A Unifying Model Checking Approach for Safety Properties of Parameterized Systems

Monika Maidl

LFCS, University of Edinburgh
The Kings's Buildings, Edinburgh EH9 3JZ

Abstract. We present a model checking algorithm for safety properties that is applicable to parameterized systems and hence provides a unifying approach of model checking for parameterized systems. By analysing the conditions under which the proposed algorithm terminates, we obtain a characterisation of a subclass for which this problem is decidable. The known decidable subclasses, token rings and broadcast systems, fall in our subclass, while the main novel feature is that (unnested) quantification over index variables is allowed, which means that global guards can be expressed.

1 Introduction

We present a model checking algorithm for safety properties that is applicable to parameterized systems. A parameterized system is a family of systems, one for each instantiation of the parameter, where an instantiation by n is the composition of n copies of the system, and the verification problem consists in checking whether all instantiations fulfil a given property. Model checking for parameterized systems has been shown to be undecidable in general Suz88, so the problem can only be approached for subclasses or by semi-algorithmic methods. Solutions based on different technical frameworks have been proposed. Since our model checking algorithm is applicable to parameterized systems in general, it provides a unifying method for different subclasses.

By exploring under which restrictions the algorithm terminates, we obtain a characterisation of a class of parameterized systems for which model checking of safety properties is decidable. It turns out that the restrictions concern almost exclusively the way the copies communicate with each other: The admissible forms of communication are the very restricted synchronisation of token rings (no information exchanged between neighbours except "I have/have not the token") and the anonymous synchronisation of broadcast protocols. Another form of communication, using values of variables in neighbouring copies in guards or assignments like in the Dining Philosophers example, has to be restricted. We can however allow communication by global variables and by global guards, (expressed by universal quantification over index variables, which run over the instantiated copies). The latter is used e.g. in the Bakery algorithm, and in cache coherence protocols with a global condition considered in Del00, and extends the known subclasses of decidable systems.

G. Berry, H. Comon, and A. Finkel (Eds.): CAV 2001, LNCS 2102, pp. 311–323, 2001.

As specification logic, we consider a linear time logic built over state predicates that can contain index variables. We restrict ourselves to safety properties; liveness properties of broadcast protocols were shown to be undecidable in EFM99, so decidability of model-checking certainly does not extend to liveness properties for the whole class we characterise.

Related Work. Model checking for parameterized systems has been addressed by many researchers. One line of research is concerned with restrictions that, when imposed on parameterized systems, make model checking decidable for safety properties or even full temporal logic. The systems considered there are either token rings, like in EN95, or broadcast protocols, which were introduced in EN98 and also considered in EFM99; GS92 deals with a restricted form of broadcast protocols. Both types are subsumed by our subclass.

The approach for broadcast systems in EN98 and EFM99 falls under the paradigm of using well-ordered sets for the verification of infinite-state systems ACJT96. All sets that are considered are sets that are upward closed with respect to a well order. This means that all guards of transitions have to describe upward closed sets, which excludes certain global conditions, e.g. the one used in the cache coherence protocol that Delzanno considered in Del00.

Regular model checking, advocated by e.g. KMM$^+$97 and BW98, is based on representing sets of states by regular languages. Termination is obtained by applying some form of acceleration in order to compute the transitive closure of the transition relation BJNT00. Emphasis lies not on detecting decidable classes but providing general, not necessarily exact methods to handle a large class of systems. In this context, handling global guards was considered in ABJN00; our method provably terminates on the examples considered there. The main difference is that is our work is not based on acceleration techniques.

The specification language we use is rather general: The properties considered by EN95 had restricted number of quantified variables, and the properties for broadcast protocols considered in EN98; EFM99 can only express upward closed properties about numbers of processes being in a certain control state.

Overview. First we explain the types of parameterized systems we consider. The third section contains our model checking algorithm for ordinary systems, while in the fourth section it is adapted to parameterized systems, and the conditions under which the algorithm terminates are given. Missing proofs and details can be found in the full version at `http://www.dcs.ed.ac.uk/~monika`.

2 Framework

As program notation we use concurrent state-based guarded-command systems; a system is hence of form $(V, C_1, \ldots, C_n, I)$, where V is a set of variables, C_i are components and I is a predicate describing the initial states. A component consists of a set of transitions, where guards and assignments are built over boolean or enumerative variables or integer terms. More precisely, the terms occurring on

the right-hand side of assignments are terms of Presburger arithmetic, enumerative constants or formulas of Presburger arithmetic[1], depending on the sort of the left-hand side of the assignment, and guards are formulas of *Presburger arithmetic*. The restriction to Presburger arithmetic, i.e. to multiplication only with constants, guarantees decidability. A step is defined by choosing some component and one of its transitions with a guard that is satisfied in the current state, and performing all assignments of this transition simultaneously. As an example of the program notation, consider Table 1, the well-known Bakery algorithm, in a version for 2 components.

Table 1. Program Text for the 2-Component Bakery Algorithm.

V: $c_1, c_2 : \{T, W, C\}$, $n_1, n_2 : \mathrm{NAT}$	
I: $c_1 = T \wedge c_2 = T \wedge n_1 = 0 \wedge n_2 = 0$	
Component 1:	Component 2:
$c_1 = T \longrightarrow \left\langle \begin{array}{l} c_1 := W \\ n_1 := max(n_1, n_2) + 1 \end{array} \right\rangle$	$c_2 = T \longrightarrow \left\langle \begin{array}{l} c_2 := W \\ n_2 := max(n_1, n_2) + 1 \end{array} \right\rangle$
$\begin{array}{l} c_1 = W \wedge \\ (n_2 = 0 \vee n_1 < n_2) \end{array} \longrightarrow \langle\, c_1 := C \,\rangle$	$\begin{array}{l} c_2 = W \wedge \\ (n_1 = 0 \vee n_2 < n_1) \end{array} \longrightarrow \langle\, c_2 := C \,\rangle$
$c_1 = C \longrightarrow \left\langle \begin{array}{l} c_1 := T \\ n_1 := 0 \end{array} \right\rangle$	$c_2 = C \longrightarrow \left\langle \begin{array}{l} c_2 := T \\ n_2 := 0 \end{array} \right\rangle$

2.1 Parameterized Systems

Before describing our notion of parameterized systems, we first have to define their state language.

Definition 1 (Index Predicates). Index terms *are of form $j + k$ or k, where j is an index variable and k is an integer constant.*
Index predicates *are defined as follows:*

- Basic index predicates *are the formulas of Presburger arithmetic without quantification*[2]*, where variables can (but need not) be indexed by index terms.*
- *We say that the index term $j + k$* occurs *in the index predicate p if there is some variable y occurring in p in form $y[j + k]$.*
- *If p is a basic index predicate and $j_1, \ldots, j_n$ are index variables s.t. all index terms occurring in p that contain some j_i have constant 0, then*
$$\forall\, j_1 \ldots \forall\, j_n\, (a \rightarrow p) \text{ and } \exists\, j_1 \cdots \exists\, j_n\, (a \wedge p)$$
are index predicates, where a is a conjunction of expressions of form $j_i \neq k$ or $j_i \neq j + k$ for an index variable j.
- *Index predicates are closed under boolean operations.*

[1] For simplicity, we consider boolean variables and equations between enumerative variables also to be formulas of Presburger arithmetic.
[2] Since the system variables are intended to have finite domain, quantification over integer variables would not add expressive power.

The restriction to unnested quantification is used in the proof of Theorem 1, while the restriction that a quantified variable j can only occur in index terms without constants is necessary for termination of the model checking algorithm, more precisely to guarantee that no new index terms are generated by instantiating quantifiers.

Models of index predicates with respect to n consist of a valuation v for the occurring index variables in $\{0, \ldots, n-1\}$ and of a valuation s for the system variables, where for every indexed system variable y, s defines values for $y[0], \ldots, y[n-1]$. We write $s, v \models_n p$ if s and v form a model for p with respect to n.

A **parameterized system** $S = (V, C[i], I)$ differs from an ordinary system in that the transitions of $C[i]$ are parameterized by the index variable i.[3] Accordingly, some variables of V are indexed, while the others act as global variables. The guards of an *parameterizable component* $C[i]$, are index predicates, but we do not allowed quantification over index variables on the right-hand side of assignments. This guarantees that the predicates generated during model checking remain in the class of indexed predicates. The only index variable appearing freely in transitions of $C[i]$ is i, and on the left-hand side of an assignment we only allow i as index term, without constant, which means that a copy can only modify its own variables. This not necessary but simplifies the computation of weakest preconditions. The initial predicate I is a closed index predicate. A parameterized version of the Bakery algorithm, shown in Table 2, should illustrate the notion of parameterized system.

Table 2. Parameterized Bakery Algorithm.

$V:\ c[i] : \{T, W, C\},\ n[i] : \mathrm{INT}$
$I:\ \forall\, i\, (n[i] = 0 \wedge c[i] = T)$
$c[i] = T \longrightarrow \left\langle \begin{array}{l} c[i] := W \\ n[i] := (max_j n[j]) + 1 \end{array} \right\rangle$
$c[i] = W \wedge \forall j (j \neq i \rightarrow n[i] < n[j] \vee n[j] = 0) \longrightarrow \left\langle c[i] := C \right\rangle$
$c[i] = C \longrightarrow \left\langle \begin{array}{l} c[i] := T \\ n[i] := 0 \end{array} \right\rangle$

For a natural number n, the **instantiation** $S[n]$ **of a parameterized system** S is $(V[n], C[0], \ldots, C[n-1], I[n])$, where $V[n]$ is the set of ordinary variables of V together with, for every indexable variable y in V, $y[0], \ldots, y[n-1]$, and where for a natural number h, $C[h]$ is obtained by replacing i by h in all indexed expressions. All expressions are intended to be *modulo* n. This can be done on

[3] The proposed approach is applicable to systems composed of different parameterizable components and of ordinary components. For simplicity of the presentation, this generalization is omitted.

the syntactic level by replacing all terms and predicates X by X mod n as follows: For a variable y and an index term $i+k$, $y[i+k]$ mod n is $y[(i+k) \bmod n]$; this is extended in the usual way to terms and basic predicates. For quantified predicates, we define $\forall\, j\, (a \rightarrow p)$ mod n to be $\bigwedge_{0 \leq h < n} (a \bmod n \rightarrow p[j := h] \bmod n)$ and $\exists\, j\, (a \wedge p) \bmod n$ *to be* $\bigvee_{0 \leq h < n} (a \bmod n \wedge p[j := h] \bmod n)$.

The interpretation of $(i + k)$ mod n under a given valuation v is the natural one. Note that the instantiation of a parameterized system is an ordinary system since i is the only free index variable occurring in $C[i]$.

The following **decidability result** is crucial for the model checking procedure we present. For any index predicate p and any n, satisfiability of p mod n is decidable since the domains of the free index variables can be restricted to $\{0, \ldots, n-1\}$. Decidability holds since it suffices to check satisfiability for some large enough n, more precisely for n so that all of the index terms and all of the existentially quantified variables can be interpreted with different values. This argument does not hold for nested quantification, since if variable j is existentially quantified in the scope of a universal quantification over i, then for every value for i the possibility of a new value for j has to be considered, but this new value for j must be taken into consideration as value for i, which results in a cycle.

Theorem 1. *Satisfiability of index predicates is decidable.*

In a **specification logic** for parameterized systems, it is desirable to be able to quantify over index variables. For example, the property of mutual exclusion in the Bakery example can be formulated as: $\forall\, i_1 \forall\, i_2\, (i_1 \neq i_2 \rightarrow G\, (c[i_1] \neq C \vee c[j_2] \neq C))$. More generally, we consider formulas of the form: $\forall\, j_1 \cdots \forall\, j_n\, (a \rightarrow \phi)$, where ϕ is an LTL formula with index predicates as state formulas, where $j_1, \ldots, j_n$ are the free index variables occurring in ϕ and where a is a conjunction of inequalities $j_i \neq [i+]k$ for some index variable i. In Mai01 we specified a fragment of LTL for which it is possible to allow universal quantification over index variables anywhere in the formula.

Now we can formally state the **model-checking problem for parameterized systems:**

$$S \models \forall\, j_1 \cdots \forall\, j_m\, (a \rightarrow \phi) \quad \text{if f. a. } n,\ S[n], s \models_n (\forall\, j_1 \cdots \forall\, j_m\, (\rightarrow \phi) \bmod n.$$

2.2 Types of Parameterized Systems

The characterisation we give of parameterized systems for which model checking is decidable concerns mainly the communication between different copies. A possible way of communication is to read the value of a variable of a different copy. This is the case if on the right-hand side of an assignment or in a guard, an expression $y[i+k]$ occurs, where k is not zero, and hence the transition depends of the value of the variable y in the k-th neighbour. An example for this type of communication is the following, a transition from the Dining Philosophers

algorithm.

$$c[i] = h \wedge pr[i] \wedge \neg pr[i-1] \wedge\ c[i-1] \neq e \wedge c[i+1] \neq e \longrightarrow \left\langle \begin{array}{l} c[i] := e \\ pr[i] := \text{false} \end{array} \right\rangle$$

While this general form of communication has to be restricted severely (see Definition 4), other forms of communication, namely that of token rings and broadcast systems are unproblematic.

Both token rings and broadcast systems have a *control-state*, i.e. there is a special variable c of enumerative sort $\mathcal{D}_{control}$ such that the guard of every transition is of form $c[i] = d \wedge p$ for some index predicate p, and contains an assignment $c[i] := d'$.

A **token ring** (EN98) is a control-state component $C[i]$ for which some transitions are marked by *send* and some by *rec*. In all transitions of $C[i]$, only i occurs as index term, so a copy can only read the value of its own variables or that of global variables. With the execution of a *rec* transition, the copy acquires the token, while with a *send* transition, the token is passed to the right neighbour. We require that *send* and *rec* transitions alternate along every path through $C[i]$. Token rings are not executed in a completely asynchronous fashion as the general parameterized systems we consider here: To execute *send* or *rec* transitions, neighbouring copies have to synchronise: For any instantiation with n and some $0 \leq h < n$, a transition in $C[h]$ marked by *rec* can only be executed in parallel with a *send* transition in $C[h-1]$ (or $C[n-1]$ if $h = 0$ resp.), and vice versa. By $\mathcal{D}_{token}$ we denote the set of control states in a token ring in which the copy "has the token", i.e. which are reachable by a sequence transitions such that the last marked transition was a *rec*-transition.

Table 3. Illinois Protocol.

$V:\ c[i] : \{invalid, exclusive, dirty, shared\}$
$I:\ \forall i\,(c[i] = invalid)$
$(read!!), (write!!), (rep!!)\ c[i] = invalid \longrightarrow \left\langle c[i] = invalid \right\rangle$
$(read??)\ c[i] = invalid \longrightarrow \left\langle \begin{array}{l} p \longrightarrow c[i] := shared \\ \neg p \longrightarrow c[i] := exclusive \end{array} \right\rangle$
$\vdots$
p abbreviates the global guard: $\exists j\,(j \neq i \wedge c[j] \neq invalid)$

A **broadcast component** (EN98) is again a control-state component $C[i]$, and communication between copies is only possible in form of synchronisation. But the copy to synchronise with is not determined but can be any other copy that can execute a matching transition. Moreover, a broadcast synchronisation is possible: In such a synchronisation step, all copies execute a transition. More formally, let $\Sigma = \Sigma_{rv} \cup \Sigma_{bc}$ be an *action alphabet*, where Σ_{rv} and Σ_{bc} are disjoint finite sets. The transitions can be marked by $a!$ or $a?$ for $a \in \Sigma_{rv}$, or by $a!!$ or

$a??$ for $a \in \Sigma_{bc}$. The semantics of broadcast systems is as follows: A transition marked by $a?$ in some component can only be executed simultaneously with a transition marked by $a!$ in some other component and vice versa. The broadcast actions $a!!$ can only be executed simultaneously with an action marked by $a??$ in all other components, and an action marked by $a??$ can only be executed in such a situation.

The cache coherence protocol used as example in Del00 can be modelled as broadcast protocol as partly shown in Table 3, and provides an example of a broadcast protocol with global guards.

3 Model Checking by Tree Construction

Model checking of safety linear temporal logic formulas can be restricted to model checking of formulas of form $EF\,p$ for some state predicate p by using an automaton that accepts the bad prefixes for a safety linear temporal logic formula (KV99). The fixed point approach for verifying that S satisfies $EF\,q$ uses the fact that the set of states satisfying $EF\,q$ is the least fixed point of the functional: $F(X) = \{s \mid s \models q\} \cup \neg wp.S.\neg X$.[4] We carry out the fixpoint computation on predicates, i.e. we compute a state predicate which characterises the least fixed point of F, starting with the empty predicate false. The necessary ingredients needed to do this are:

- Decidability of implication and satisfiability for state predicates. The former is needed to decide whether a fixed point has been reached, and the latter for deciding whether an initial state satisfies an expression.
- For a given state predicate p, a state predicate representing the *weakest precondition* [5] of the set of states satisfying p must be computable.

For a program S and a given state predicate p, a predicate $wp.S.p$ that represents $wp.S.\{s \mid s \models p\}$ is easy to define: Let $Tr(C)$ be the set of all transitions of component C, where a transition is of form $g \longrightarrow < v_0 := t_0, \ldots, v_k := t_k >$, then $wp.S.p = \bigwedge_{C \text{ comp. of } S} = \bigwedge_{C \text{ comp. of } S} \bigwedge_{g \longrightarrow < v_0 := t_0, \ldots, v_k := t_k > \in Tr(C)} (g \rightarrow p[v_0 := t_0, \ldots, v_k := t_k])$, where $[v_0 := t_0, \ldots, v_k := t_k]$ denotes simultaneous substitution.

The specific feature of our algorithm is that the fixed point computation is represented in form of a tree over $\mathbb{N}$, i.e. as set of finite sequences over $\mathbb{N}$ that is closed under prefix-formation: The root is denoted by ϵ, and e. g. 00 is the first successor of the first successor of the root. Note that the algorithm works directly on the program notation.

Definition 2 (Proof Tree Construction). *Let q be a predicate of the form $\bigwedge_i p_i$ for literals p_i. The proof tree $\mathcal{T}(EF\,q, S)$ is a tree over $\mathbb{N}$ with labelling $l : \mathcal{T}(EF\,q, S) \longrightarrow \{ \bigwedge_i p_i \mid p_i$ literals $\}$ such that*

[4] The set $\neg wp.S.\neg X$ consists of all states that have at least on successor in X.

[5] For a system S and a set of states X, the weakest precondition $wp.S.X$ is the set of states s of S so that all successors of s lie in X.

– $l(\epsilon) = q$ *and*
– $\bigvee_j l(xj) \Leftrightarrow \neg wp.S.\neg l(x)$.[6]

A node $x \in \mathcal{T}(EF\, q, S)$ *is a leaf if one of the following conditions holds:*

(i) $l(x)$ *is not satisfiable (unsuccessful leaf);*
(ii) $I \wedge l(x)$ *is satisfiable (where* I *is the initial predicate of the system), i.e. there is an initial state of* S *satisfying* $l(x)$ *(successful leaf);*
(iii) *there is a node* $y \in \mathcal{T}(EF\, q, S)$ *such that* $|x| > |y|$ *and* $l(x) \Rightarrow l(y)$ *(unsuccessful leaf).*

The tree $\mathcal{T}(EF\, q, S)$ is *successful* if it has a successful leaf. The following theorem states the correctness of the algorithm.

Theorem 2. *(a)* $S \models AG\,(\neg q)$ *if and only if* $\mathcal{T}(EF\, q, S)$ *is not successful.*
(b) If all sorts of variables of S *are finite, then the construction of* $\mathcal{T}(EF\, q, S)$ *terminates.*

To illustrate the tree construction, the first steps of the proof tree construction for the 2-component Bakery algorithm are shown in Figure 1. For the root, $\neg wp.S.\neg(c_1 = C \wedge c_2 = C)$ is $(c_1 = W \wedge (n_1 < n_2 \vee n_2 = 0) \wedge c_2 = C) \vee (c_1 = C \wedge (n_2 < n_1 \vee n_1 = 0) \wedge c_2 = W)$, and by transforming this into disjunctive normal form, the four successors of the root node are obtained.

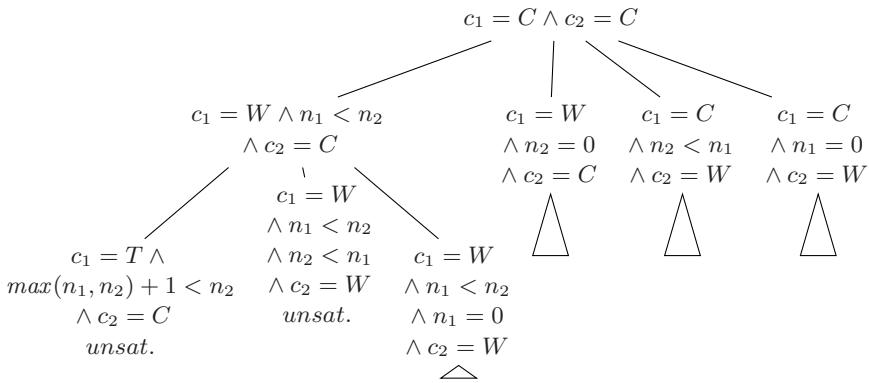

Fig. 1. Proof Tree for the 2-Component Bakery Algorithm.

An advantage of using a tree structure to represent the least fixed point is that the predicates labelling the nodes are relatively small and automatically only elements of the frontier set are considered for the next iteration step. This distinguishes the approach from other approaches to use Presburger arithmetic for representing sets of states during the fixed point iteration BGP97 and makes it easier to check the Conditions (i), (ii) and (iii). Producing counterexamples is easy since the paths in the tree form potential refutation sequences.

[6] It is irrelevant for the correctness which representation in disjunctive normal form is chosen.

4 Model Checking for Parameterized Systems

4.1 Adaption of the Tree Construction to Parameterized Systems

The first of the requirements on page 317 for the proof tree construction is guaranteed by Lemma 1. It remains to explain the computation of $\neg wp.S.\neg p$ for a parameterized system S. For an ordinary parameterized system S and a given instantiation with n, and for an index predicate p without free index variables, $wp.S[n].p$ can be define as on page 317. This easily generalizes to token rings or a broadcast systems: All possibilities of synchronisation have to be considered.

For the computation of $wp.C[t].p$, where $C[i]$ is the component of S, t is an index term, and p is an index predicate, it has to be clear which indexed variables are modified by a transition of $C[t]$. So if p contains other index terms t', either $t \neq t'$ or $t = t'$ has to be assumed to be able to compute $wp.C[t].p$. The obvious solution is to consider all possibilities of dividing the index terms in p in those that are equal to t and those that are not equal to t, and to compute $wp.C[t].p$ for each of those possibilities. To make this precise, we need some notation. Let $t_1, \ldots, t_n$ be the free index terms occurring in p. For an ordinary parameterized system, we define

- $G(p) \stackrel{\text{def}}{=} \{Equ(t_{i_1}, \ldots, t_{i_k}) \mid \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}\}$, where
- $Equ(t_{i_1}, \ldots, t_{i_k}) \stackrel{\text{def}}{=} t_{i_1} = t_{i_2} \wedge \cdots \wedge t_{i_1} = t_{i_k} \wedge \bigwedge_{r \in \{1,\ldots,n\} \setminus \{i_1,\ldots,i_k\}} t_{i_1} \neq t_r$, [7],
- $Equ(\emptyset) = \bigwedge_{1 \leq r \leq n} i^* \neq t_r$, where i^* is a new index variable.

For an element $g \in G(p)$, let $rep(g)$ be t_{j_1} or i^* respectively. For token rings, in addition sets of index terms that equal $rep(g)+1$ or $rep(g)-1$ have to be selected, and for broadcast systems, a set of index terms standing for the component that is chosen for synchronisation has to be chosen.

For $g \in G(p)$, by $p\langle g \rangle$ we denote the result of replacing all index terms that are by g equal to $rep(g)$ by $rep(g)$ throughout p, and by replacing quantified subexpressions as follows: $(\forall\, j\, (a \rightarrow p))\langle g \rangle = (a \rightarrow p)[j := rep(g)]\langle g \rangle \wedge \forall\, j\, ((a \wedge j \neq rep(g) \rightarrow p)\langle g \rangle)$, and the definition for existential quantification is accordingly, using disjunction.

By $wp^{simp}.C[t].p$ we denote the result of computing the weakest precondition of p (e.g. by the formula given on page 317) by considering variables with index terms syntactically different from t to be not modifiable by $C[t]$. The following lemma states that $wp.S[n].p$ can be represented by computing wp^{simp} for all elements of $G(p)$. Note that $p[\{j := v(j) \mid j \text{ free index variable in } p\}] \bmod n$ does not contain free index variables and can hence be considered as ordinary predicate over the system variables of $S[n]$.

Lemma 1. *For all n, states s and valuations v with respect to n:*
For all valuations w of the index variables introduced in $G(p)$,

$$s, v + w \models_n (\textstyle\bigwedge_{g \in G(p)} g \rightarrow wp^{simp}.C[rep(g)].p\langle g \rangle) \bmod n$$
$$\iff s, v \models_n wp.S[n].(p[\{j := v(j) \mid j \text{ free index variable in } p\}] \bmod n).$$

[7] Note that $i + k = i + k'$ can be satisfiable in some instantiation for even if $k \neq k'$.

Lemma 1 implies that for parameterized systems, a proof tree can be constructed generically for all instantiations $S[n]$ by computing

$$\neg wp.S.\neg q = \bigvee_{g \in G(q)} (g \wedge \neg wp^{simp}.C[rep(g)].\neg q\langle g \rangle).$$

The key property is the following: There is some n and s, v with respect to n, and a valuation w for the new index variables in $G(q)$ such that $s, v{+}w \models_n \neg wp.S.\neg q$, iff there is a successor s' of s in $S[n]$ such that $s', v \models_n \neg q$.

So we can now define the adaption of the proof tree construction for parameterized systems. Note that there is only one additional termination conditions, which only applies to token rings.

Definition 3 (Proof Tree Construction for Parameterized Systems). *Let p be an index predicate. $\mathcal{B}^{para}(EF\,p, S)$ is a tree over $\mathbb{N}$ with labelling $\bigwedge_i p_i$, where p_i is either a quantifier-free index predicate which is a literal, or a quantified index predicate.*

- $l(\epsilon) = p$ *and*
- $\bigvee_j l(xj) = \neg wp.S.\neg l(x)$.

A node $x \in \mathcal{B}^{para}(EF\,p, S)$ is a leaf if one of the following holds:

(i) *$l(x)$ is not satisfiable (unsuccessful leaf);*
(ii) *$I \wedge l(x)$ is satisfiable (successful leaf);*
(iii) *there is a node $y \in \mathcal{B}^{para}(EF\,q, S)$ such that $|x| > |y|$ and $\exists\, l\,(x) \Rightarrow \exists\, l\,(y)$, where $\exists\_$ denotes existential quantification over all free index variables (unsuccessful leaf).*
(iv) *For a* token ring component *$C[i]$: Let $i + k_1, \ldots, i + k_r$ be all index terms containing i that occur in p (note that in $C[i]$, only i occurs as index term) and let k_1 and k_r be maximal resp. minimal among $\{k_1 \ldots, k_r\}$. Then all labels containing $i + k_r - 2$ as index term, are leaves (unsuccessful leaf). Furthermore, all labels that have more than one token are leaves, i.e. those containing literals of form $c[t] = d$ and $c[t'] = d'$ for $d, d' \in D_{token}$ and for two index terms t and t' such that $t \neq t'$, where c is the control-state variable of $C[i]$ (unsuccessful leaf).*

Note that the existential quantification in Condition (iii) can be applied since I is a closed index predicate. The reason why Condition (iv) is correct is that due to the restricted communication, if a label contains an expression $P[i + k_r - k]$ that contains $i + k_r - k$ as index term for some $k > 2$, then there also is a node that does not differ in the expressions that contain $i + k_1, \ldots, i + k_r$ as index terms and besides that only contains $P[i + k_r - 2]$. It follows that if the former node is satisfiable, then already the latter is satisfiable, under the additional requirement that for token rings, the initial state is uniform in all indices execpt 0. Furthermore, due to the synchronisation, labels that contain index terms $i + k_1 + k$ for some $k > 1$ necessarily contain several tokens.

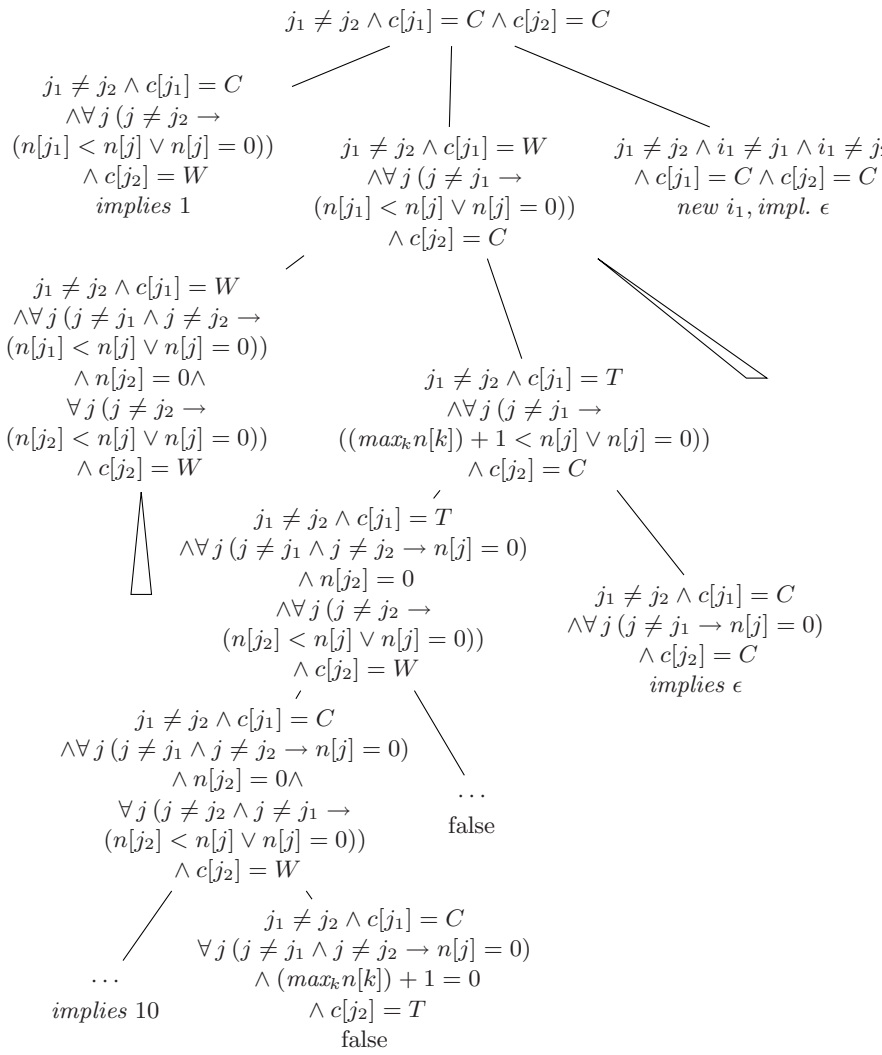

Fig. 2. Proof Tree for the Parameterized Bakery Algorithm.

Figure 2 gives an example for the proof tree construction for a parameterized system. The first successor of the root represents the condition that has to hold if $C[j_2]$ takes a step, the second condition is the case that $C[j_1]$ takes a step, and the third successor is the dual of the weakest precondition for the case that $C[i_1]$, where i_1 is a new variable, takes a step. Since this is obviously only necessary if there are global variables (which could be modified by $C[i_1]$), this step is omitted in the rest of the tree. Only parts of the tree are displayed, while the full tree has 31 nodes. The full paper also discusses the proof tree construction for the Illinois protocol.

4.2 Parameterized Systems for which Model Checking of Safety Properties Is Decidable

By exploring under which conditions the proof tree construction terminates, we obtain a characterisation of parameterized systems for which model checking of safety properties is decidable. The main observation is that Condition (iii) holds eventually along a path of the proof tree if for a given index variable j, only *finitely many* index terms of form $j + k$ occur in labels of the proof tree. This holds for broadcast systems and token rings (for the latter due to Condition (iv)), but not for parameterized systems $C[i]$ in which expressions of form $i+k$ for $k \neq 0$ occur in guards of in right-hand sides of assignments as in the example transition on page 316. The following restriction however guarantees this property.

Definition 4 (Component with bounded communication). *A component* $C[i]$ *has* unbounded communication *if there is a sequence of indexed variables* $y_0, y_1, \ldots, y_n$ *such that* $y_0 = y_n$*, and for all* $j < n$*, there is a transition* tr *of* $C[i]$ *that contains an assignment* $y_j[i] := t$ *in* $C[i]$ *such that for some integer constant* k_j*,* $y_{j+1}[i + k_j]$ *occurs in guard*(tr) *or in* t*, and* $k_j \neq 0$ *for at least one* j*.*

We can now define the subclass of parameterized systems for which our model checking algorithm terminates:

Definition 5 (Terminating parameterized system). $C[i]$ *is a* terminating parameterized component *if*

1. *it does not have unbounded communication, or*
2. *it is a token ring, where the initial predicate must be uniform for all indices except the index* 0*,*
3. *it is a broadcast component.*

The proof tree construction terminates for terminating parameterized systems if the domains of system variables are finite: Condition (iii) must apply eventually along a path of the proof tree: A label is determined by choosing a set of literals and a set of quantified expressions, and a set of index terms that occur in the label. There are two types of index variables, those occurring in the property (only finitely many) and those introduced by $G(p)$. The number of the latter is not bounded but the number of different such index variables within one label is bounded by the maximal iteration depth of quantifiers occurring in the property or in some guard plus 1. As argued above, the number of different index variables also bounds the number of different index terms. So up to equivalence after existentially quantifying over index variables (which is used in Condition (iii)), there are only finitely many possibilities for labels.

We can summarise the properties of the extended proof tree construction in the following theorem. By using an automaton representing bad prefixes for ψ, this extends to formulas $\forall\, j_1 \ldots \forall\, j_m\, (a \rightarrow \psi)$ with ψ an LTL safety-formula.

Theorem 3. *(a)* $S \models \forall\, j_1 \ldots \forall\, j_m\, (a \rightarrow AG\,(\neg p))$ *iff* $\mathcal{B}^{para}(EF\, p, S)$ *is not successful.*

(b) If all system variables of S *have finite domains, then the construction of* $\mathcal{B}^{para}(EF\, p, S)$ *terminates.*

References

ABJN00. Abdulla, P. A., Bouajjani, A., Jonsson, B. and Nilsson, M. *Handling global conditions in parameterized system verification.* In: *Proc. 12th Intl. Conf. on Computer Aided Verification.* 2000, LNCS 1855.

ACJT96. Abdulla, P., Cerans, K., Jonsson, B. and Tsay, Y.-K. *General decidability theorems for infinite-state systems.* In: *Proc. 11th Symp. Logic in Computer Science.* 1996.

BGP97. Bultan, T., Gerber, R. and Pugh, W. *Symbolic model checking of infinite state systems using Presburger arithmetic.* In: *Proc. 9th Intl. Conf. on Computer Aided Verification.* 1997, LNCS 1254.

BJNT00. Bouajjani, A., Jonsson, B., Nilsson, M. and Touili, T. *Regular model checking.* In: *Proc. 12th Intl. Conf. on Computer Aided Verification.* 2000, LNCS 1855.

BW98. Boigelot, B. and Wolper, P. *Verifying systems with infinite but regular state space.* In: *Proc. 10th Intl. Conf. on Computer Aided Verification.* 1998.

Del00. Delzanno, G. *Automatic verification of parameterized cache coherence protocols.* In: *12th Intl. Conf. on Computer Aided Verification.* 2000, LNCS 1855.

EFM99. Esparza, J., Finkel, A. and Mayr, R. *On the verification of broadcast protocols.* In: *Proc. 14th Symp. Logic in Computer Science.* 1999.

EN95. Emerson, E. A. and Namjoshi, K. S. *Reasoning about rings.* In: *Proc. 22th ACM Conf. on Principles of Programming Languages.* 1995.

EN98. Emerson, E. A. and Namjoshi, K. S. *On model checking for nondeterministic infinite-state systems.* In: *Proc. 13th Symp. Logic in Computer Science.* 1998.

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

KMM+97. Kesten, Y., Maler, O., Marcus, M., Pnueli, A. and Shahar, E. *Symbolic model checking with rich assertional languages.* In: *Proc. 9th Intl. Conf. on Computer Aided Verification.* 1997, LNCS 1254.

KV99. Kupferman, O. and Vardi, M. Y. *Model checking of safety properties.* In: *Proc. 11th Intl. Conf. on Computer Aided Verification.* 1999, LNCS 1633.

Mai01. Maidl, M. *Temporal logic extended by universal quantifiers*, 2001. Presented at AVIS 01 Workshop in Berlin.

Suz88. Suzuki, I. *Proving properties of a ring of finite state machines.* Information Processing Letters, 28: 213–214, 1988.