Transitive Closures of Regular Relations for Verifying Infinite-State Systems*

Bengt Jonsson and Marcus Nilsson

Dept. of Computer Systems, P.O. Box 325, S-751 05 Uppsala, Sweden
{bengt,marcusn}@docs.uu.se

Abstract. We consider a model for representing infinite-state and parameterized systems, in which states are represented as strings over a finite alphabet. Actions are transformations on strings, in which the change can be characterized by an arbitrary finite-state transducer. This program model is able to represent programs operating on a variety of data structures, such as queues, stacks, integers, and systems with a parameterized linear topology. The main contribution of this paper is an effective derivation of a general and powerful *transitive closure* operation for this model. The transitive closure of an action represents the effect of executing the action an arbitrary number of times. For example, the transitive closure of an action which transmits a single message to a buffer will be an action which sends an arbitrarily long sequence of messages to the buffer. Using this transitive closure operation, we show how to model and automatically verify safety properties for several types of infinite-state and parameterized systems.

1 Introduction

In recent years, substantial progress has been made regarding the automated verification of finite-state systems. Fully automated techniques have now been developed to the extent that they can routinely handle systems with millions of states. Partial order techniques and symbolic representations, such as Binary Decision Diagrams (BDDs) [BCM+90] have been important in this development. There is also progress in the development of verification algorithms for infinite-state systems (e.g., [ACD90,AH89,BS95,Sti96,Fin94,AJ96]), and for parameterized systems, i.e., systems consisting of an arbitrary number of homogeneous processes connected in a regular topology (e.g., [GS92,KMM+97,ABJN99]).

The problem of verifying that a system satisfies a certain correctness property is usually reduced to checking some form of reachability problem on a transition system model of the system. For example, verifying that a system never gets into an "unsafe" state consists in checking that no "unsafe" state can be reached (by a sequence of transitions) from the set of initial states. This problem is often analyzed by symbolic or enumerative state-space exploration, starting from

* support in part by the ASTEC competence center, and by the Swedish Board for Industrial and Technical Development (NUTEK)

S. Graf and M. Schwartzbach (Eds.): TACAS/ETAPS 2000, LNCS 1785, pp. 220–235, 2000.

the set of initial states. However, naive reachability analysis is guaranteed to terminate only when the reachable state-space has finite depth, meaning that there is a uniform bound on the number of transitions needed to get to any reachable state. Finite-state systems trivially have a state-space with finite depth (bounded by the number of states), but the depth of the state-space of an infinite-state system is in general infinite.

In this paper, we will consider infinite-state systems whose states can be represented as finite strings over a finite alphabet. This includes parameterized systems consisting of an arbitrary number of homogeneous finite-state processes connected in a linear or ring-formed topology: take the (finite) set of local states of each process as the alphabet, and let a string of process states represent a system state (the length of the string is equal to the number of processes). Our model also includes systems that operate on queues, stacks, integers, and other data structures that can be represented as sequences of symbols. We represent the transition relation of such a system by a finite set of actions; each action is a regular relation between strings, which can be represented, e.g., by a finite-state transducer.

For this class of systems, reachability analysis can be performed as follows. Assume that a set of initial states and a set of "unsafe" states are both given as regular sets. Using the transducer representation of actions, we can calculate the set of successors of a regular set of states as a regular set. Explore the reachable states by successive calculations of sets of successor states, starting from the set of initial states. If the set of reachable states has finite depth, this exploration is guaranteed to terminate. Dually, we can perform the exploration backwards, starting from the unsafe states and taking predecessors, with guaranteed termination if the "backwards depth" of the state-space is finite. This approach is pursued, e.g., by Kesten et al. [KMM+97], who illustrate their approach by some examples with finite backwards depth.

In general, however, an infinite-state or parameterized system need neither have a finite depth nor a finite backwards depth. To explore the entire state-space, one must then be able to calculate the effect of arbitrarily long sequences of transitions. This problem would be solved if we could calculate the transitive closure of an action, and then adding this transitive closure to set of actions that are explored during verification. Remember that an action can be seen as a relation on states: therefore the transitive closure of an action represents the effect of executing the action an arbitrary number of times. For example, the transitive closure of an action which transmits a single message to a buffer will be an action which sends an arbitrarily long sequence of messages to the buffer. The transitive closure of an action in which a process in a parameterized system passes a token to its neighbor, will be an action that passes the token through an arbitrary sequence of neighbors to another process.

The main contribution of this paper is a construction of the transitive closure for a large class of actions in the system model described above. This transitive closure is also representable as a finite-state transducer, and can therefore be used in the state-space exploration in the same way as the original actions. Further-

more, we show how this construction makes is possible to verify safety properties of many parameterized and infinite-state systems fully automatically. We reduce safety properties to reachability using standard techniques. To check reachability, we first calculate the transitive closure of each action, if possible, and thereafter perform reachability analysis with the extended set of actions. In many cases, this analysis terminates even when the state space does not have finite depth. We have implemented the technique, using the Mona package [HJJ+96a], and present some examples.

In order to characterize the class of actions for which our transitive-closure construction works, we define a notion of *local depth*. Intuitively, an action has local depth k if each position in the string is transformed at most k times in any sequence of executions of that action. For example, an action in a parameterized system, in which a process passes a token to its right neighbor has local depth 2, since in an arbitrary execution sequence, each process is affected at most twice: once when receiving the token, and once when sending the token. Similarly, if a queue is modeled by an unbounded array with a finite contiguous segment of positions filled by messages, then an action which sends a message to the channel, thus putting a message into an empty position, has local depth 1. The main theorem of the paper shows that we can calculate the transitive closure of any action with finite local depth, and represent it as a finite-state transducer. We can also approximate the transitive closure of an action which does not have finite local depth. For an arbitrary k, we can calculate, as a finite-state transducer, the action which corresponds to executing the action an arbitrary number of times, subject to the constraint that each position is transformed at most k times.

A special case of the present paper is our earlier work [ABJN99]. There we consider parameterized systems, whose state can be represented by a string, and where each action is allowed to change the string in only one position. By requiring this change to be idempotent, each action gets local depth 1, and we could give a construction of its transitive closure. The restrictions limited the applicability to certain classes of parameterized algorithms.

Related Work The use of regular sets as a symbolic state representation in the verification of infinite-state systems has been proposed by e.g., Boigelot and Wolper [WB98] and Kesten et al. [KMM+97]. Kesten et al. perform backwards reachability analysis on state spaces of finite depth or finite backward depth. The decidability results for systems with unbounded lossy FIFO channels [AJ96] and for well-structured systems [AČJYK96] follow from the fact that the considered verification problems have a finite backward depth. Other researchers, e.g., [FO97], use regular sets in a deductive framework, where basic manipulations on regular sets are performed automatically, e.g., using the Mona [HJJ+96a] or MoSel [KMMG97] packages. In [Sis97], parameterized systems are verified using induction on the number of processes, where the inductive invariant is specified using automata on two dimensional strings.

A related technique for analyzing unbounded sequences of executions of an action, called *acceleration*, has been developed for finite state processes that com-

municate via unbounded FIFO channels [BG96,BGWW97,BH97,ABJ98] and for systems that operate on integer variables [BW94]. An acceleration of an action can be seen as the application of the transitive closure to a particular set of states. By a suitable representation, we can model the operations considered in the work on QDDs [BG96] as actions with finite local depth. Bouajjani and Habermehl [BH97] also perform acceleration of actions with non-finite local depth, sometimes resulting in non-regular sets of states. The acceleration operation is related to the *widening* operation, used for computing fixpoints in abstract interpretation [CC77], but aims at computing an *exact* fixpoint of a sequence of approximations, each of which represents a bounded number of executions of an action.

Outline In the next section, we define our model of systems and illustrate it by modeling a parameterized termination detection algorithm intended for a ring topology. In Section 3, we review the principles of symbolic reachability analysis for verifying safety properties, and note that this analysis would be significantly improved by taking the transitive closure of actions. Section 4 presents the main result: a construction for computing the transitive closure of an action with local depth k, for arbitrary k. Section 5 discusses two extra composition operations which also augment the power of the analysis. In Section 6, we outline how our results can be used for modeling and analysis of programs that operate on unbounded FIFO channels and integers. An implementation of our method, and the modeling and automated analysis of several infinite-state algorithms is reported in Section 7. Section 8 contains conclusions.

2 Program Model

In this section, we introduce our model of programs. A global state (or a configuration) of a system is represented as a string over a finite alphabet C. As usual, C^* is the set of finite strings over C. The dynamic behavior of a system is defined through a finite set of *actions*. Each action rewrites a certain portion of the string that represents the state. The rewriting relation is given by a finite-state transducer. The rewriting may furthermore be conditioned on the sequence of symbols to the right and to the left of the rewritten portion of the string. We use subclasses of regular languages, called *left contexts* and *right contexts*, to represent such conditions.

Definition 1. A *left context* is a regular language which can be accepted by a deterministic finite-state automaton with a unique accepting state, and where all outgoing transitions from the accepting state are self-loops. (transitions with identical source and target states). A *right context* is a language such that the language of reversed strings is a left context. The *tail* of a left context is the set of symbols that label self-loops from the accepting state. The tail of a right context is the tail of the left context which is its reverse language. □

Examples of left contexts are a^* (with tail $\{a\}$), and $(a+b)^*b(a+b)^*$ (with tail $\{a,b\}$). An example of a regular language which is not a left context is $(a+b)^*b$. This language is, however, a right context with tail $\{a,b\}$.

We will represent (length-preserving) relations on C^* by *finite-state transducers*. A finite-state transducer defines a regular language over $C \times C$. It represents the relation on C^* which contains the pair $(c_1 c_2 \cdots c_n \ , \ c'_1 c'_2 \cdots c'_n)$ iff the transducer accepts the string $(c_1, c'_1)\ (c_2, c'_2)\ \cdots\ (c_n, c'_n)$.

We are now ready to give the formal definition of our model.

Definition 2. A *program* is a triple $\mathcal{P} = \langle C, \phi_I, \mathcal{A} \rangle$ where

C is a finite alphabet, called the set of *colors*,
ϕ_I is a regular set over C, denoting a set of *initial configurations*, and
$\mathcal{A}$ is a finite set of *actions*. An *action* is a triple

$$\phi_L \ \boxed{\tau} \ \phi_R$$

where ϕ_L is a left context, ϕ_R is a right context, and τ is a regular set over $C \times C$.

A *configuration* γ of a program $\mathcal{P}$ is a string $\gamma[1]\ \gamma[2]\ \cdots\ \gamma[n]$ over C. For a regular expression ϕ, we use $\gamma \in \phi$ to denote that γ is a string in the language denoted by ϕ. For $i, j : 1 \leq i \leq j \leq n$, we use $\gamma[i \,..\, j]$ to denote the substring $\gamma[i]\ \gamma[i+1]\ \cdots\ \gamma[j]$. An action

$$\alpha = \phi_L \ \boxed{\tau} \ \phi_R$$

defines a relation α on configurations such that $(\gamma, \gamma') \in \alpha$ if γ and γ' are of equal length n, there are i, j with $1 \leq i \leq j \leq n$ such that

- $(\gamma[i], \gamma'[i])(\gamma[i+1], \gamma'[i+1]) \cdots (\gamma[j], \gamma'[j]) \in \tau$,
- $\gamma[1 \,..\, i-1] = \gamma'[1 \,..\, i-1] \in \phi_L$, and
- $\gamma[j+1 \,..\, n] = \gamma'[j+1 \,..\, n] \in \phi_R$.

If these conditions hold, we say that $\alpha(\gamma, \gamma')$ holds with *active index pair* (i, j). □

The above program model is a generalization of the one in our earlier work [ABJN99]. Our earlier model had the same structure, but the middle component τ in an action was constrained to be a relation on C, instead of a (regular) relation on C^*, implying that each action can only change one position in the string. With the new definition, a much wider class of systems can be modeled.

Example To illustrate our program model, we will model an algorithm for termination detection in a ring-shaped network, due to Dijkstra, Feijen, and van Gasteren [DFvG83]. The algorithm is intended to detect termination of an underlying computation among a ring of N processes, numbered from 1 to N. Each process can spontaneously change state from *computing* (non-idle) to *idle*, but process i can change from *idle* to *computing* only if process $i-1$ (or N if $i = 1$) is

computing. The system is *terminated* when all processes are idle. The detection algorithm employs a token, which is sent around by process 1, when it is idle, in increasing order of indices until it reaches process 1 again. When the token is sent out, it is white. Each process passes the token on, and paints it black if it is non-idle. If it comes back to process 1 and is still white, then termination is signaled provided that process 1 was idle during the entire round. Otherwise another round of termination detection will be started at a later moment.

We model the state of the algorithm by a string, where the ith element represents the state of process i. The state of process i is defined by a boolean variable q_i which is true iff process i is idle, and a variable t_i ranging over $\{\mathbf{black}, \mathbf{white}, \mathbf{none}\}$, which has value **none** when process i does *not* have the token, and otherwise denotes the color of the token. In addition, process 1 has a boolean variable *wasq* (w for short), which is true if it has stayed idle during the current round. Thus, the set of colors is the set of triples of form $\langle q, t, w\rangle$ where q and w are boolean, and $t \in \{\mathbf{black}, \mathbf{white}, \mathbf{none}\}$. The value of w is relevant only in position 1.

The set of initial states of the system, in which process 1 has a black token, is described by the regular expression

$$(t = \mathbf{black}) \quad (t = \mathbf{none})^*$$

where we use predicates to denote sets of colors, e.g., $(t = \mathbf{black})$ denotes the set of triples $\langle q, t, w\rangle$ such that $(t = \mathbf{black})$. An undesired state, in which detection is signaled although the system is not terminated is given by the regular expression

$$\left[(t = \mathbf{white} \ \wedge \ w) \ \ true^* \ \ \neg q \ \ true^*\right] \quad \cup \quad \left[(\neg q \ \wedge \ t = \mathbf{white} \ \wedge \ w) \ \ true^*\right]$$

which states that the condition $t = \mathbf{white} \ \wedge \ w$ for process 1 to signal detection is satisfied, but some process is not idle.

Let us then give examples of actions. An action in which some process i with $1 < i < N$ passes the token to its next neighbor, possibly after painting it black, is described by the action $(t = \mathbf{none})^+ \ \boxed{\tau} \ (t = \mathbf{none})^*$ where τ is the relation between strings of length two such that

$$\tau \quad \big(\ \langle q_1, t_1, w_1\rangle \ \langle q_2, t_2, w_2\rangle \quad , \quad \langle q_1', t_1', w_1'\rangle \ \langle q_2', t_2', w_2'\rangle \ \big)$$

iff $q_1' = q_1, \quad q_2' = q_2, \quad t_2 = t_1' = \mathbf{none}$, and $t2' = \mathbf{if}\ q_1\ \mathbf{then}\ t_1\ \mathbf{else}\ \mathbf{black}$. We also need an action which models the passing of the token from process N to process 1. This is the action $\{\epsilon\} \ \boxed{\tau} \ \{\epsilon\}$ where τ is the relation between strings of length at least two such that

$$\tau\big(\ \langle q_1, t_1, w_1\rangle \ \gamma \ \langle q_2, t_2, w_2\rangle \quad , \quad \langle q_1', t_1', w_1'\rangle \ \gamma \ \langle q_2', t_2', w_2'\rangle \ \big)$$

where γ is any string in $(t = \mathbf{none})^*$, where $q_2' = q_2, \quad q_1' = q_1, \quad t_1 = t_2' =$ **none**,, and where $t_1' = \mathbf{if}\ q_2\ \mathbf{then}\ t_2\ \mathbf{else}\ \mathbf{black}$. In addition, we need an action for passing the token from process 1 to process 2, which is given in an analogous way. In Section 4, we will see that the first action has local depth 2, and that

its transitive closure represents an action which passes the token from position i to position j for arbitrary $1 < i < j < N$. However, it is not meaningful to take the transitive closure of the second action, since it disables itself after being executed.

Summarizing, we see that the passing of a token from one process to the next is modeled by one action for the "standard" case $1 < i < N$ and by separate actions for special cases, such as passing from N to 1, and passing from 1 to 2. The changes to q, modeling the underlying computation, can be modeled in a similar way.

The above algorithm is an example of a parameterized distributed algorithm which assumes a linear or ring topology. In our earlier work [ABJN99], we were able to model a restricted class of parameterized algorithms where only one process changed its local state in each transition. In the above algorithm, two processes change their state simultaneously. In Section 6, we will describe how we can also model and analyze programs that operate on unbounded queues, and unbounded integers.

3 The Reachability Problem

We write $\gamma_1 \longrightarrow \gamma_2$ to denote that $\alpha(\gamma_1, \gamma_2)$ for some action $\alpha \in \mathcal{A}$. We use $\stackrel{*}{\longrightarrow}$ to denote the transitive closure of $\longrightarrow$. A configuration γ is said to be *reachable* if there is a configuration $\gamma_I \in \phi_I$ such that $\gamma_I \stackrel{*}{\longrightarrow} \gamma$.

The *reachability problem* is defined as follows.

Instance A program $\mathcal{P}$ and a set of configurations of $\mathcal{P}$ represented by a regular expression ϕ_F.

Question Is any $\gamma \in \phi_F$ reachable?

It is well-known (e.g., [VW86]) that the problem of verifying linear-time safety properties can be transformed into the problem of checking that a set of "bad" states is not reachable.

The reachability problem can be analyzed using standard *symbolic reachability analysis* to explore the state-space. The analysis maintains a set of reachable configurations, which is initially the set of initial configurations. At each step of the algorithm the set of reachable configurations is extended with the configurations that can be reached by executing some action in the program from a configuration in the current set

We use regular sets of strings to represent (in general infinite) sets of configurations. A regular set is represented by an automaton. The effect of executing an action, which is represented by a finite-state transducer, can be calculated by computing, in the usual way, the product of the automaton and the transducer, and then projecting on the second component in the alphabet of the transducer. This approach is proposed and described in more detail in [KMM+97].

A limitation of the above approach is that it can only explore state spaces of finite depth[1]. After k iterations, one can only explore configurations at a

[1] the depth of a state space is the maximal distance (measured in computation steps) from the set of initial states to any reachable state

distance at most k from the set of initial configurations. For instance, in the above algorithm one can only explore the effect of passing a token through a sequence of k processes. To explore the entire state-space, one must in general be able to calculate the effect of executing arbitrarily long sequences of computation steps. This can be done by augmenting verification by adding the transitive closure of an action, i.e., the effect of executing an action an arbitrary number of times. For example, the transitive closure of the first token-passing action in the previous example, is an action in which the token is passed through an arbitrary sequence of neighbors to another process. In the next section we show how to compute the transitive closure of a large class of actions.

4 Computing the Transitive Closure

In the previous section, we showed how to use actions in the symbolic reachability analysis by representing the effect of executing an action in terms of a finite-state transducer. For a subclass of actions, we will now show how to represent the effect of an unbounded number of executions in terms of a finite-state transducer. We will classify actions according to a *local depth*, which is the maximum number of rewritings of a symbol in a configuration, defined more precisely below.

As usual, let α^* denote the relation on strings, which is the reflexive and transitive closure of α. Let α^+ denote the transitive closure of α. For an action α, we say that a sequence of configurations $\gamma_0, \cdots, \gamma_m$ is a *configuration sequence* of α with active index pairs $(l_1, l'_1) \cdots (l_m, l'_m)$ iff for each p with $1 \leq p \leq m$ we have that $\alpha(\gamma_{p-1}, \gamma_p)$ holds with active index pair (l_p, l'_p).

Definition 3. *A configuration sequence $\gamma_0, \cdots, \gamma_m$ of α has* local depth *k if each position $i \geq 1$ satisfies $|\{p \ : \ l_p \leq i \leq l'_p\}| \leq k$. An action α has* local depth *k if whenever $(\gamma, \gamma') \in \alpha^*$ then there is a configuration sequence $\gamma_0, \cdots, \gamma_m$ of α with local depth k such that $\gamma = \gamma_0$ and $\gamma_m = \gamma'$.* □

For an action α with local depth k for some k, we can represent α^+ by a finite-state transducer, due to the following theorem.

Theorem 1. *Let $\alpha = \phi_L \boxed{\tau} \phi_R$ be an action with local depth k for some k. Then α^+ is a regular relation on strings, which can be represented by a finite-state transducer with no more than $7^{k+1} \cdot (|\tau| + 1)^k + |\phi_L| + |\phi_R|$ states, where $|R|$ is the number of states in the automaton representing R.*

Proof. (Sketch) Let $\alpha = \phi_L \boxed{\tau} \phi_R$ be an action as above. For each pair of configurations (γ, γ') in α^+, we can establish matrix of the form:

$$\begin{array}{ccccccccc}
t_0^1 & \xrightarrow{\langle c_1^0, c_1^1 \rangle} & t_1^1 & \xrightarrow{\langle c_2^0, c_2^1 \rangle} & t_2^1 & \cdots & t_{n-1}^1 & \xrightarrow{\langle c_n^0, c_n^1 \rangle} & t_n^1 \\
t_0^2 & \xrightarrow{\langle c_1^1, c_1^2 \rangle} & t_1^2 & \xrightarrow{\langle c_2^1, c_2^2 \rangle} & t_2^2 & \cdots & t_{n-1}^2 & \xrightarrow{\langle c_n^1, c_n^2 \rangle} & t_n^2 \\
 & & & & \vdots & & & & \\
t_0^m & \xrightarrow{\langle c_1^{m-1}, c_1^m \rangle} & t_1^m & \xrightarrow{\langle c_2^{m-1}, c_2^m \rangle} & t_2^m & \cdots & t_{n-1}^m & \xrightarrow{\langle c_n^{m-1}, c_n^m \rangle} & t_n^m
\end{array}$$

where

- t_i^j is a state in the transducer for α, for every i, j.
- t_0^j is a starting state and t_n^j is an accepting state of the transducer for α, for every j.
- $t \xrightarrow{\langle c,c' \rangle} t'$ iff the transducer for α can make a transition from t to t' on $\langle c, c' \rangle$.
- $c_i^0 = \gamma[i]$ and $c_i^m = \gamma'[i]$, for every i.

In the transducer for α, let q_L be the accepting state of ϕ_L and q_R be the starting state of ϕ_R. From the definition of contexts, it follows that each column in the above picture is either 1) a sequence of identical states in the transducer that copies ϕ_L, 2)a sequence of identical states in the transducer that copies ϕ_R, or 3) a sequence consisting of occurrences of q_L, q_R, and states inside the transducer for τ.

The main step of the proof is now to show that if we can construct a matrix as above, then we can construct a matrix in which all columns of the third form are sequences of form

$$w_0\ r_1\ w_1\ r_2\ w_2\ \cdots\ w_{l-1}\ r_l\ w_l$$

where l is at most the local depth of α, where each r_j is a state in the transducer for τ, and where each w_j is in one of the seven sets (this is where the number 7 in the theorem comes from)

$$\{\varepsilon\} \quad q_L^+ \quad q_R^+ \quad q_L^+ q_R^+ \quad q_R^+ q_L^+ \quad q_R^+ q_L^+ q_R^+ \quad q_L^+ q_R^+ q_L^+$$

This can be proven by starting from an arbitrary matrix as above and permuting its rows when some column has too many consecutive alternations of q_R and q_L until it is on the just described regular form. If the permutation of rows is done carefully, the initial and final configurations (γ and γ') are not affected.

We finally observe that the number of consecutive q_L's in a column is unimportant for the effect to the left of that column, and vice versa for the number of consecutive q_R's. By disregarding the number of repetitions of q_L's and q_R's, we get a finite number of different possible columns. Each such column will be a state of the transducer for α^+, and we build a transition relation which emulates the effect of the above matrix. □

As a corollary, we note that we can also approximate the transitive closure of an action which does not have a finite local depth. More precisely, for an action α, define the *approximation to local depth k of α^** as the set of pairs (γ, γ') such that there is a configuration sequence $\gamma_0, \cdots, \gamma_m$ of α with local depth k where $\gamma = \gamma_0$ and $\gamma_m = \gamma'$. From the construction in Theorem 1 it follows that we can compute the approximation to local depth k of the transitive closure of any action, represented as a transducer.

5 Compositions of Actions

For some algorithms, we need to add actions representing the composition of two or more actions. In combination with the transitive closure operation, this

makes it possible to compute the effect of an unbounded number of executions of a sequence, e.g. a loop, or a choice, e.g. modeling an if-statement, consisting of the actions in the composition.

Let $\alpha = \phi_L \boxed{\tau} \phi_R$ and $\alpha' = \phi'_L \boxed{\tau'} \phi'_R$ be two actions. Their *sequential composition* can be defined as

$$\phi_L \cap \phi'_L \boxed{\tau \circ \tau'} \phi_R \cap \phi'_R$$

where $\tau \circ \tau'$ is the transducer corresponding to the relational composition of the relations given by τ and τ'. Their *union* can be similarly defined as

$$\phi_L \cap \phi'_L \boxed{\tau \cup \tau'} \phi_R \cap \phi'_R$$

where $\tau \cup \tau'$ is the union of τ and τ'. We note that the intersection of two left contexts is always a left context, and similarly for right contexts.

As an isolated operation, composition does not add to the power of reachability analysis. If, however, used in combination with the transitive closure operation, i.e., using the transitive closure of composed actions, it can often give extra power to the reachability algorithm.

An important observation is that, in an action which is the result of a composition operation, it is sometimes possible to extend the left or right context by including a part of the string which is transformed by τ, if τ leaves that part unchanged. As a concrete example, if τ_1 changes the first position from a to b and τ_2 changes the first position from b to a, then the left context of the sequential composition of these two actions may be extended by an extra symbol, which then has to be a. After having the context in this way, the local depth of the action may decrease, thus giving even more power to the transitive closure operation.

6 Modeling Different Classes of Infinite-State Algorithms

In Section 2, we showed how to model a parameterized algorithm in which each computation step changes the state of 2 processes. In this section, we will show how our framework can also be applied to programs operating on unbounded FIFO channels, and to certain programs that use unbounded sequence numbers. In Section 7, we show how algorithms in these classes can be verified automatically, thanks to our transitive closure operation.

6.1 Programs Operating on Unbounded FIFO Channels

In this subsection, we outline how our framework can model and analyze protocols in which a set of finite-state processes communicate by sending messages over a set of unbounded FIFO channels. The verification of such protocols has been considered by Boigelot and Godefroid, [BG96,BGWW97], who propose to use a representation called QDDs (Queue Decision Diagrams) to represent sets

of states of such a system. QDDs are essentially automata which recognize the contents of the channels. In the paper [BG96] it is shown how to calculate the effect of exploring the acceleration of certain actions from a given set of states represented as a QDD. We will here show how transitive closure of corresponding operations can be calculated in our framework.

For this presentation, consider a system of two finite-state processes that communicate via one unbounded FIFO channel in each direction. Assume that the control state of each process belongs to a set Q of control states, and that each channel contains a sequence of messages in some finite set $\mathcal{M}$. A state of the system, where the processes are in control states q_1 and q_2, respectively, and the channels contain the sequences w_1 and w_2, respectively, can be represented by a string in the set given by the regular expression

$$q_1 \quad q_2 \quad \bot^* \; w_1 \; \bot^* \quad \| \quad \bot^* \; w_2 \; \bot^*$$

where the symbol $\bot$ represents an empty position in a channel, and $\|$ is used to separate the syntactic representations of the channels from each other. Thus, the representation of each channel is surrounded by "padding" with an arbitrary number of $\bot$ symbols. This allows each the contents of a channel to expand to the right when messages are inserted, and to shrink from the left when messages are removed. Of course, each particular representation of a system state allows only a finite number of insertions into a channel before all the $\bot$ symbols are "used up". However, since the padding can be arbitrarily long, we can capture the effect of arbitrarily long but finite sequences of insertions and removals, which is sufficient for analyzing safety properties.

An operation in which the first process changes control state from q_1 to q_1' and sends message m to the first channel, is modeled by an action which changes q_1 to q_1' and changes any sequence of form $\bot^{n_1} \; w_1 \; \bot^{n_2} \; \|$ with $n_2 \geq 1$ into $\bot^{n_1} \; w_1 \; m \; \bot^{n_2 - 1} \; \|$. If $q_1 \neq q_1'$, the action is idempotent, and it is thus not interesting to compute its transitive closure. If $q_1 = q_1'$, then we can calculate its transitive closure. We must then first represent the action with contexts as

$$q_1 \; Q \; \bot^* \; \mathcal{M}^* \; \boxed{\tau} \; \bot^* \; \| \; \bot^* \; \mathcal{M}^* \; \bot^*$$

where τ changes the one-symbol string $\bot$ into m. This action has local depth 1, and we can compute its transitive closure. The transmission of a sequence of messages to a channel, or the reception of a sequence of messages from a channel, can be represented in an analogous way.

A more challenging operation is one in which the first process in control state q_1 receives message m from the second channel and transmits it to the first. For the special case that our state representation has no padding symbols $\bot$ around the separator $\|$, we can model the operation by the action

$$q_1 \quad Q \quad \bot^* \; \mathcal{M}^* \quad \boxed{\tau} \quad \mathcal{M}^* \; \bot^*$$

where τ transforms strings of length 2 of form $\| \, m$ into $m \, \|$. This action has local depth 2, and we can therefore calculate its transitive closure. However,

this transitive closure results only in states without padding symbols $\perp$ around the separator $\|$. After applying the transitive closure, we must therefore "renormalize" the representation by a non length-preserving transformation which inserts an arbitrary amount of padding around the separator $\|$. This representation can be performed directly on the regular expression or automaton. As with the previous operation, we can generalize this treatment to operations that receive a sequence of message from one channel and transmit a sequence to another. In Section 7, we describe how the above method has been used to generate the set of reachable states of a version of the alternating-bit protocol with unbounded FIFO channels.

6.2 Programs Operating on Integers

We will also give a sketchy presentation of how systems that operate with integers, e.g., as counters or sequence numbers, can be modeled. The state of such a system can be modeled by letting the string represent the number line with the values "laid out" at the position corresponding to their value. Thus, the set of colors has a bit for each variable, which is true if the variable has the value corresponding to that position. The number line is infinite, but it suffices that we represent an arbitrary finite segment which contains the values of all variables. Thus, the predicate $x + 2 = y$ is modeled by the regular expression

$$(\neg x \ \wedge \ \neg y)^* \quad x \wedge \neg y \quad (\neg x \ \wedge \ \neg y) \quad \neg x \wedge y \quad (\neg x \ \wedge \ \neg y)^*$$

It should not be difficult to see that we can represent, e.g., incrementation of a variable by a constant under some conditions, and compute the transitive closure of such an action. In Section 7, we describe how the above method has been used to generate the set of reachable states of a version of a sliding window protocol with unbounded sequence numbers.

7 Experiments

We have implemented a special case of the method described in this paper, for actions that have a local depth of 2 and where the sequence of active index pairs in a configuration sequence is either increasing or decreasing. The implementation builds a transducer for the transitive closure of each action and converts the union of these transducers into the DFA library of MONA[KM98,HJJ+96b] which is implemented using BDDs[Bry86] to represent the transitions. Using the implementation, we have modeled and generated the set of reachable states of the following algorithms:

Parameterized Mutual Exclusion Algorithms We have analyzed idealized versions of parameterized algorithms for mutual exclusion, including Szymanski's algorithm, Burns's and Dijkstra's mutual exclusion algorithms, and the bakery and ticket algorithms by Lamport. Several of these could be handled by the limited framework in our earlier work [ABJN99].

Parameterized Distributed Algorithms As an example of a parameterized distributed algorithm operating on a ring, we have considered the termination detection of Dijkstra, Feijen, and van Gasteren [DFvG83], presented in Section 2.

Algorithms operating on unbounded FIFO channels . We have modeled and analyzed the Alternating Bit Protocol with unbounded FIFO channels. We have used the model of [AJ96].

Algorithms with unbounded sequence numbers . We have modeled and analyzed a sliding window protocol, in which the maximal sequence number is a parameter n. The sender window has size n and the receiver window size 1. We use a version where the channel from the sender to receiver has a capacity of 3 messages, and the channel from the receiver to the sender is synchronous. The length of the channels can of course be changed. However, we have not figured out how to model and analyze the case where both the channels and the sequence numbers are unbounded

In Table 7, we show for each algorithm the domains of the variables that are infinite, the number of steps required to generate the set of reachable configurations, the size of the transducer, the maximum number of states among automata generated during analysis, and the maximum number of BDD nodes among automata generated during analysis. Note that all automata are deterministic.

8 Conclusions

We have presented techniques for reachability analysis of parameterized and infinite-state systems whose state can be represented as a string over a finite alphabet. Since naive symbolic reachability analysis does not in general converge for such systems, we propose to use acceleration of actions to obtain termination. The main contribution is the definition of a notion of local depth of an action, and the construction of the transitive closure of an action with finite local depth, in the form of a finite-state transducer. We have shown that with this framework, we are able to model and verify a variety parameterized algorithms, and

Table 1. Experiments

Algorithm	Domains	Steps	Size	Max states	Max BDD
Szymanski	process id	8	26	144	3574
Dijkstra	process id	15	22	2503	81487
Bakery	process id, integers	5	10	32	163
Ticket	process id, integers	3	12	30	338
Burns	process id	5	14	111	2445
Termination detection	process id	7	29	133	1497
Alternating bit protocol	queues	15	67	4000	66149
Sliding Window: queue length 3	integers	21	45	339	4788

infinite-state systems operating on queues and integers. In comparison with our earlier work [ABJN99], we are able to cover a much broader class of systems. For instance, we can model systems of finite-state processes that communicate over unbounded FIFO channels, and perform transitive closure operations that are analogous to the meta-transitions presented by Boigelot and Godefroid [BG96], using QDDs. Our work is not more powerful, but shows how the techniques can be seen as part of a uniform framework.

Future work includes the treatment of liveness properties.

Acknowledgments We are grateful to Parosh Abdulla and Ahmed Bouajjani for fruitful collaboration and to Amir Pnueli for fruitful discussions during this work. At the presentation of our earlier work at CAV 1999, Ken McMillan posed a question about generalizing that work which we hope to have partially answered in this paper.

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} *CAV*, volume 1427 of *LNCS*, pages 305–318, 1998. 223

ABJN99. Parosh Aziz Abdulla, Ahmed Bouajjani, Bengt Jonsson, and Marcus Nilsson. Handling global conditions in parameterized system verification. In *Proc.* 11^{th} *CAV*, volume 1633 of *LNCS*, pages 134–145, 1999. 220, 222, 224, 226, 231, 233

ACD90. R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proc.* 5^{th} *LICS*, pages 414–425, Philadelphia, 1990. 220

AČJYK96. Parosh Aziz Abdulla, Karlis Čerāns, Bengt Jonsson, and Tsay Yih-Kuen. General decidability theorems for infinite-state systems. In *Proc.* 11^{th} *LICS*, pages 313–321, 1996. 222

AH89. R. Alur and T. Henzinger. A really temporal logic. In *Proc.* 30^{th} *Annual Symp. Foundations of Computer Science*, pages 164–169, 1989. 220

AJ96. Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996. 220, 222, 232

BCM+90. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proc.* 5^{th} *LICS*, 1990. 220

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} *CAV*, volume 1102 of *Lecture Notes in Computer Science*, pages 1–12. Springer Verlag, 1996. 223, 229, 230, 233

BGWW97. B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *Proc. of the Fourth International Static Analysis Symposium*, LNCS. Springer Verlag, 1997. 223, 229

BH97. A. Bouajjani and P. Habermehl. Symbolic reachability analysis of fifo-channel systems with nonregular sets of configurations. In *Proc. ICALP '97*, volume 1256 of *Lecture Notes in Computer Science*, 1997. 223

Bry86. R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Computers*, C-35(8):677–691, Aug. 1986. 231

BS95. O. Burkart and B. Steffen. Composition, decomposition, and model checking of pushdown processes. *Nordic Journal of Computing*, 2(2):89–125, 1995. 220

BW94. B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *Proc.* 6^{th} *CAV*, volume 818 of *LNCS*, pages 55–67. Springer Verlag, 1994. 223

CC77. P. Cousot and R. Cousot. Abstract interpretation: A unified model for static analysis of programs by construction or approximation of fixpoints. In *Proc.* 4^{th} *ACM Symp. on Principles of Programming Languages*, pages 238–252, 1977. 223

DFvG83. E.W. Dijkstra, W.H.J. Feijen, and A.J.M. van Gasteren. Derivation of a termination detection algorithm for distributed somputations. *Information Processing Letters*, 16(5):217–219, 1983. 224, 232

Fin94. A. Finkel. Decidability of the termination problem for completely specified protocols. *Distributed Computing*, 7(3), 1994. 220

FO97. L. Fribourg and H. Olsén. Reachability sets of parametrized rings as regular languages. In *Proc. 2nd Int. Workshop on Verification of Infinite State Systems (INFINITY'97)*, volume 9 of *Electronical Notes in Theoretical Computer Science*. Elsevier Science Publishers, July 1997. 222

GS92. S. M. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992. 220

HJJ+96a. 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 *LNCS*, 1996. 222

HJJ+96b. J.G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop, TACAS '95, LNCS 1019*, 1996. Also available through http://www.brics.dk/~klarlund/Mona/main.html. 231

KM98. N. Klarlund and A. Møller. *MONA Version 1.3 User Manual*. BRICS Notes Series NS-98-3 (2.revision), Department of Computer Science, University of Aarhus, October 1998. 231

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} *CAV*, volume 1254, pages 424–435, Haifa, Israel, 1997. Springer Verlag. 220, 221, 222, 226

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 *LNCS*, pages 183–202, Heidelberg, Germany, March 1997. Springer–Verlag. 222

Sis97. A. Prasad Sistla. Parametrized verification of linear networks using automata as invariants. In O. Grumberg, editor, *Proc.* 9^{th} *CAV*, volume 1254 of *LNCS*, pages 412–423, Haifa, Israel, 1997. Springer Verlag. 222

Sti96. C. Stirling. Decidability of bisimulation equivalence for normed pushdown processes. In *Proc. CONCUR '96,* 7^{th} *Int. Conf. on Concurrency Theory*, volume 1119 of *LNCS*, pages 217–232. Springer Verlag, 1996. 220

VW86. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc.* 1^{st} *LICS*, pages 332–344, June 1986. 226

WB98. Pierre Wolper and Bernard Boigelot. Verifying systems with infinite but regular state spaces. In *Proc. 10th CAV*, volume 1427 of *LNCS*, pages 88–97, Vancouver, July 1998. Springer Verlag. 222