,b_1)}{\Longrightarrow_{\simeq}} \cdots \overset{(a_n,b_n)}{\Longrightarrow_{\simeq}} X_n \overset{(a_{n+1},b_{n+1})}{\Longrightarrow_{\simeq}} X_{n+1}$$

meaning that there are $y_n \in X_n$ and $y_{n+1} \in X_{n+1}$ such that $y_n \overset{(a_{n+1},b_{n+1})}{\Longrightarrow} y_{n+1}$. Since $x_n \in X_n$ we have $x_n \simeq y_n$, and hence there is a z_n such that $x_n \leq_B z_n$ and $y_n \leq_F z_n$. From $y_n \leq_F z_n$ we infer that there is a $z_{n+1} \in X_{n+1}$ such that

$$z_n \overset{(a_{n+1},b_{n+1})}{\Longrightarrow} z_{n+1}$$

From $x_n \leq_B z_n$ we infer that there is a sequence

$$z_0 \overset{(a_1,b_1)}{\Longrightarrow} \cdots \overset{(a_n,b_n)}{\Longrightarrow} z_n$$

such that $x_i \leq_F z_i$ for $i = 0, \ldots, n$, implying that $z_i \in X_i$ for $i = 0, \ldots, n$. We can thus conclude that the sequence

$$z_0 \overset{(a_1,b_1)}{\Longrightarrow} \cdots \overset{(a_n,b_n)}{\Longrightarrow} z_n \overset{(a_{n+1},b_{n+1})}{\Longrightarrow} z_{n+1}$$

satisfies the conditions for the inductive step.

□

From this theorem, we can deduce that $T_{\simeq}$ and T^+ accept the same relation.

Corollary 1. *$T_{\simeq}$ and T^+ accept the same relation.*

Proof. We notice that the union of the sets attached to each final state of $T_{\simeq}$ is the set of final columns of T^+ (they form a partition of it w.r.t $\simeq$). Thus we can conclude that any trace that is an accepting run in one automaton is also an accepting run in the other automaton. □

Soundness and Completeness We are now ready to prove the soundness and completeness of the algorithm. For a column x, let $[x]_{\simeq}$ denote the equivalence class for x. We will use the following property of the operator $\star$ to prove the soundness.

Lemma 3. *For any sets X, X' associated with some equivalence classes, we have that $X \cdot X' \subseteq X \star X'$.*

Proof. By observing that $E_1 \cdot q^+ \cdot q^+ \cdot E_2 \subseteq E_1 \cdot q^+ \cdot E_2$. □

The following is the soundness theorem.

Theorem 3. *For every k, $\Longrightarrow_k \subseteq \Longrightarrow_{\simeq}$.*

Proof. For $k = 0$, let X, Y be two sets of columns such that $X \overset{(a,a')}{\Longrightarrow}_0 Y$ for some pair (a, a'). Since $\Longrightarrow_0$ is obtained from T by substituting each state with its equivalence-class, we must have that $X = [q]_{\simeq}$ and $Y = [q']_{\simeq}$ for some states q, q' such that $q \overset{(a,a')}{\longrightarrow} q'$. Thus $X \overset{(a,a')}{\Longrightarrow}_{\simeq} Y$.

Now take $k > 0$ and assume that for all $k' < k$ we have $\Longrightarrow_{k'} \subseteq \Longrightarrow_{\simeq}$. Let X, X', Y, Y' be sets of columns such that $X \overset{(a,b)}{\Longrightarrow}_{k-1} Y$ and $X' \overset{(b,c)}{\Longrightarrow}_0 Y'$. Then we have to show that $X \star X' \overset{(a,c)}{\Longrightarrow}_{\simeq} Y \star Y'$. By induction, we have that $X \overset{(a,b)}{\Longrightarrow}_{\simeq} Y$ and $X' \overset{(b,c)}{\Longrightarrow}_{\simeq} Y'$. Thus, there are $x \in X, y \in Y, x' \in X', y' \in Y'$ such that $x \overset{(a,b)}{\Longrightarrow} y$ and $x' \overset{(b,c)}{\Longrightarrow} y'$, and hence, $x \cdot y \overset{(a,c)}{\Longrightarrow} x' \cdot y'$. Since $x \cdot y \in X \cdot Y$ and $x' \cdot y' \in X' \cdot Y'$, by Lemma 3 we get that $x \cdot y \in X \star Y$ and $x' \cdot y' \in X' \star Y'$, and thus $X \star X' \overset{(a,c)}{\Longrightarrow}_{\simeq} Y \star Y'$. □

The following completeness theorem states that any pair in the transitive closure will eventually be generated by the algorithm.

Theorem 4. *Let (w, w') be a word in R^+. Then there is some k such that $\widetilde{T}_k$ accepts (w, w').*

Proof. Let $x_1, x_2, \cdots, x_n$ be a run of T^+ accepting (w, w'). This run can be organized as columns of the following matrix:

$$
\begin{array}{ccccccc}
q_1^1 & \xrightarrow{(a_1^0, a_1^1)} & q_2^1 & \xrightarrow{(a_2^0, a_2^1)} & \cdots \; q_{n-1}^1 & \xrightarrow{(a_{n-1}^0, a_{n-1}^1)} & q_n^1 \\
q_1^2 & \xrightarrow{(a_1^1, a_1^2)} & q_2^2 & \xrightarrow{(a_2^1, a_2^2)} & \cdots \; q_{n-1}^2 & \xrightarrow{(a_{n-1}^1, a_{n-1}^2)} & q_n^2 \\
 & & & & \vdots & & \\
q_1^m & \xrightarrow{(a_1^{m-1}, a_1^m)} & q_2^m & \xrightarrow{(a_2^{m-1}, a_2^m)} & \cdots \; q_{n-1}^m & \xrightarrow{(a_{n-1}^{m-1}, a_{n-1}^m)} & q_n^m
\end{array}
$$

where for all i with $1 \leq i < n$ and all j with $1 \leq j \leq m$ we have that $q_i^j \overset{(a_i^{j-1}, a_i^j)}{\longrightarrow} q_{i+1}^j$.

We now prove by induction on the number of rows of this matrix that the pair (w, w') is eventually accepted by the transducer built by the algorithm.

By the definition of $\Longrightarrow_0$ we get that $[q_i^j]_{\simeq} \overset{(a_i^{j-1}, a_i^j)}{\Longrightarrow}_0 [q_{i+1}^j]_{\simeq}$, for all i with $1 \leq i < n$ and all j with $1 \leq j \leq m$. Taking $j = 1$, we get that $[q_1^1]_{\simeq}, [q_2^1]_{\simeq}, \ldots, [q_n^1]_{\simeq}$ is a run of $\widetilde{T}_0$ accepting $(a_1^0, a_1^1) \cdot (a_2^0, a_2^1) \ldots (a_{n-1}^0, a_{n-1}^1)$.

Now suppose that in some step i in the algorithm, for some k we have built the transition relation $\Longrightarrow_i$ such that $E_1, E_2, \ldots, E_n$ is a run of $\widetilde{T}_i$ accepting $(a_1^0, a_1^k) \cdot (a_2^0, a_2^k) \cdots (a_{n-1}^0, a_{n-1}^k)$. Then since $[q_1^{k+1}]_{\simeq}, [q_2^{k+1}]_{\simeq}, \ldots, [q_n^{k+1}]_{\simeq}$ is a run of $\widetilde{T}_0$ accepting $(a_1^k, a_1^{k+1}) \cdot (a_2^k, a_2^{k+1}) \cdots (a_{n-1}^k, a_{n-1}^{k+1})$, transitions will be in $\Longrightarrow_{i+1}$ such that $E_1 \star [q_1^{k+1}]_{\simeq}, E_2 \star [q_2^{k+1}]_{\simeq} \ldots E_n \star [q_n^{k+1}]_{\simeq}$ is a run of $\widetilde{T}_{i+1}$ accepting $(a_1^0, a_1^{k+1}) \cdot (a_2^0, a_2^{k+1}) \cdots (a_{n-1}^0, a_{n-1}^{k+1})$. $\square$

Termination The termination of our method is dependent on the number of different equivalence classes of the form $e_1 e_2 \cdots e_n$ which might be generated during construction of the transitive closure. This number in turn depends on two parameters:

1. the number of non-copying states in columns, and
2. the number of alternations of copying states in columns.

Therefore a bound on these two parameters is a sufficient condition for termination.

In [JN00], we introduced a class of systems which satisfied the *bounded local depth* property. Roughly speaking, this property means that there is a bound on the number of times each position in a word is rewritten when applying the transducer to the word an arbitrary number of times. For example, a system passing a token to the right has local depth 2, since each position can be rewritten at most twice; once when passing the token, and once when receiving it. In [JN00], we also assumed that there could only be at most one left-copying state and one right-copying state. For this class of systems, it can be shown that for each pair of words in the transitive closure, there is a run of the column transducer having at most two alternations of the left-copying state and the right-copying state. Thus, our construction will also terminate under the condition of bounded local depth.

In [Cau00], a class of rewriting systems is considered which has the *right-overlapping* (or symmetrically the *left-overlapping*) property. This means that all rewritings must occur in strict order from the left to the right. In such a case, our construction gives columns of the form $q_R^+ q_1 q_2 \cdots q_n q_L^+$ where q_L is a left-copying state, q_R is a right-copying state and each q_i is some state representing a rewriting. This implies that the number of alternations of copying states is at most one, and that the number of non-copying states is bounded.

5 Implementation

We have implemented the technique in this paper and run it on a number of mutual exclusion and termination detection protocols. We have compared the performance of the algorithm with our earlier work [BJNT00].

The technique in [BJNT00] is based on applying subset construction to the column transducer and on-the-fly identification of equivalent (w.r.t. suffixes) states. The subset construction technique represents sets of states (columns) by finite-state automata and involves several operations on regular sets, such as

- computing post-images of sets of columns represented by finite-state automata,
- saturating generated sets of columns, and
- testing saturated sets for equality against all previous sets.

All the above operations can potentially be expensive. In contrast, our new algorithm represents the equivalence classes as vectors of states. The concatenation operator is a variant of concatenation of vectors, and equivalence checking of vectors is fast and can be hashed effectively.

Furthermore, we have implemented an obvious optimization to our new method to avoid generating useless states, namely, in each step, we only merge transitions where $x \stackrel{(a,b)}{\Longrightarrow} x'$ and $y \stackrel{(b,c)}{\Longrightarrow} y'$ only if all x, x', y, y' are both reachable and productive in the transducer obtained in the previous step. Using this technique, we substantially reduce the number of generated useless states.

We have used both methods to compute the transitive closures of a set of transitions in some algorithms. It should be mentioned that it is not clear whether all operations in the subset construction method are implemented in the most efficient way, since there are many ways to represent automata. Nevertheless, the initial experiments indicate that the new method runs several times faster. Furthermore, there are a large number of potential optimizations which have still not been carried out.

The results are shown in Table 1.

Algorithm	Subset construction	Matching	Speedup
Dijkstra	435s	39s	11.2
Szymanski	278s	178s	1.5
Termination detection	47s	22s	2.1
Ticket	17s	20s	0.85

Table 1. Improvements of new method compared to subset construction method

For the ticket algorithm, the new algorithm performed slightly worse. The ticket algorithm is a rather small example, and it seems that the new method scales better than the old one. We expect to be able to improve the new method by considering different ways of scheduling the matching operations. Also, it may be possible to find ways to remember already tried combinations to avoid repeated work.

6 Conclusions and Future Research

We have presented a new technique for performing regular model checking. More precisely, given a finite-state transducer, our algorithm generates a new transducer corresponding to the transitive closure of the original one. The algorithm involves two ingredients, namely a matching operation which combines existing transitions to add new ones, and an equivalence relation which enables us to merge states. An important property of the equivalence relation is that it is syntactically characterized and hence possible to decide locally.

A crucial aspect in the application of the algorithm is the order in which the matching operation is performed on transitions. By defining appropriate matching strategies, we believe that our algorithm can be made both to uniformly simulate existing algorithms for parameterized protocols [JN00,BJNT00], rewriting systems [Cau00], push-down systems [BEM97,Cau92,FWW97], etc, and to produce more efficient versions of these algorithms. Furthermore, we think that the generality of construction will enable us to extend the algorithm to other classes of relations than those on words, e.g., relations on trees and graphs. This would allow us to verify systems with dynamic behaviours such as data security protocols, mobile protocols, etc.

Acknowledgement

This work was partially supported by the European project ADVANCE, Contract No. IST – 1999-29082.

References

[ABJ98] Parosh Aziz Abdulla, Ahmed Bouajjani, and Bengt Jonsson. On-the-fly analysis of systems with unbounded, lossy fifo channels. In *Proc.* 10^{th} *Int. Conf. on Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 305–318, 1998.

[ABJN99] Parosh Aziz Abdulla, Ahmed Bouajjani, Bengt Jonsson, and Marcus Nilsson. Handling global conditions in parameterized system verification. In *Proc.* 11^{th} *Int. Conf. on Computer Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 134–145, 1999.

[BCMD92] J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98:142–170, 1992.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model Checking. In *Proc. Intern. Conf. on Concurrency Theory (CONCUR'97)*. LNCS 1243, 1997.

[BFL] B. Boigelot, J-M. Franois, and L. Latour. The Liége automata-based symbolic handler (lash). Available at `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.

[BG96] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In Alur and Henzinger, editors, *Proc. 8^{th} Int. Conf. on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 1–12. Springer Verlag, 1996.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *Proc. of the Fourth International Static Analysis Symposium*, Lecture Notes in Computer Science. Springer Verlag, 1997.

[BH97] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of fifo-channel systems with nonregular sets of configurations. In *Proc. ICALP '97, 24^{th} International Colloquium on Automata, Lnaguages, and Programming*, volume 1256 of *Lecture Notes in Computer Science*, 1997.

[BJNT00] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In Emerson and Sistla, editors, *Proc. 12^{th} Int. Conf. on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418, 2000.

[BMT01] A. Bouajjani, A. Muscholl, and T. Touili. Permutation rewriting and algorithmic verification. In *Proc. LICS' 01 17^{th} IEEE Int. Symp. on Logic in Computer Science*. IEEE, 2001.

[BW94] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *Proc. 6^{th} Int. Conf. on Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 55–67. Springer Verlag, 1994.

[Cau92] Didier Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106(1):61–86, Nov. 1992.

[Cau00] Didier Caucal. On word rewriting systems having a rational derivation. In *FOSSACS 2000*, volume 1784 of *Lecture Notes in Computer Science*, pages 48–62, April 2000.

[CJ98] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *CAV'98*. LNCS 1427, 1998.

[DLS01] D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, 2001.

[ES01] J. Esparza and S. Schwoon. A bdd-based model checker for recursive programs. In *Proc. 13^{th} Int. Conf. on Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 324–336, 2001.

[FWW97] A. Finkel, B. Willems, , and P. Wolper. A direct symbolic approach to model checking pushdown systems (extended abstract). In *Proc. Infinity'97, Electronic Notes in Theoretical Computer Science*, Bologna, Aug. 1997.

[HJJ+96] J.G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *Proc. TACAS '95, 1^{th} Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1019 of *Lecture Notes in Computer Science*, 1996.

[JN00] Bengt Jonsson and Marcus Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In S. Graf and M. Schwartzbach, editors, *Proc. TACAS '00, 6^{th} Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *Lecture Notes in Computer Science*, 2000.

[KMM$^+$97] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In O. Grumberg, editor, *Proc. 9^{th} Int. Conf. on Computer Aided Verification*, volume 1254, pages 424–435, Haifa, Israel, 1997. Springer Verlag.

[KMMG97] P. Kelb, T. Margaria, M. Mendler, and C. Gsottberger. Mosel: A flexible toolset for monadic second–order logic. In *Proc. of the Int. Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'97), Enschede (NL)*, volume 1217 of *Lecture Notes in Computer Science (LNCS)*, pages 183–202, Heidelberg, Germany, March 1997. Springer–Verlag.

[PS00] A. Pnueli and E. Shahar. Liveness and acceleration in parameterized verification. In *Proc. 12^{th} Int. Conf. on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 328–343, 2000.

[Tou01] T. Touili. Regular Model Checking using Widening Techniques. *Electronic Notes in Theoretical Computer Science*, 50(4), 2001. Proc. Workshop on Verification of Parametrized Systems (VEPAS'01), Crete, July, 2001.

[WB98] Pierre Wolper and Bernard Boigelot. Verifying systems with infinite but regular state spaces. In *Proc. 10th Int. Conf. on Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 88–97, Vancouver, July 1998. Springer Verlag.