# Regular Model Checking for LTL(MSO)*

Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson,
Julien d'Orso, and Mayank Saksena

Dept. of Information Technology, P.O. Box 337, S-751 05 Uppsala, Sweden
{parosh,bengt,marcusn,juldor,mayanks}@it.uu.se

**Abstract.** Regular model checking is a form of symbolic model checking for parameterized and infinite-state systems whose states can be represented as words of arbitrary length over a finite alphabet, in which regular sets of words are used to represent sets of states. We present $LTL(MSO)$, a combination of the logics MSO and LTL as a natural logic for expressing temporal properties to be verified in regular model checking. $LTL(MSO)$ is a two-dimensional modal logic, where MSO is used for specifying properties of system states and transitions, and LTL is used for specifying temporal properties. In addition, the first-order quantification in MSO can be used to express properties parameterized on a position or process. We give a technique for model checking $LTL(MSO)$, which is adapted from the automata-theoretic approach: a formula is translated to a (Büchi) transducer with a regular set of accepting states, and regular model checking techniques are used to search for models. We have implemented the technique and show its application to a number of parameterized algorithms from the literature.

## 1 Introduction

Regular model checking is a framework for algorithmic symbolic verification of parameterized and infinite-state systems [BLW03,KMM+01,WB98,BJNT00]. It considers systems whose states can be represented as finite words of arbitrary length over a finite alphabet, including array or ring-formed parameterized systems with an arbitrary number of finite-state processes, and systems that operate on queues, stacks, integers, and other linear unbounded data structures. In a system description, the set of initial states is represented as a regular set of strings, and the transition relation is given as a finite regular length-preserving transducer. Previous work on regular model checking [JN00,BJNT00,AJNd02] has developed methods for computing the set of reachable states of a system description, as well as the set of reachable loops, obtained from the transitive closure of the transition relation. In general, this problem is undecidable, but decidable results for certain classes have been obtained [JN00].

* This work was supported in part by the European Commission (FET project ADVANCE, contract No IST-1999-29082), and by the the Swedish Research Council (http://www.vr.se/).

R. Alur and D.A. Peled (Eds.): CAV 2004, LNCS 3114, pp. 348–360, 2004.

The techniques for computing reachable states and reachable loops can in principle be used to verify both safety and liveness properties of parameterized system descriptions, but do not provide a convenient approach for checking arbitrary temporal logic properties of parameterized and infinite-state systems. Significant ingenuity is required in order to manually transform the verification of a temporal property of a parameterized system into a property of reachable states and transitive closures, in particular if the verification uses fairness properties that are parameterized on system components [BJNT00,PS00]. It would be desirable to have a framework, analogous to the automata-theoretic approach in finite-state model checking [VW86], where the property of verifying a temporal property is automatically transformed into a problem of checking emptiness for a Büchi automaton.

In this paper, we address this problem by presenting an extension of the automata-theoretic approach [VW86] to the setting of regular model checking. We present a logic for expressing system models and temporal properties, which is a combination of the logics MSO over finite words and LTL. We use MSO for specifying sets of states and transition relations and LTL for specifying temporal constraints. The result is a two-dimensional modal logic, where MSO is used in the "space" (system state) dimension and LTL is used in the "time" dimension. Models of the logic are infinite sequences of (constant-length) words, representing computations of the specified system. We can then specify a verification problem as the conjunction of a system specification and a negation of the property to be verified.

Following the automata-theoretic approach, we present an automated translation from the translation from a formula $\varphi$ in $LTL(MSO)$ to a *Büchi transducer*, consisting of a regular set $I$ of initial states, a regular length-preserving transducer $T$, and a regular set $F$ of accepting states. Accepting runs of the Büchi transducer are infinite sequences of words, where the first word is in $I$, consecutive words satisfy $T$, and infinitely many words are in $F$. Thus, $\varphi$ is satisfiable if and only if the Büchi transducer has an accepting run. Since $T$ is length-preserving, the existence of an accepting run can be checked by searching for a reachable loop which contains a state in $F$.

A nice feature of our combination of MSO with LTL is that we get the power to express parameterized temporal properties for free: MSO offers variables to represent positions and quantify over them, which can be interleaved with temporal operators. As a concrete example, for a parameterized mutual exclusion algorithm, a typical property one would want to express is the following.

> If all processes satisfy a weak fairness requirement, then each process that is interested in entering its critical section will eventually do so.

If the number of processes is fixed, the terms like *"each process"* can be replaced by explicit conjunctions to obtain a standard model checking problem in propositional temporal logic. However, in regular model checking the number of processes is arbitrary. Fortunately, our logic gives quantification for free, so we can express this property directly, by a formula like

$$\forall i.[\Box\Diamond(\text{blocked}(i) \vee \text{progressing}(i))] \longrightarrow \forall i.[\Box(\text{trying}(i) \rightarrow \Diamond\text{critical}(i))]$$

where $i$ ranges over positions in the state, and each position represents a process. In this formula, we apply LTL operators ($\Box$ and $\Diamond$) to formulas with the MSO variable $i$, and later use MSO quantification over $i$ to express parameterized properties. In our logic $LTL(MSO)$, temporal operators can be applied to formulas with at most one free first-order variable and no free second-order variables. This restriction allows to express parameterized temporal properties (e.g., fairness constraints) of individual processes in a parameterized system, as well as temporal properties of pairs of adjacent processes (in positions $i$ and $i+1$ using one free variable $i$).

A further nice property of adapting the automata-theoretic approach is that our transformation results in a uniform problem of checking for accepting runs, for which we can develop techniques that are more uniform than those presented in previous work [JN00,BJNT00,AJNd02]. We have extended our tool for regular model checking [AJNd03] to check whether Büchi transducers have accepting runs. This is done in two steps. First, the set of reachable states are computed as $Inv = I \circ T^*$. Secondly, loops are found by identifying identical pairs in $(F \cap T \cap Inv) \circ T^*$. This computation is more uniform and more efficient than the approach to verification of temporal logic properties outlined in [BJNT00], which builds on computation of the transitive closure $T^+$ of the transducer relation. We have verified safety properties with the tool for many of the examples in our previous work, as well as liveness properties for some of the examples.

As special cases, when the formula contains no temporal operators, our method specializes into a decision procedure for MSO similar to that of MONA [HJJ$^+$96], and when the formula contains no quantifiers our method specializes to ordinary LTL model checking.

The remainder of the paper is structured as follows. In the next section, we present the logic $LTL(MSO)$. Section 3 illustrates how it can be used to model and specify parameterized algorithms. The model checking technique, including the translation to Büchi transducer is presented and proven correct in Section 4. New techniques and evaluations of our extended implementation are presented in Section 5.

*Related Work.* In addition to the work on regular model checking, cited earlier, there is a large body of research on the problem of model checking parameterized systems of *identical* processes, in which there is no ordering between processes, and hence the system state can be represented as a multiset of process states (e.g., [BLS01,Del00,EK03,EK00,GS92]). This problem is substantially simpler, since ordering between processes need not be considered. Emerson and Namjoshi [EN95] give a technique for verifying a restricted class of parameterized token-passing algorithms by reducing an arbitrary ring to a small fixed-size ring under certain conditions. These restrictions are substantially stronger than in our framework. Sistla [Sis97] uses Büchi automata over two dimensional languages (reminding of transducers) to specify network invariants when verifying systems by induction over their linear process structure. It is unclear what class of systems can be handled automatically by this technique.

The problem of checking liveness properties of array-shaped parameterized systems was considered by Pnueli and Shahar [PS00], who presented a technique for computing the transitive closure of a restricted class of transition relations. They also first manually employ abstractions to make the implementation terminate.

Pnueli, Xu, and Zuck [PXZ02] present an interesting use of specialized abstractions in order to prove absence of starvation properties for Szymanski's algorithm and the Bakery algorithm. The abstractions keep track of the number of processes with certain properties, and generate a finite-state system, which can be model-checked. The presented abstraction is specialized to prove non-starvation, and loses much information so that e.g., safety properties can no longer be checked.

Our logic *LTL*(*MSO*) applied to words is related to existential monadic second-order logic (EMSO) on grids to define picture languages accepted by *tiling systems* (see e.g. [GR97]). Indeed, transducers over words can be considered as *tiling systems* where each transition represents a *tile*. Thus, it is expected that our logic *LTL*(*MSO*) is equivalent to EMSO on grids. However, the two logics come from different motivations. While EMSO on grids is used to reason about *pictures*, our logic is used to reason about *parameterized structures over time*. When applied to the word structure, the two logics coincide.

## 2 The Logic *LTL*(*MSO*)

We present the syntax and semantics of *LTL*(*MSO*). This logic combines the operators of MSO and LTL, under the restriction that temporal operators can be applied to formulas with at most one free first-order (MSO) variable and no free second-order variables. In this paper, we restrict the presentation to the temporal operators $\Box$ (always) and $\Diamond$ (eventually), but this is not a fundamental restriction.

**Syntax.** We denote *first-order variables* $\mathrm{Var}_{FO}$ by lowercase letters $i, j, k, \ldots$, *second-order variables* $\mathrm{Var}_{SO}$ by uppercase letters $I, J, K, \ldots$ and *configuration variables* $V$ by lowercase letters $x, y, z, \ldots$

| | |
|---|---|
| $\varphi ::= i \in I \mid I \subseteq J \mid i = j + 1 \mid x[i] \mid x'[i]$ | Atomic MSO formulas |
| $true \mid false$ | Boolean constants |
| $\varphi \sim \varphi, \sim \in \{ \wedge , \vee , \rightarrow , \leftrightarrow \} \mid \neg\varphi$ | Propositional connectives |
| $\exists i\varphi \mid \forall i\varphi \mid \exists I\varphi \mid \forall I\varphi$ | MSO Quantification |
| $\Box\varphi \mid \Diamond\varphi$ | Temporal operators |

No negation should have any temporal operator within its scope, which is why we provide dual versions of each operator. We further require that in every subformula of the form $\Box\varphi$ or $\Diamond\varphi$, there is at most one free first-order variable (and no free second-order variables). More operators on first-order variables, such as $<$, $\leq$, $=$, can be expressed in MSO using standard encodings.

**Semantics.** Closed *LTL*(*MSO*) formulas are interpreted over *matrices* $M$ over $2^V$ of dimension $\infty \times n$ where $n \in \mathbf{Z}_+$ is a parameter that can take an arbitrary positive integer value. We call the vertical (first) dimension *time*, and the horizontal (second) dimension *space*. Let $\mathbf{Z}_n = \{0, 1, \ldots, n-1\}$. The element $M(t,i) \in 2^V$ for $t \in \mathbf{N}$ and $i \in \mathbf{Z}_n$ represents the system configuration at time $t$ of position (or subsystem) $i$, which assigns truth values to the configuration variables $V$. Thus, each row $M(t,0) \cdots M(t,n)$ represents the system configuration at time $t$.

In general, a formula $\varphi$ depends on its free first- and second-order variables, which are interpreted in the space dimension, and on a time point $t$. A model $\mathcal{M}$ for an arbitrary *LTL*(*MSO*)-formula $\phi$ is a triple $(M, \mathcal{I}, t)$, where $M : (\mathbf{N} \times \mathbf{Z}_n) \to 2^V$ is a matrix over $2^V$, where $\mathcal{I}$ is a valuation $\mathcal{I}$: $\mathrm{Var}_{FO} \to \mathbf{Z}_n$ and $\mathcal{I}$: $\mathrm{Var}_{SO} \to 2^{\mathbf{Z}_n}$ of first-order and second-order variables, and where $t$ is a time point $t \in \mathbf{N}$. Satisfaction of formulas is defined as follows.

$(M, \mathcal{I}, t) \not\models false$
$(M, \mathcal{I}, t) \models true$

| | |
|---|---|
| $i \in I$ | iff $\mathcal{I}(i) \in \mathcal{I}(I)$ |
| $I \subseteq J$ | iff $\mathcal{I}(I) \subseteq \mathcal{I}(J)$ |
| $i = j + 1$ | iff $\mathcal{I}(i) = \mathcal{I}(j) + 1$ |
| $x[i]$ | iff $x \in M(t, \mathcal{I}(i))$ |
| $x'[i]$ | iff $x \in M(t+1, \mathcal{I}(i))$ |
| $\varphi_1 \wedge \varphi_2$ | iff $(M, \mathcal{I}, t) \models \varphi_1$ and $(M, \mathcal{I}, t) \models \varphi_2$ |
| $\varphi_1 \vee \varphi_2$ | iff $(M, \mathcal{I}, t) \models \varphi_1$ or $(M, \mathcal{I}, t) \models \varphi_2$ |
| $\varphi_1 \rightarrow \varphi_2$ | iff $(M, \mathcal{I}, t) \models \varphi_1$ implies $(M, \mathcal{I}, t) \models \varphi_2$ |
| $\varphi_1 \leftrightarrow \varphi_2$ | iff $(M, \mathcal{I}, t) \models \varphi_1$ if and only if $(M, \mathcal{I}, t) \models \varphi_2$ |
| $\neg\varphi$ | iff $(M, \mathcal{I}, t) \not\models \varphi$ |
| $\exists i \varphi$ | iff $\exists m \in \mathbf{Z}_n$ $(M, \mathcal{I}[i \mapsto m], t) \models \varphi$ |
| $\forall i \varphi$ | iff $\forall m \in \mathbf{Z}_n$ $(M, \mathcal{I}[i \mapsto m], t) \models \varphi$ |
| $\exists I \varphi$ | iff $\exists S \in 2^{\mathbf{Z}_n}$ $(M, \mathcal{I}[I \mapsto S], t) \models \varphi$ |
| $\forall I \varphi$ | iff $\forall S \in 2^{\mathbf{Z}_n}$ $(M, \mathcal{I}[I \mapsto S], t) \models \varphi$ |
| $\Box\varphi$ | iff $\forall t' \geq t$ $(M, \mathcal{I}, t') \models \varphi$ |
| $\Diamond\varphi$ | iff $\exists t' \geq t$ $(M, \mathcal{I}, t') \models \varphi$ |

where $\mathcal{I}[i \mapsto m]$ is the valuation which sends $i$ to $m$ and otherwise acts as $\mathcal{I}$, and $\mathcal{I}[I \mapsto S]$ is defined analogously.

## 3 Modeling in *LTL*(*MSO*)

In this section, we discuss how to model systems and set up a verification problem in *LTL*(*MSO*).

A system model consists of a set of initial states, a transition relation, and possibly fairness properties. Let a *state formula* be a formula without temporal operators or primed configuration variables. Following the style of TLA [Lam91], the initial states are specified by a state formula $\varphi_I$, the transition relation by a formula $\varphi_T$ over unprimed and primed configuration variables without temporal

operators, and fairness properties by a formula $\varphi_{fair}$. A *property* is given as a formula $\varphi$; for instance, an invariant property is of the form $\Box\varphi_{Inv}$ for a state formula $\varphi_{Inv}$. To check whether the system model satisfies the property $\varphi$, we search for models of the formula

$$\varphi_I \ \wedge \ \Box\varphi_T \ \wedge \ \varphi_{fair} \ \wedge \ \neg\varphi \ .$$

If $\varphi$ is a safety property, the fairness properties $\varphi_{fair}$ are not necessary, and can be omitted.

There are several forms of fairness properties. Process fairness can be expressed as

$$\varphi_{fair} = \forall i \Box\Diamond(\mathcal{A}(i) \ \vee \ \neg\text{enabled}(\mathcal{A}(i))$$

where $\mathcal{A}(i)$ represents all actions of process $i$, and enabled$(\mathcal{A}(i))$ represents the set of states where the action $\mathcal{A}(i)$ can be taken, obtained using an existential quantification of the primed configuration variables in $\mathcal{A}(i)$.

*Example.* To illustration modeling in *LTL*(*MSO*), we take the Bakery algorithm for mutual exclusion. This algorithm synchronizes entries into critical section for an arbitrary number of processes, using a mechanism with integer-numbered "tickets". A process which wants to get into the critical section receives a ticket which is the maximum of all the outstanding tickets plus one. When a process has the lowest outstanding ticket, it enters the critical section and drops the ticket when leaving. Below is a pseudo-code description of the algorithm, in which the ticket of process $p$ is represented by the variable $ticket_p$, whose value is initially 0.

| | |
|---|---|
| Idle: | $ticket_p := 1 + \max_q ticket_q$ |
| Waiting: | **await** $\forall q \neq p : (ticket_p < ticket_q \ \vee \ ticket_q = 0)$ |
| Critical: | $ticket_p := 0$ |

**Fig. 1.** Bakery Algorithm

To model the Bakery Algorithm in *LTL*(*MSO*), we change the perspective; rather than modeling the vector of process states, we let a configuration represent the states of the sequence of ticket numbers, using the configuration variable $q$. For each $i$, the value of $q[i]$ is

- $\perp$ if there is no process that has ticket $i+1$, and
- $W$ if some process with ticket $i+1$ is waiting, and
- $C$ if some process with ticket $i+1$ is in critical

Note that we do not model tickets with number 0, since this is the ticket number of all "inactive" processes. For brevity, we implicitly use the invariant that each positive ticket number can be held by at most one process. This invariant can be verified separately, or not be assumed (e.g., by adding one more value of $q[i]$ representing that several process have this ticket number).

The initial state and transition relation of the Bakery Algorithm can then be specified by the following formulas.

$$
\begin{array}{lcl}
\textbf{initial} & = & \forall i\ q[i] = \bot \\
\textbf{ticket}(i) & = & q[i] = \bot\ \wedge\ q'[i] = W\ \wedge\ (\forall j > i\ q[j] = \bot)\ \wedge \\
 & & (i \neq 0\ \rightarrow\ q[i-1] \neq \bot) \\
\textbf{enter}(i) & = & q[i] = W\ \wedge\ q'[i] = C\ \wedge\ (\forall j < i\ q[j] = \bot) \\
\textbf{exit}(i) & = & q[i] = C\ \wedge\ q'[i] = \bot \\
\textbf{copy}(i) & = & q[i] = q'[i] \\
\textbf{actions}(i) & = & (\textbf{ticket}(\mathrm{i})\ \vee\ \textbf{enter}(i)\ \vee\ \textbf{exit}(i))\ \wedge\ (\forall j \neq i\ \textbf{copy}(j)) \\
\textbf{sys} & = & \textbf{initial}\ \wedge\ \Box((\exists i\ \textbf{actions}(i))\ \vee\ \forall i\ \textbf{copy}(i))
\end{array}
$$

Mutual exclusion can be specified by the formula

$$\textbf{mutex} \quad = \quad \Box\neg(\exists i \exists j\ i \neq j\ \wedge\ q[i] = C\ \wedge\ q[j] = C)$$

In order to specify non-starvation, we add a fairness constraint for the actions $\textbf{enter}(i)$ and $\textbf{exit}(i)$. We add no fairness constraint for $\textbf{ticket}(i)$, since the arrival of new processes should not be controlled by the algorithm itself.

$$
\begin{array}{lcl}
\textbf{fairactions}(i) & = & (\textbf{enter}(i)\ \vee\ \textbf{exit}(i))\ \wedge\ (\forall j \neq i\ \textbf{copy}(j)) \\
\textbf{fairness} & = & \forall i\ \Box\Diamond(\textbf{fairactions}(i)\ \vee\ \neg\mathrm{enabled}(\textbf{fairactions}(i))) \\
\textbf{non-starvation} & = & \forall i\ \Box\ (q[i] = W\ \rightarrow\ \Diamond q[i] = C)
\end{array}
$$

To check that the algorithm satisfies mutual exclusion and non-starvation, we should check that the formulas

$$\textbf{sys}\ \wedge\ \neg\textbf{mutex}$$
$$\textbf{sys}\ \wedge\ \textbf{fairness}\ \wedge\ \neg\textbf{non-starvation}$$

do not have any models. The property that models are of arbitrary but fixed size implies that we actually verify the algorithm under the assumption that there is an arbitrarily chosen upper bound on ticket numbers. For safety properties, this is not a limitation since violations will be finite sequences of computation steps, but for fairness properties it can play a role. For the Bakery Algorithm, it can be seen that an arbitrary upper limit on ticket numbers does not affect non-starvation for waiting processes, but in general one must be aware of this modeling constraint.

## 4 Translating *LTL*(*MSO*) to Büchi Transducers

In this section, we describe how to transform a formula in $LTL(MSO)$ into a Büchi transducer, consisting of a regular set $I$ of initial states, a regular length-preserving transducer $T$, and a regular set $F$ of accepting states. In this section, we will represent such a Büchi transducer by the $LTL(MSO)$ formula $\phi_I \wedge \Box\phi_T \wedge \Box\Diamond\phi_F$, where $\phi_I$ and $\phi_F$ are MSO formulas denoting the regular sets of initial and accepting states, and $\phi_T$ is an MSO formula over primed and unprimed

configuration variables denoting the regular transition relation. The accepting runs of the Büchi transducer accept exactly the models of the temporal formula, so in the following we will consider $\phi_I \wedge \Box\phi_T \wedge \Box\Diamond\phi_F$ to be a Büchi transducer where $I$, $T$, and $F$ are regular transducers expressed in MSO.

The idea of the construction is similar to that of the standard translation of propositional temporal logic to Büchi Automata [VW86]: the semantics of temporal operators is translated to additional state information in the Büchi automaton. In our case the operators are translated to new configuration variables that represent the values of certain temporal subformulas of $\phi$. The semantics of these subformulas is represented by constraints on the values of the new configuration variables.

We assume that $\phi$ is in negative normal form. Define a *core subformula* of $\phi$ as a subformula of $\phi$ which has a temporal operator as its main connective. For each core subformula $\psi$ of $\phi$, we introduce an auxiliary configuration variable $x_\psi$. Intuitively, the value of $x_\psi$ at time point $t$ and space position $i$ should represent the value of $\psi$ at that entry in the matrix. For each core subformula $\Diamond\psi$ where the main connective is $\Diamond$ we introduce an auxiliary configuration variable $y_{\Diamond\psi}$ (called an *eventuality variable*). Intuitively, if the variable $y_{\Diamond\psi}$ is true, then the formula $\psi$ should be true at some future time point.

Using the auxiliary configuration variables, the value of any subformula $\psi$ can be represented by a formula in MSO over the extended set of configuration variables. Let us define this formula $\langle\!\langle\psi\rangle\!\rangle$ as follows:

$$\begin{array}{lcl}
\langle\!\langle\phi\rangle\!\rangle & = & \phi \text{ for } \phi \text{ in MSO} \\
\langle\!\langle\psi_1 \wedge \psi_2\rangle\!\rangle & = & \langle\!\langle\psi_1\rangle\!\rangle \wedge \langle\!\langle\psi_2\rangle\!\rangle \\
\langle\!\langle\psi_1 \vee \psi_2\rangle\!\rangle & = & \langle\!\langle\psi_1\rangle\!\rangle \vee \langle\!\langle\psi_2\rangle\!\rangle \\
\langle\!\langle\exists i\psi\rangle\!\rangle & = & \exists i\langle\!\langle\psi\rangle\!\rangle \\
\langle\!\langle\forall i\psi\rangle\!\rangle & = & \forall i\langle\!\langle\psi\rangle\!\rangle \\
\langle\!\langle\exists I\psi\rangle\!\rangle & = & \exists I\langle\!\langle\psi\rangle\!\rangle \\
\langle\!\langle\forall I\psi\rangle\!\rangle & = & \forall I\langle\!\langle\psi\rangle\!\rangle \\
\langle\!\langle\Box\psi\rangle\!\rangle & = & x_{\Box\psi} \\
\langle\!\langle\Diamond\psi\rangle\!\rangle & = & x_{\Diamond\psi}
\end{array}$$

Let $\mathbf{localconstr}(\phi)$ define the conjunction of a set of constraints on the auxiliary variables, called *local constraints*, as defined below.

(a) For each auxiliary variable of form $x_\psi$, where $i$ is the (possibly) free first-order variable in $\psi$, define its constraint as follows.

$$\begin{array}{ll}
\forall i.\left(x_{\Box\psi_1}[i] \leftrightarrow \langle\!\langle\psi_1\rangle\!\rangle[i] \wedge x'_{\Box\psi_1}[i]\right) & \text{when } \psi \text{ is } \Box\psi_1 \\
\forall i.\left(x_{\Diamond\psi_1}[i] \leftrightarrow \langle\!\langle\psi_1\rangle\!\rangle[i] \vee x'_{\Diamond\psi_1}[i]\right) & \text{when } \psi \text{ is } \Diamond\psi_1
\end{array}$$

(b) Let $y_{\Diamond\psi_1}, \ldots, y_{\Diamond\psi_k}$ be the set of eventuality variables. We define their local constraint as:

$$\bigwedge_{m=1}^{k} \forall i.\left(y_{\Diamond\psi_m}[i] \ \wedge \ \neg y'_{\Diamond\psi_m}[i] \longrightarrow \langle\!\langle\psi_m\rangle\!\rangle[i]\right)$$

Furthermore, we require that all eventuality variables are false infinitely often and that they load new eventualities: let **evconstr**($\phi$) define the conjunction of a set of constraints on the auxiliary variables, called *eventuality constraint*, defined below.

$$\bigwedge_{m=1}^{k} \forall i.\, \big(\neg y_{\Diamond\psi_m}[i] \ \wedge\ (y'_{\Diamond\psi_m}[i] \leftrightarrow x'_{\Diamond\psi_m}[i])\big)$$

The goal of this section is to prove that for each formula $\phi$ there is a model $M$ such that $(M,\mathcal{I},t) \models \phi$ iff there is a model $M'$ such that $(M',\mathcal{I},t) \models \langle\!\langle\phi\rangle\!\rangle \ \wedge\ \Box\mathbf{localconstr}(\phi) \ \wedge\ \Box\Diamond\mathbf{evconstr}(\phi)$ where $M'$ differs from $M$ only on the auxiliary variables. By its form, the latter formula can be regarded as a Büchi transducer, and so we have translated $\phi$ into an equivalent Büchi transducer. We first prove the following lemma.

**Lemma 1.** *If* $(M',\mathcal{I},t) \models \Box\mathbf{localconstr}(\phi) \ \wedge\ \Box\Diamond\mathbf{evconstr}(\phi)$*, then for all auxiliary variables of* $\phi$ *we have*

*(a)* $(M',\mathcal{I},t) \models \forall i.x_{\Box\psi}[i] \longrightarrow \Box\langle\!\langle\psi\rangle\!\rangle$
*(b)* $(M',\mathcal{I},t) \models \forall i.x_{\Diamond\psi}[i] \longrightarrow \Diamond\langle\!\langle\psi\rangle\!\rangle \ \vee\ \Diamond y_{\Diamond\psi}[i]$
*(c)* $(M',\mathcal{I},t) \models \forall i.y_{\Diamond\psi}[i] \longrightarrow \Diamond\langle\!\langle\psi\rangle\!\rangle$

*Proof.* (a) Let $(M',\mathcal{I}',t) \models x_{\Box\psi}[i]$ for some $\mathcal{I}' = \mathcal{I}[i \mapsto m]$. Since $(M',\mathcal{I},t) \models \Box\mathbf{localconstr}(\phi)$, we have

$$(M',\mathcal{I}',t) \models \Box\left(x_{\Box\psi}[i] \leftrightarrow \langle\!\langle\psi\rangle\!\rangle[i] \wedge x'_{\Box\psi}[i]\right)$$

It follows that $(M',\mathcal{I}',t') \models \langle\!\langle\psi\rangle\!\rangle[i]$ for every $t' \geq t$.

(b) Let $(M',\mathcal{I}',t) \models x_{\Diamond\psi}[i]$ for some $\mathcal{I}' = \mathcal{I}[i \mapsto m]$. Since $(M',\mathcal{I},t) \models \Box\mathbf{constr}(\phi) \ \wedge\ \Box\Diamond\mathbf{evconstr}(\phi)$, we have
  – $(M',\mathcal{I}',t) \models \Box\left(x_{\Diamond\psi}[i] \leftrightarrow \langle\!\langle\psi\rangle\!\rangle[i] \vee x'_{\Diamond\psi}[i]\right)$.
  – $(M',\mathcal{I}',t') \models y'_{\Diamond\psi}[i] \leftrightarrow x'_{\Diamond\psi}[i]$ for some $t' \geq t$

  If $(M',\mathcal{I}',t'') \models \langle\!\langle\psi(i)\rangle\!\rangle$ for some $t'' \geq t$ we are done, suppose that no such $t''$ exists. Then it follows that $(M',\mathcal{I}',t') \models y'_{\Diamond\psi}[i]$ and thus $(M',\mathcal{I}',t'+1) \models y_{\Diamond\psi}[i]$.

(c) Let $(M',\mathcal{I}',t) \models y_{\Diamond\psi}[i]$ for some $\mathcal{I}' = \mathcal{I}[i \mapsto m]$, and let $t' > t$ be the earliest point in time when

$$(M',\mathcal{I}',t') \models \neg y_{\Diamond\psi}[i]$$

Then we have

$$(M',\mathcal{I}',t'-1) \models y_{\Diamond\psi}[i] \ \wedge\ \neg y'_{\Diamond\psi}[i]$$

and by the eventuality constraint of $y_{\Diamond\psi}$ we get

$$(M',\mathcal{I}',t'-1) \models \langle\!\langle\psi\rangle\!\rangle(i)$$

Since $t'-1 \geq t$ we conclude that

$$(M',\mathcal{I}',t) \models \Diamond\langle\!\langle\psi(i)\rangle\!\rangle$$

**Theorem 1.** *For each subformula $\psi$ of $\phi$, interpretation $I$ and timepoint $t$ we have there is a model $M$ such that $(M,\mathcal{I},t) \models \psi$ iff there is a model $M'$ such that $(M',\mathcal{I},t) \models \langle\langle\psi\rangle\rangle \wedge \Box\textbf{localconstr}(\phi) \wedge \Box\Diamond\textbf{evconstr}(\phi)$ where $M$ differs from $M'$ only in the auxiliary variables of $\phi$.*

*Proof.* $\Longrightarrow$: This case is rather straight-forward. Define $M^{\phi}(i)$ by letting $x_{\psi(i)}$ be true at point $M^{\phi}(t,i)$ iff $(M,\mathcal{I},t) \models \psi(i)$. Then the local constraints follow by the definitions of the temporal operators. The values of variables $y_{\Diamond\psi(i)}$ are then defined in a way that satisfies the eventuality constraint:

- at time $t$, all variables $y_{\Diamond\psi}[i]$ are set to false;
- if all $y_{\Diamond\psi}[i]$ are false at time $t'$, then set the value of each $y_{\Diamond\psi}[i]$ at time $t'+1$ to the value of corresponding $x_{\Diamond\psi}[i]$ at time $t'+1$;
- if variable $y_{\Diamond\psi}[i]$ is true at time $t'$, and $\langle\langle\psi(i)\rangle\rangle$ holds, then set $y_{\Diamond\psi}[i]$ to false at time $t'+1$.

$\Longleftarrow$: Let $(M',\mathcal{I},t) \models \langle\langle\psi\rangle\rangle \wedge \Box\textbf{localconstr}(\phi) \wedge \Box\Diamond\textbf{evconstr}(\phi)$. We prove by induction over the structure of $\psi$.

$\psi \in MSO$ : Since $\langle\langle\psi\rangle\rangle = \psi$, we get $(M',\mathcal{I},t) \models \psi$.

$\psi = \exists i\psi_0$ : We get $(M',\mathcal{I},t) \models \exists i\langle\langle\psi_0\rangle\rangle$ and by semantics $(M',\mathcal{I}[i \mapsto m],t) \models \langle\langle\psi_0\rangle\rangle$ for some $m \in \mathbf{Z}$. Since $\Box\textbf{localconstr}(\phi) \wedge \Box\Diamond\textbf{evconstr}(\phi)$ is a closed formula and thus does not depend on $i$, it follows that $(M',\mathcal{I}[i \mapsto m],t) \models \Box\textbf{localconstr}(\phi) \wedge \Box\Diamond\textbf{evconstr}(\phi)$. By induction there is some $M$ that differs from $M'$ only in the auxiliary variables of $\psi_0$ such that $(M,\mathcal{I}[i \mapsto m],t) \models \psi_0$. By semantics we get $(M,\mathcal{I},t) \models \exists i\psi_0$.

$\psi = \{\exists I\psi_0, \forall i\psi_0, \forall I\psi_0\}$ Similar to $\psi = \exists i\psi_0$.

$\psi = \Box\psi_0$ : We get $(M',\mathcal{I},t) \models x_{\Box\psi_0}$. By Lemma 1 it follows that $(M',\mathcal{I},t) \models \Box\langle\langle\psi_0\rangle\rangle$. Then by semantics for all $t' \geq t$, we have $(M',\mathcal{I},t') \models \langle\langle\psi_0\rangle\rangle$ and then by induction there is some $M$ that differs from $M'$ only in the auxiliary variables of $\psi_0$ such that $(M,\mathcal{I},t') \models \psi_0$. Since this holds for all $t' \geq t$ we have by semantics that $(M,\mathcal{I},t) \models \Box\psi_0$.

$\psi = \Diamond\psi_0$ : We get $(M',\mathcal{I},t) \models x_{\Diamond\psi_0}$. By Lemma 1 it follows that $(M',\mathcal{I},t) \models \Diamond\langle\langle\psi_0\rangle\rangle \vee \Diamond y_{\Diamond\psi_0}$. We have $(M',\mathcal{I},t') \models \langle\langle\psi_0\rangle\rangle$ for some $t' \geq t$ in both cases as follows. In the case $(M',\mathcal{I},t) \models \Diamond y_{\Diamond\psi_0}$ holds, it follows by Lemma 1, and in the case $(M',\mathcal{I},t) \models \Diamond\langle\langle\psi_0\rangle\rangle$ holds it follows from semantics. Then by induction there is some $M$ that differs from $M'$ only in the auxiliary variables of $\psi_0$ such that $(M,\mathcal{I},t') \models \psi_0$. Since $t' \geq t$ we have by semantics that $(M,\mathcal{I},t) \models \Diamond\psi_0$. □

**Corollary 1.** *For a formula $\phi$, interpretation $I$ and timepoint $t$ we have that there is a model $M$ such that $(M,\mathcal{I},t) \models \phi$ iff there is a model $M'$ such that $(M',\mathcal{I},t) \models \langle\langle\phi\rangle\rangle \wedge \Box\textbf{localconstr}(\phi) \wedge \Box\Diamond\textbf{evconstr}(\phi)$ where $M$ differs from $M'$ only in the auxiliary variables of $\phi$.*

## 5 Implementation

The transformation of the previous section results in a formula of the form

$$I \wedge \Box T \wedge \Box\Diamond F$$

where the formulas $I$, $T$, and $F$ are in MSO and can be translated into transducers. The transducers can be seen as binary relations on words that are closed under standard operations such as union, intersection and composition. For the sake of presentation, we will assume that $I$ only contains non-primed configuration variables and look at $I$ as a set instead of a binary relation. The general case does not introduce any significant additional problems.

We find models of the formula $I \ \wedge \ \Box T \ \wedge \ \Box\Diamond F$ as follows.

(a) Compute the set of reachable states as $Inv = I \circ T^*$.
(b) Define $T_{Inv}$ as $\{(w, w') \in T \ : \ w \in Inv\}$ and compute the set of loops by finding identical pairs in $(T_{Inv} \ \cap \ F) \circ T^*$

We use an optimized version of the above scheme as follows. When computing the invariant $I \circ T^*$ the transition relation $T$ contains local constraints for eventuality variables. For eventualities that hold infinitely often, such local constraints are only needed when checking for loops through $F$ and not when computing the set of reachable states. To simplify the transition relation used when computing the set of reachable states, the formula is transformed into the following form:

$$I \ \wedge \ \Box T_0 \ \wedge \ \Diamond(F \ \wedge \ \Box T \ \wedge \ \Box\Diamond F)$$

where $T_0$ is like $T$ but without the local constraints on eventuality variables for eventualities that hold infinitely often. The final procedure is then as follows.

(a) Compute the set of reachable states $Inv = I \circ T_0^*$.
(b) As before, define $T_{Inv}$ as $\{(w, w') \in T \ : \ w \in Inv\}$ and compute the set of loops by finding identical pairs in $(T_{Inv} \ \cap \ F) \circ T^*$.

To calculate expressions like $T_1 \circ T_2^*$, we employ a technique for computing arbitrary relational compositions of transducers, expressed as regular expressions, by encoding the states of the automaton recognizing the regular expression in the first position of the word. As an example, the expression $T_1 \circ T_2^*$ corresponds to an automaton with two states $q_1$, and $q_2$. We use a transducer accepting the union of $(q_1, q_2) \cdot T_1$ and $(q_2, q_2) \cdot T_2$. Words in the transitive closure of the resulting transducer, of the form $(q_1, q_2) \cdot (w[1], w'[1]) \cdots (w[n], w'[n])$, then represent an element $(w, w')$ in the relation $T_1 \circ T_2^*$.

We have successfully verified safety properties for many of the examples in our previous work, as well as liveness properties for some of the smaller examples. Execution times are given in the table below.

| Algorithm | Safety (ms) | Liveness (ms) |
|---|---|---|
| Tokenpass | 3930 | 14790 |
| Tokenring | 4840 | 29590 |
| Bakery | 4840 | 23070 |
| Burns | 38560 | |
| Dijkstra | 593220 | |
| Szymanski | 1073200 | |

We experience high execution times for checking safety properties for some of the algorithms. This is because we use the entire transition as opposed to computing the transitive closure of separate actions and using them for reachability analysis. This reachability analysis does not always terminate, however, since some algorithms require the transitive closure of combinations of actions for the analysis to terminate. Using the entire transition relation is a more general technique, but potentially more costly.

## References


[AJNd02] Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Julien d'Orso. Regular model checking made simple and efficient. In *Proc. CONCUR 2002,* $13^{th}$ *Int. Conf. on Concurrency Theory*, volume 2421 of *Lecture Notes in Computer Science*, pages 116–130, 2002.

[AJNd03] Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Julien d'Orso. Algorithmic improvements in regular model checking. In *Proc.* $14^{th}$ *Int. Conf. on Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, 2003.

[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. Springer Verlag, 2000.

[BLS01] K. Baukus, Y. Lakhnech, and K. Stahl. Verification of parameterized networks. *Journal of Universal Computer Science*, 7(2), 2001.

[BLW03] Bernard Boigelot, Axel Legay, and Pierre Wolper. Iterating transducers in the large. In *Proc.* $14^{th}$ *Int. Conf. on Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 223–235, 2003.

[Del00] G. Delzanno. Automatic verification of cache coherence protocols. In Emerson and Sistla, editors, *Proc.* $12^{th}$ *Int. Conf. on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 53–68. Springer Verlag, 2000.

[EK00] E.A. Emerson and V. Kahlon. Reducing model checking of the many to the few. In *Proc. 17th International Conference on Automated Deduction*, volume 1831 of *Lecture Notes in Computer Science*, pages 236–254. Springer Verlag, 2000.

[EK03] E.A. Emerson and V. Kahlon. Rapid parameterized model checking of snoopy cache coherence protocols. In *Proc. TACAS '03,* $9^{th}$ *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, 2003.

[EN95] E.A. Emerson and K.S. Namjoshi. Reasoning about rings. In *Proc.* $22^{th}$ *ACM Symp. on Principles of Programming Languages*, 1995.

[GR97] D. Giammarresi and A. Restivo. Two-dimensional languages. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages*, volume 3, Beyond Words, pages 215–267. Springer-Verlag, Berlin, 1997.

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

[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+01] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *Theoretical Computer Science*, 256:93–112, 2001.

[Lam91] L. Lamport. The temporal logic of actions. Technical report, DEC/SRC, 1991.

[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. Springer Verlag, 2000.

[PXZ02] A. Pnueli, J. Xu, and L. Zuck. Liveness with $(0, 1, \infty)$-counter abstraction. In Brinskma and Larsen, editors, *Proc.* $14^{th}$ *Int. Conf. on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 107–122. Springer Verlag, 2002.

[Sis97] A. Prasad Sistla. Parametrized verification of linear networks using automata as invariants. In O. Grumberg, editor, *Proc.* $9^{th}$ *Int. Conf. on Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 412–423, Haifa, Israel, 1997. Springer Verlag.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. LICS '86,* $1^{st}$ *IEEE Int. Symp. on Logic in Computer Science*, pages 332–344, June 1986.

[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.