Liveness and Acceleration in Parameterized Verification *

Amir Pnueli ** and Elad Shahar

Dept. of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, Israel

Abstract. The paper considers the problem of *uniform verification* of *parameterized systems* by symbolic model checking, using formulas in FS1S (a syntactic variant of the 2nd order logic WS1S) for the symbolic representation of sets of states. The technical difficulty addressed in this work is that, in many cases, standard model-checking computations fail to converge.
Using the tool TLV[P], we formulated a general approach to the *acceleration* of the transition relations, allowing an unbounded number of different processes to change their local state (or interact with their neighbor) in a single step. We demonstrate that this acceleration process solves the difficulty and enables an efficient symbolic model-checking of many parameterized systems such as mutual-exclusion and token-passing protocols for any value of N, the parameter specifying the size of the system. Most previous approaches to the uniform verification of parameterized systems, only considered *safety properties* of such systems. In this paper, we present an approach to the verification of *liveness* properties and demonstrate its application to prove accessibility properties of the considered protocols.

Keywords: Symbolic model checking; Parametric systems; Acceleration; Liveness; Regular expressions; WS1S

1 Introduction

The problem of uniform verification of parameterized systems is one of the most thoroughly researched problems in computer-aided verification. The problem seems particularly elusive in the case of systems that consist of regularly connected finite-state processes (a process network). Such a system can be model checked for any given configuration, but this does not provide a conclusive evidence for the question of *uniform verification*, i.e., showing that the system is correct for *all* possible configurations.

* This research was supported in part by the Minerva Center for Verification of Reactive Systems, a gift from Intel, and a grant from the U.S.-Israel bi-national science foundation.

** Corresponding author. E-mail: Amir@wisdom.weizmann.ac.il

In [KMM+97], we proposed an approach to the uniform verification of parameterized systems based on symbolic model checking in which the assertional language used to represent sets of reachable global states is that of a regular expressions over a finite alphabet which represents the local state of each of the processes in the system. As a trivial illustrative example, consider a parameterized system $S(N)$ consisting of N processes arranged in a linear array. Assume that the local state of each process can be represented by the two values 0 and 1, where the state of a process $P[i]$ is 1 iff $P[i]$ currently has the token which is passed around.

The initial global state (to which we refer as a *configuration*) can be described by the regular expression $I = 10^*$ representing the global state in which the leftmost process has the token. Note that even though every instance of the system $S(N)$ has a unique initial configuration, the expression 10^* represents the infinite set of initial configurations obtained by considering the infinitely many different values of N.

The transition relation of this parameterized system can be represented by the binary rewrite rule given by $10 \quad \rightarrow \quad 01$. This rewrite rules states that a single step of the system applied to a configuration represented by a word w may locate the substring 10 within w and replace it by the substring 01. Obviously, such a step represents the transmission of a token from a process with a token to its right neighbor, provided the neighbor is not currently in possession of a token.

To represent such rewrite rules in the most general context, [KMM+97] suggested to use a finite-state *transducer* which is an automaton reading a string of pairs of letters, one representing the pre-transition configuration and the other representing the post-transition configuration. Using the standard notation of unprimed and primed values to respectively represent these two configurations, the transducer corresponding to the above transmission transition can again be represented by the following regular expression:

$$T \quad = \quad (00' + 11')^* \, (10') \, (01') \, (00' + 11')^*$$

Given a finite-state transducer T representing the transition relation and a regular expression E representing a set of configurations, it is not difficult to compute the set of T-postimages or T-preimages of the configurations in E which is guaranteed to be another regular expression. For example the T-postimage of 10^* is the regular expression 010^*. We denote by $E \diamond T$ and $T \diamond E$ the T-postimages (T-successor) and T-preimages (T-predecessor) of E, respectively.

To perform symbolic model checking we usually need the iterated versions of these two operators computed as follows:

$$E \diamond T^* = E \; + \; E \diamond T \; + \; (E \diamond T) \diamond T \; + \; ((E \diamond T) \diamond T) \diamond T \; + \; \cdots$$
$$T^* \diamond E = E \; + \; T \diamond E \; + \; T \diamond (T \diamond E) \; + \; T \diamond (T \diamond (T \diamond E)) \; + \; \cdots$$

Now, if φ is a regular expression representing a property we wish to prove an invariant of the system, then $S(N) \models \varphi$ for every N iff

$$(I \diamond T^*) \cap \overline{\varphi} = \emptyset \qquad \text{or} \qquad (T^* \diamond \overline{\varphi}) \cap I = \emptyset,$$

where $\overline{\varphi}$ denotes the complement of φ. The first clause corresponds to *forward exploration* starting from the initial condition I while the second clause corresponds to *backwards exploration* starting from the set of states violating the property φ.

The difficulty specific to regular model checking of parameterized systems is that, unlike BDD-based model checking of finite-state systems, the computation of either $I \diamond T^*$ or $T^* \diamond \overline{\varphi}$ may fail to terminate. In fact, theoretical considerations predict that there will be cases in which these computations cannot terminate. Termination of the computation of $I \diamond T^*$ implies that the set of strings encoding reachable configurations is a regular language, and it is easy to construct systems in which the set of reachable configurations forms a context-free language.

However, experience with these methods shows that there are many cases in which the set of reachable configurations is regular yet the straightforward computation of $I \diamond T^*$ fails to converge. Assume that we wish to establish for the above example system the invariance of the property $\varphi = 0^*10^*$, claiming that all reachable configurations contains precisely one token. To apply backwards exploration, we first compute the set of violating configurations, given by $\overline{\varphi} = 0^* + (0+1)^*1(0+1)^*1(0+1)^*$. The computation of $T^* \diamond \overline{\varphi}$ terminates in one step, yielding $T^* \diamond \overline{\varphi} = \overline{\varphi} = 0^* + (0+1)^*1(0+1)^*1(0+1)^*$ which, obviously, has an empty intersection with $I = 10^*$, establishing that φ is an invariant of the considered system.

On the other hand, the computation of the forward exploration according to $I \diamond T^*$ fails to terminate, yielding the following infinite sequence of approximations:

$$10^* + 010^* + 0010^* + 00010^* + \cdots$$

The source of the problem was identified in [ABJN99] as stemming from the fact that the transition relation T represents a step in which only one process (or a pair of contiguous processes) makes a move. The remedy proposed by this paper is to use the notion of an *accelerated transition* in which several (unbounded many) processes can make a move at the same step. For example, the accelerated version of the transition relation $T = (00' + 11')^* (10') (01') (00' + 11')^*$ can be computed to be

$$T_a \quad = \quad (00' + 11')^* (10') (00')^* (01') (00' + 11')^*$$

Applying the accelerated transition in a forward exploration terminates now in a single step and yields $I \diamond T_a^* = 0^*10^*$.

The work in [ABJN99] proposes a "speed-up" (acceleration) operation which transforms a single-process transition relation presented by a transducer T into an accelerated transducer T_a which represents the effect of many processors taking an action in the same step, under certain conditions restricting the dependency of a single-process action on the local states of the other processes. The analysis there is based on a language-theoretic representation of the assertions and the representation of transition relations by finite-state transducers. Using such acceleration techniques, [ABJN99] managed to verify fully automatically various parameterized protocols such as the Bakery and Ticket algorithms

by Lamport, Burn's protocol, Dijkstra's and Szymanski's algorithms for mutual exclusion.

The methods of [ABJN99] could only accelerate elementary transitions which only modified the local state of one process at a time. This made them inapplicable to the representation of systems which included synchronous message passing, such as the binary transformation $10 \rightarrow 01$ appearing in our example system. This drawback has been recently corrected in [JN00] which presents a speed-up operation which can be applied to elementary transitions that modify several contiguous processes at the same time.

The work presented here improves upon the results of [ABJN99] and [JN00] in several directions. To start with, our presentation framework uses the logic FS1S (a syntactic variant of WS1S, the weak second-order monadic theory of one successor [Tho90]) to present sets of configurations, e.g. the initial condition and the properties, as well as the transition relation. This uniform presentation by a powerful logic enables us to formulate several acceleration schemes still within the same language. Furthermore, the soundness of the transducer-based acceleration schemes of [ABJN99] and [JN00] depends on particular assumptions that the transition relation has to satisfy, such as a particular forms of left- and right-contexts, These have to be checked whenever one wants to apply the acceleration schemes of [ABJN99] and [JN00] to a particular transition relation. In our case, the acceleration is always sound and could never lead to false positive. In the worse case, they will not produce a useful acceleration and the process will continue to diverge even after the acceleration.

Using our acceleration schemes which are applicable to unary and binary elementary transitions in an unrestricted way, we managed to verify the protocols considered in [ABJN99] in a very efficient manner, and consider some additional protocols which use synchronous communication, such as a token-passing protocol for mutual exclusion and the distributed termination detection algorithm of [DFvG83].

However, the most important contribution of this paper is the extension of the regular model checking method to include verification of *liveness* properties, while all previous efforts concentrated on the parameterized verification of safety properties. Using these extensions, we managed to verify the property of accessibility for some of the protocols considered above.

Related Work

There are several results on algorithmic verification of parameterized systems [SG92,AJ98,CGJ95]. In most of these works the transitions are guarded by local conditions involving the local states of a fixed (unparameterized) number of processes, in contrast with the general global dependency which is allowed in [KMM+97,ABJN99,JN00]. The notions of speed-ups and acceleration of transitions were considered in [BG96,BGWW97,BH97,ABJ98]. However, the accelerations considered there only condensed several moves of a fixed number of processes, while in our case (and in [ABJN99,JN00]) we consider speed-ups obtained by performing actions of an unbounded number of different processes, sequentially or in parallel. Previous attempts to verify parameterized protocols such as

Burn's protocol [JL98] and Szymanski's algorithm [GZ98,MAB+94,MP90] relied on abstraction functions or lemmas provided by the user. Other approaches to uniform parameterized verification are based on induction, where the user supplies the induction hypothesis either in the form of an assertion or in the form of a network invariant [CGJ95,KM89,WL89].

A recent work which has a significant overlap with our work has been presented by Bodeveix and Filali in [BF00]. Similarly to our approach, they advantageously employ the expressive power of WS1S to present explicit formulas which capture various acceleration schemes. They report about a tool FMONA which is a high-level macro-processor for MONA [HJJ+96]. The main differences between their work and ours are that, at this point, they do not consider liveness. Also, on the technical level, unlike the TLV[P] tool which we use for the verification reported in this paper, the FMONA tool does not seem to support a programming layer in which algorithms such as model-checking for safety and liveness can be programmed. As a result, if one wants to iterate the application of a transition relation to a set of states until it converges, it is necessary to provide an a priori bound n on how many iterations are necessary and to invoke the FMONA macro processor which will expand the appropriate iteration into a pure MONA code of size linear in n.

2 The Logic FS1S

We use the logic FS1S, (*finitary second-order theory of one successor*) as a specification language for sets of global states of parameterized systems. This logic is derived from the *weak second order logic of one successor* [Tho90] and also resembles the language M2L used in MONA [HJJ+96]. The main difference between WS1S and FS1S is that, in FS1S, we assume the existence of a special variable M which provides an upper bound to the size of all arrays. We found the use of this common upper bound to be of much help in the description of circular architectures such as rings. This is only a matter of convenience, because, it is always possible to introduce M as a second-order variable of WS1S and postulate its upper-bound properties.

It is well known that FS1S (as well as WS1S) has the expressive power of regular expressions, as well as finite automata which are the representation underlying our implementation. Following is a brief definition of the logic.

Syntax

We assume a *signature* $\Xi : \{\Sigma_1, \ldots, \Sigma_k\}$ consisting of a finite set of finite alphabets. The *vocabulary* consists of *position variables* $p_1, p_2, \ldots$ and, for each $\Sigma_i \in \Xi$, a set of Σ_i*-array variables* $X_i, Y_i, Z_i, \ldots$. The special position variable M denotes the upper bound on the length of all arrays and all position variables.

- Position (First-order) terms:
  The constant 1 and any position variable p_i are position terms. If t is a position term then so is $t + 1$.
- Letter terms:

- Every $a \in \Sigma_i$ is a Σ_i-term.
- If X is a Σ_i-array variable and t is a position term, then $X[t]$ is a Σ_i-term.

- Atomic Formulas:
  - $t_1 \sim t_2$, where t_1 and t_2 are position terms and $\sim \in \{=, <\}$.
  - $x = y$, where x and y are Σ_i-terms for some $\Sigma_i \in \Xi$.
- Formulas:
  - An atomic formula is a formula.
  - Let φ and ψ be formulas. Then $\neg\varphi$, $\varphi \vee \psi$, $\exists p : \varphi$, $\exists X : \varphi$ are formulas, where p is a position variable and X is an array variable.

For example, assume that Π is an array over the alphabet $\Sigma_1 = \{N, T, C\}$ intended to represent the control location of a process in a process-array $P[1], \ldots, P[M]$. Similarly, assume that *tok* is a Boolean array (special case of $\Sigma_2 = \{0, 1\}$) intended to represent the fact that process $P[i]$ currently has the token. Then, the WS1S-formula

$$\Theta : \qquad \forall i : (\Pi[i] = N) \ \wedge \ tok[1] \ \wedge \ \forall j \neq 1 : \neg tok[j]$$

characterizes the set of initial configurations in which all processes are in their initial control location N and only the leftmost process (process $P[1]$) has the token.

We refer the reader to [KMM+97] for the definition of the semantics of FS1S.

3 The Logic FS1S is Adequate

In this section we demonstrate the use of FS1S for expressing the constituents of a parameterized system. As a running example, we will use program MUX of Fig. 1 which implements mutual exclusion by synchronous communication.

The body of the program is a variable-size parallel composition of processes $P[1], \ldots, P[M]$. Each process $P[i]$ has two local state variables: a local boolean variable *tok* whose initial value is 1 (*true*) for $i = 1$ and 0 (*false*) for all other processes, and a control variable Π ranging over the set of locations $\{N, T, C\}$ (the noncritical section, the trying section, and the critical section, respectively). Process $P[i]$ sends the boolean value 1 on channel $\alpha[i \oplus_M 1]$ to its right neighbor ($i \oplus_M 1$ is addition modulo M) and reads into variable *tok* a (true) boolean value from its left neighbor on channel $\alpha[i]$. As seen in the program, process $P[i]$ can enter its critical section only if $P[i].tok = 1$.

As our computational model we use the model of *fair discrete systems* consisting of a set X of *state variables*, an *initial condition* Θ, a *transition relation* ρ, a set $\mathcal{J}$ of *justice* (*weak fairness*) requirements, and a set $\mathcal{C}$ of *compassion* (*strong fairness*) requirements. We proceed to show how these constituents can be specified in FS1S for system MUX .

The State Variables: We define the type

$$state = \textbf{record of } \langle \Pi : \{N, T, C\}, tok : \textbf{boolean} \rangle$$

and the array variable

$$X : \ \textbf{array } 1..M \textbf{ of } state.$$

Note that this is equivalent to the definition of two arrays, the array $\Pi[1..M]$ and the Boolean array $tok[1..M]$. Therefore, we will often abbreviate $X[i].\Pi$ and

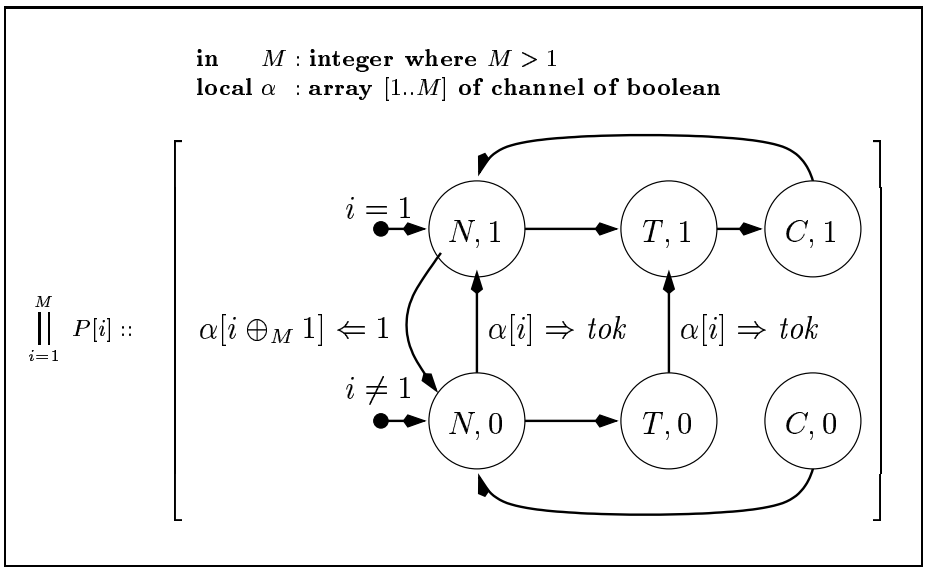

Fig. 1. Parameterized Program MUX.

$X[j].tok$ to $\Pi[i]$ and $tok[j]$ respectively. On the other hand, we write $X[i] = X[j]$ as an abbreviation for $(X[i].\Pi = X[j].\Pi) \ \wedge \ (X[i].tok = X[j].tok)$ and $\exists X : \varphi$ as an abbreviation for $\exists X.\Pi : \exists X.tok : \varphi$.

The Initial Condition: The initial condition can be given by the FS1S formula

$$\Theta : \quad (\forall i : \Pi[i] = N) \ \wedge \ tok[1] \ \wedge \ \forall i \neq 1 : \neg tok[i]$$

The Transition Relation: The transition relation can be formed as the disjunction of three types of elementary transitions. Using the abbreviation $presX(j) = X'[j] = X[j]$, these can be expressed as follows:

$$idle : \forall j : presX(j)$$

$$\rho_1(X, X', i) : idle \ \vee \ (\forall j \neq i : presX(j)) \ \wedge \begin{pmatrix} (\Pi[i] = N) \ \wedge \ (\Pi'[i] = T) \ \wedge \ (tok'[i] = tok[i]) \\ \vee \ (\Pi[i] = C) \ \wedge \ (\Pi'[i] = N) \ \wedge \ (tok'[i] = tok[i]) \\ \vee \ (\Pi[i] = T) \ \wedge \ (\Pi'[i] = C) \ \wedge \ (tok'[i] = tok[i] = 1) \end{pmatrix}$$

$$\rho_2(X, X', i) : idle \ \vee \ (\forall j \notin \{i, i \oplus_M 1\} : presX(j)) \ \wedge \ (\Pi[i] = N) \ \wedge \ tok[i] \ \wedge \ (\Pi[i \oplus_M 1] \in \{N, T\}) \ \wedge \ \neg tok[i \oplus_M 1] \ \wedge \ (\Pi'[i] = N) \ \wedge \ \neg tok'[i] \ \wedge \ (\Pi'[i \oplus_M 1] = \Pi[i \oplus_M 1]) \ \wedge \ tok'[i \oplus_M 1]$$

Subtransition *idle* represents the case that the system does not change it's state. Subtransition $\rho_1(X, X', i)$ is a *unary transition* in which a single process $P[i]$ takes a local action that can only modify the local state of $P[i]$. All other processes retain their local state. Finally, subtransition $\rho_1(X, X', i)$ corresponds to

a *binary transition* in which process $P[i]$ sends the token to process $P[i \oplus_M 1]$. Only the two involved processes change their local states.

We can now define the global transition relation by taking

$$\rho(X, X') = idle \ \vee \ (\exists i : \rho_1(X, X', i) \ \vee \ \rho_2(X, X', i)).$$

However, as explained in the introduction, this single-action transition can be used in few cases for backwards exploration model checking but will often fail to converge when used in a forward exploration model checking.

We defer the specification of the justice and compassion requirements of system MUX to Section 5 in which we discuss the verification of liveness properties, where the fairness requirements become relevant.

3.1 Model Checking

Having obtained the FS1S representation of the transition relation $\rho(X, X')$ of a system such as MUX, there are several symbolic model checking tasks we can perform. For an FS1S formula $\varphi(X)$ representing a set of configurations, we can compute the ρ-successor and ρ-predecessor of φ by the following expressions:

$$\varphi \diamond \rho = unprime(\exists X : \varphi(X) \ \wedge \ \rho(X, X'))$$
$$\rho \diamond \varphi = \exists V : \rho(X, V) \ \wedge \ \varphi(V),$$

where *unprime* is a substitution operation which transforms each occurrence of $X'[k]$ into $X[k]$, and V is an auxiliary array variable of type *state*.

Note that $\rho \diamond \varphi$ computes the set of states satisfying $\mathbf{EX}\varphi$ from which, by iteration and boolean operations, we can compute $\mathbf{EF}\varphi$ and $\mathbf{AG}\varphi$, provided the iteration converges.

4 Acceleration

Acceleration condenses a potentially unbounded number of applications of transitions into a single transition, by defining a single "accelerated transition relation". It is up to the user to observe that acceleration is required and select the appropriate accelerations schemes to be applied. Since all accelerations are sound, there is no danger (except loss of time) in applying all the acceleration schemes which are available at a particular implementation.

Since the verification problem for parameterized system is, in general, undecidable [AK86], there is no chance of accumulating a "complete" set of acceleration schemes. The best we can hope for is the assembly of a large set of schemes which can cover many of the useful examples.

To handle most of the cases in which regular model checking with single-action transition relation failed to terminate, we found it necessary to consider three types of acceleration which we will now present.

4.1 Local Acceleration

In this mode of acceleration, we allow several actions to be taken in succession by the same process $P[i]$. Given a unary transition relation $\rho_1(X, X', i)$, we can compute its locally accelerated version by the repeated composition

$$\rho_1^\alpha \quad = \quad \rho_1 \ \vee \ \rho_1 \diamond \rho_1 \ \vee \ (\rho_1 \diamond \rho_1) \diamond \rho_1 \ \vee \ ((\rho_1 \diamond \rho_1) \diamond \rho_1) \diamond \rho_1 \ \vee \ \cdots,$$

where the composition $\rho_a \diamond \rho_b$ is defined by

$$\rho_{a;b}(X, X', i) \quad = \quad (\exists V : \rho_a(X, V, i) \ \wedge \ \rho_b(V, X', i)).$$

For example, applying local acceleration to the unary transition relation $\rho_1(X, X', i)$ of program MUX, we obtain (after some manual simplification) the following accelerated unary transition:

$$\rho_1^\alpha(X, X', i) \quad = \quad \left(\begin{array}{l} \quad \Pi[i] \in \{N, T, C\} \ \wedge \ tok[i]' = tok[i] \\ \wedge \ \forall j \neq i : (\Pi'[j] = \Pi[j] \ \wedge \ tok'[j] = tok[j]) \\ \wedge \ \left(\begin{array}{l} \quad \Pi[i]' = \Pi[i] \\ \vee \ tok[i] = 0 \ \wedge \ \Pi'[i] \in \{N, T\} \\ \vee \ tok[i] = 1 \ \wedge \ \Pi'[i] \in \{N, T, C\}) \end{array} \right) \end{array} \right)$$

4.2 Global Acceleration of Unary Transitions

Next, we consider the acceleration of a unary transition on which each of a set of processes takes a single action. Assume as before that the unary transition relation of process $P[i]$ is given by $\rho_1(X, X', i)$, and that $idle \rightarrow \rho_1$. The following formula expressing this acceleration uses the auxiliary *state*-array variables T and V.

$$\rho_1^g(X, X') \quad = \forall i \left(\begin{array}{l} \quad X'[i] = X[i] \\ \vee \ \exists T, V \left[\begin{array}{l} \forall j \left[\begin{array}{l} T[j] = \textbf{if } j < i \textbf{ then } X'[j] \textbf{ else } X[j] \\ V[j] = \textbf{if } j \leq i \textbf{ then } X'[j] \textbf{ else } X[j] \end{array} \right] \\ \wedge \ \rho_1(T, V, i) \end{array} \right] \end{array} \right)$$

This accelerated transition applies $\rho_1(X, X', i)$ to processes $P[1], \ldots, P[M]$ in sequential order. Every activated process $P[i]$ may non-deterministically choose to idle (which is one of the options allowed by ρ_1) or change its local state according to ρ_1. For process $P[i]$ we require that, after all processes $P[1], P[2], \ldots, P[i-1]$ have taken their actions, we reach a configuration from which $P[i]$ can take its action. This is done by forming the two arrays T and V. where V represents the configuration prior to $P[i]$'s action and V represents the configuration resulting from $P[i]$'s action.

For example, applying global acceleration to the accelerated unary transition relation $\rho_1^\alpha(X, X', i)$ of program MUX, we obtain (after some manual simplification to improve readability) the following accelerated unary transition:

$$\rho_1^g(X, X') \quad = \quad \forall i \left(\begin{array}{l} \quad \Pi[i] \in \{N, T, C\} \ \wedge \ tok[i]' = tok[i] \\ \wedge \ \left(\begin{array}{l} \quad \Pi[i]' = \Pi[i] \\ \vee \ tok[i] = 0 \ \wedge \ \Pi'[i] \in \{N, T\} \\ \vee \ tok[i] = 1 \ \wedge \ \Pi'[i] \in \{N, T, C\}) \end{array} \right) \end{array} \right)$$

Note that the acceleration scheme presented here proceeds from left to right. It is straightforward to define an acceleration scheme which proceeds from right to left.

4.3 Global Acceleration of Binary Transitions

Finally, let us consider the acceleration of a binary transition, such as $\rho_2(X, X', i)$ previously presented for program MUX.

Unlike the acceleration of unary transitions, where the local state of each process changed at most once, in the case of binary acceleration some processes may change their local state twice. For example, they may change their state once when they receive the token from their left neighbor and then once more when they send the token to their right neighbor. Thus the acceleration of a binary token-passing transition may in one step move the token from process $P[i]$ to process $P[j]$ for an arbitrary $j > i$. To accommodate the phenomenon that some processes may change their values twice, we employ an additional *state*-array W to save the sequence of intermediate local states for these processes.

Let $\rho_2(X, X', i)$ be a binary transition which may affect at most the components $X[i]$ and $X[i \oplus_M 1]$. Without loss of generality, assume that $idle \rightarrow \rho_2$. The formula $\rho_2^g(X, X')$, expressing the global acceleration of $\rho_2(X, X', i)$ is given by

$$
\exists W \left(\begin{array}{l} W[1] = X[1] \ \wedge \ W[M] = X'[M] \\ \wedge \ \forall i < M \, \exists T, V \left(\begin{array}{l} \forall j \left(\begin{array}{l} T[j] = \left[\begin{array}{ll} \textbf{case} & \\ & j = i : W[j] \\ & j < i : X'[j] \\ & 1 \quad\ : X[j] \\ \textbf{esac} & \end{array} \right] \\ \wedge \\ V[j] = \left[\begin{array}{ll} \textbf{case} & \\ & j = i+1 : W[j] \\ & j \leq i \quad : X'[j] \\ & 1 \quad\ : X[j] \\ \textbf{esac} & \end{array} \right] \end{array} \right) \\ \wedge \ \rho_2(T, V, i) \end{array} \right) \end{array} \right)
$$

As we can see, in the binary acceleration case, we sequentially apply the binary transition ρ_2 to processes $P[1], \ldots P[M-1]$, where any of them may nondeterministically choose to take the idling transition. In the general case, each of the processes $P[2], \ldots, P[M-1]$ may change their local states at most twice, while processes $P[1]$ and $P[M]$ may change their local state at most once. We use the auxiliary array W to store the intermediate value of the local state of all processes.

Note that this acceleration scheme does not apply ρ_2 to process $P[M]$. When we compute the total transition relation we add $\rho_2(X, X', M)$ as an additional explicit disjunct.

These acceleration schemes were successfully applied to program MUX and transformed the regular model checking procedure based on forward exploration from a divergent process into an efficiently convergent one, requiring no more than 4 iterations to converge in a matter of few seconds. More details about these computations are presented in Section 6.

5 Liveness

All of the previous results for the uniform algorithmic verification of parameterized systems concentrated on proofs of safety properties. Here we present an approach to the verification of liveness properties, using regular symbolic model checking. The main problem with parameterized verification of liveness properties is not so much that the property to be proven is more complex, but that we have to take into account an unbounded number of fairness assumptions, several for each process, and that these requirements are also parameterized. To appreciate the problem, let us specify the fairness requirements associated with program MUX which, for the sake of simplicity of presentation, we restricted to justice (weak fairness) only.

5.1 Justice Requirements for Program MUX

There are three justice requirements associated with each process of program MUX. Respectively, they require that the process will never get stuck at location C, that it will never get stuck at location T while the process has the token, and that the process will not retain the token forever while it's right neighbor is continuously ready to receive it. In the computational model of *fair discrete systems*, justice requirements are presented as a set of assertions $\mathcal{J} = \{J_1, \ldots, J_k\}$, with the requirement that a computation should infinitely often visit states satisfying J_j for each $j = 1, \ldots, k$. In the parameterized case, each justice requirement is also parameterized by a process index i, and the requirement should be extended to cover all $i \in [1..M]$.

For program MUX, the justice requirements are given by

$$J_1[i] : \neg(\Pi[i] = C)$$
$$J_2[i] : \neg((\Pi[i] = T) \ \wedge \ tok[i])$$
$$J_3[i] : \neg(tok[i] \ \wedge \ (\Pi[i \oplus_M 1] \in \{N, T\}))$$

In theory, one may try to verify a liveness property "every p is eventually followed by q" of a parameterized system using the standard symbolic model-checking algorithm. The core of this algorithm is the computation of the set of states lying on a fair $\neg q$-path. This computation can be succinctly described by the following fix-point formula:

$$\mathbf{E}_f\mathbf{G}\neg q \quad = \quad \nu Y(\neg q \ \wedge \ \rho \diamond Y \ \wedge \ \forall i : (\bigwedge_j((\rho \wedge \neg q)^* \diamond (Y \wedge J_j[i]))))$$

Unfortunately, this computation seldom converges, even if we use an accelerated version of the transition relation. This is certainly the case for program MUX.

5.2 Detecting Bad Cycles

Since the systems we analyze are finite-state (for every value of their parameter), it is obvious that the formula $\mathbf{E}_f\mathbf{G}\neg q$ characterizes the states from which there exists a $(\neg q)$-path leading to a fair $(\neg q)$-cycle, where the cycle being fair means that it visits at least once a J_j-state, for each $j = 1, \ldots, k$. Denoting by G the set of states that participate in a fair $(\neg q)$-cycle, an equivalent requirement is that each $s \in G$ has a successor in G and that, for each $s \in G$ and each $j = 1, \ldots, k$, there exists a cycle from s to itself which visits on the way some J_j-state.

Assume that $\rho(X, X')$ represents the total transition relation of the parameterized system, after all accelerations. The following algorithm computes the set of states participating in a fair $(\neg q)$-cycle:

1. $\varphi_1 := (\Theta \diamond \rho^*) \ \wedge \ \neg q$
2. $\rho_1 := \rho \ \wedge \ \varphi_1 \ \wedge \ (\forall i : U'[i] = U[i])$
3. $\varphi_2 := \varphi_1 \ \wedge \ (\forall i : U[i] = X[i])$
4. $\varphi_3 := \varphi_2 \ \wedge \ \rho_1^* \diamond (\rho_1 \diamond \varphi_2)$
5. **for** $j := 1, \ldots, k$ **do**
6. $\quad \varphi_3 := \varphi_3 \ \wedge \ (\forall i : \rho_1^* \diamond (J_j[i] \ \wedge \ \varphi_1 \ \wedge \ (\rho_1^* \diamond \varphi_2)))$
7. $\varphi_4 := (\exists U : \varphi_3)$

Line 1 places in φ_1 the set of $(\neg q)$-states which are reachable. Line 2 places in ρ_1 a version of ρ restricted to move only within φ_1-states and to preserve a set of variables called U, which is a newly introduced copy of the state variables. Line 3 adds to the sets of states the interpretation of the auxiliary variables U and places in φ_2 the subset of φ_1-states in which the interpretations of X and U agree. Line 4 places in φ_3 the set of φ_2-states from which there exists a non-empty ρ_1-path leading to another φ_2-state.

To see this, consider a state s_1 belonging to φ_3, and let s_2 be the φ_2-state reached at the end of the non-empty ρ_1-path. Since s_2 is a φ_2-state, we know that $s_2[U] = s_2[X]$, i.e. the interpretation of U in s_2 is identical to the interpretation of X in s_2. Since any ρ_1-path preserves the interpretation of U, we also have that s_1 and s_2 agree on the interpretation of U, i.e., $s_1[U] = s_2[U]$. Since s_1 is also a φ_2-state, it follows that $s_1[X] = s_1[U]$. Consequently, we have that $s_1[X] = s_1[U] = s_2[X] = s_2[U]$ which implies that s_1 and s_2 are identical states and, therefore, s_1 participates in a non-empty φ_1-cycle.

Following a similar argument, the iterations at line 6 retain at φ_3 only the φ_2-states which reside on a cycle containing a $J_j[i]$ state for each j and each i. The cycles may be different for different values of j and i, but they can always be combined into a very big cycle which contains them all, and may revisit the originating state many times.

It follows that, when (and if) the algorithm terminates, φ_4 contains the states which reside on a non-empty fair $(\neg q)$-cycle.

The algorithm presented above can, in principle, be used also for conventional (non-parametric) symbolic model checking of liveness properties. However it is not advisable to do so, because the algorithm is highly inefficient in the con-

ventional context due to the introduction of the auxiliary copy U of the state variables.

Normally, assertions of states and transitions relations are specified as having the types $\varphi : V \rightarrow \{0,1\}$ and $\rho : V \times V' \rightarrow \{0,1\}$. When adding an additional copy of the state variables we obtain assertions: $\varphi : V \times U \rightarrow \{0,1\}$ and $\rho : V \times U \times V' \times U' \rightarrow \{0,1\}$.

Note that all the work on acceleration actually computes $\rho \diamond \rho$ separately from its application to any assertion φ. This kind of computation is usually avoided whenever possible. For example, in symbolic backwards exploration, it is more efficient to compute $\rho \diamond (\rho \diamond \varphi)$ rather than $(\rho \diamond \rho) \diamond \varphi$.

For these reasons the additional copy of the state variables excises a heavy penalty, as is evident from the performance figures presented in Table 1 of Section 6. However, in the parameterized context, this is the only fully automatic algorithm we managed to successfully use for the verification of liveness properties.

5.3 Liveness Using Pseudo Cycles

Realizing the heavy price one has to pay for a full second copy of the state variables, we developed another approach which replaces the notion of a cycle by a *pseudo cycle*. Assume that the set of reachable states is partitioned by a partition Π into a set of disjoint classes. A pseudo-cycle, relative to the partition Π, is a path which begins and ends in two states belonging to the same class. Note, that when the partition Π is the finest possible, that is, each class containing a single state, then the notions of a pseudo-cycle and a cycle coincide.

To use this approach, the user has to provide a parameterized assertion $E(i)$, which defines the partition, consisting of a class for each value of i. The pseudo-cycle method is guaranteed to be sound but may produce false negatives due to its approximative nature.

The following is the improved algorithm for finding fair $(\neg q)$-pseudo-cycles:

1. $\varphi_1 := (\Theta \diamond \rho^*) \ \wedge \ \neg q$
2. $\rho_1 := \rho \ \wedge \ \varphi_1$
3. $\varphi_2 := \varphi_1 \ \wedge \ E(i)$
4. $\varphi_3 := \varphi_2 \ \wedge \ \rho_1^* \diamond (\rho_1 \diamond \varphi_2)$
5. **for** $j := 1, \ldots, k$ **do**
6. $\quad \varphi_3 := \varphi_3 \ \wedge \ (\forall i : \rho_1^* \diamond (J_j[i] \ \wedge \ \varphi_1 \ \wedge \ (\rho_1^* \diamond \varphi_2)))$

Let $E(i)$ be an assertion such that $\varphi_1 \rightarrow E(i)$. $E(i)$ should be such that it partitions the space of $(\neg q)$-reachable states. This partition corresponds to the set of state classes we use in order to find pseudo cycles.

The improved algorithm is similar to the original one, except for lines 2,3, in which we omitted the references to U, and line 7 which is omitted entirely. Instead, line 3 includes a conjunct of $E(i)$. The original constraint on line 3, $(\forall i : U[i] = X[i])$, uses U to form the finest partition, where each partition class contains only a single state.

It is clear that if there exists a real fair $\neg q$-cycle, then the improved algorithm will find it. Therefore, if the algorithm declares that there are no bad pseudo-cycles, this implies that, in particular there are no bad cycles, which establishes the soundness of the algorithm when it is used to deduce the absence of any bad cycles.

6 Results

In table 1, we present the results of our regular uniform verification applied to several well-known algorithms. The results do not include the computations of the accelerated transitions. It is obvious that the verification of safety properties is significantly more efficient than the verification of liveness properties.

Algorithm	Safety		Liveness		Improved Liveness	
	Time	Iterations	Time	Iterations	Time	Iterations
Token ring	0.4	3	53	40	9.2	32
Szymanski	0.2	8	–	–	–	–
Termination detection	5.6	9	–	–	–	–
Dining philosophers	0.6	3	–	–	–	–

Table 1. Experimental results (times in seconds)

7 Conclusions

In this paper we presented several significant extensions to the state-of-the art in uniform verification of parameterized systems. We demonstrated the expressive power of the logic FS1S as an efficient vehicle for expressing both the system constituents as well as the meta-operations of acceleration. We presented several acceleration schemes that lead to a very efficient regular model checking of safety parameterized properties.

Finally, we presented the first approach to the uniform verification of liveness properties of parameterized systems using the FS1S framework and the TLV[P] tool.

References

[ABJ98] P.A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In A.J. Hu and M.Y. Vardi, editors, *Proc. 10^{th} Intl. Conference on Computer Aided Verification (CAV'98)*, volume 1427 of *Lect. Notes in Comp. Sci.*, pages 305–318. Springer-Verlag, 1998.

[ABJN99] P.A. Abdulla, A. Bouajjani, B. Jonsson, and M. Nilsson. Handling global conditions in parametrized system verification. In N. Halbwachs and D. Peled, editors, *Proc. 11st Intl. Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *Lect. Notes in Comp. Sci.*, pages 134–145. Springer-Verlag, 1999.

[AJ98] P.A. Abdulla and B. Jonsson. Verifying networks of timed processes. In B. Steffen, editor, *4th Intl. Conf. TACAS'98*, volume 1384 of *Lect. Notes in Comp. Sci.*, pages 298–312. Springer-Verlag, 1998.

[AK86] K. R. Apt and D. Kozen. Limits for automatic program verification of finite-state concurrent systems. *Information Processing Letters*, 22(6), 1986.

[BF00] J-P. Bodeveix and M. Filali. Experimenting acceleration methods for the validation of infinite state systems. In *International Workshop on Distributed System Validation and Verification (DSVV'2000)*, Taipei, Taiwan, April 2000.

[BG96] B. Boigelot and P. Godefroid. Symbolic verification of coomunication protocols with infinite state spaces using QDDs. In R. Alur and T. Henzinger, editors, *Proc. 8^{th} Intl. Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *Lect. Notes in Comp. Sci.*, 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*, Lect. Notes in Comp. Sci. Springer-Verlag, 1997.

[BH97] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with non-regular sets of configurations. In *Proc. 24th Int. Colloq. Aut. Lang. Prog.*, volume 1256 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1997.

[CGJ95] E.M. Clarke, O. Grumberg, and S. Jha. Verifying parametrized networks using abstraction and regular languages. In *6th International Conference on Concurrency Theory (CONCUR'95)*, pages 395–407, Philadelphia, PA, August 1995.

[DFvG83] E.W. Dijkstra, W.H.J. Feijen, and A.J.M van Gasteren. Derivation of a termination detection algorithm for distributed computations. *Info. Proc. Lett.*, 16(5):217–219, 1983.

[GZ98] E.P. Gribomont and G. Zenner. Automated verification of szymanski's algorithm. In B. Steffen, editor, *4th Intl. Conf. TACAS'98*, volume 1384 of *Lect. Notes in Comp. Sci.*, pages 424–438. Springer-Verlag, 1998.

[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 *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.

[JL98] E. Jensen and N.A. Lynch. A proof of burn's n-process mutual exclusion algorithm using abstraction. In B. Steffen, editor, *4th Intl. Conf. TACAS'98*, volume 1384 of *Lect. Notes in Comp. Sci.*, pages 409–423. Springer-Verlag, 1998.

[JN00] B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In S. Graf, editor, *Proceedings of TACAS'00*, Lect. Notes in Comp. Sci., 2000. To Appear.

[KM89] R.P. Kurshan and K. McMillan. A structural induction theorem for processes. In P. Rudnicki, editor, *Proceedings of the 8th Annual Symposium on Principles of Distributed Computing*, pages 239–248, Edmonton, AB, Canada, August 1989. ACM Press.

[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} Intl. Conference on Computer Aided Verification (CAV'97)*, Lect. Notes in Comp. Sci., pages 424–435. Springer-Verlag, 1997.

[MAB+94] Z. Manna, A. Anuchitanukul, N. Bjørner, A. Browne, E. Chang, M. Colón, L. De Alfaro, H. Devarajan, H. Sipma, and T.E. Uribe. STeP: The Stanford Temporal Prover. Technical Report STAN-CS-TR-94-1518, Dept. of Comp. Sci., Stanford University, Stanford, California, 1994.

[MP90] Z. Manna and A. Pnueli. An exercise in the verification of multi – process programs. In W.H.J. Feijen, A.J.M van Gasteren, D. Gries, and J. Misra, editors, *Beauty is Our Business*, pages 289–301. Springer-Verlag, 1990.

[SG92] A.P. Sistla and S.M. German. Reasoning about systems with many processes. *J. ACM*, 39:675–735, 1992.

[Tho90] W. Thomas. Automata on infinite objects. *Handbook of theoretical computer science*, pages 165–191, 1990.

[WL89] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lect. Notes in Comp. Sci.*, pages 68–80. Springer-Verlag, 1989.