FI MU

Faculty of Informatics
Masaryk University Brno

Model Checking of Control-User Component-Based Parametrised Systems

by

Pavlína Vařeková
Ivana Černá

FI MU Report Series FIMU-RS-2008-06

July 2008

Publications in the FI MU Report Series are in general accessible via WWW:

http://www.fi.muni.cz/reports/

Further information can be obtained by contacting:

Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic

Model Checking of Control-User Component-Based Parametrised Systems

Pavlína Vařeková* Ivana Černá†
Faculty of Informatics, Masaryk University,
Botanická 68a, 60200 Brno,
Czech Republic.
{xvarekova1,cerna}@fi.muni.cz

July 15, 2008

Abstract

Many real component-based systems, so called Control-User systems, are composed of a stable part (*control component*) and a number of dynamic components of the same type (*user components*). Models of these systems are parametrised by the number of user components and thus potentially infinite. Model checking techniques can be used to verify only specific instances of the systems. This paper presents an algorithmic technique for verification of safety interaction properties of Control-User systems. The core of our verification method is a computation of a *cutoff*. If the system is proved to be correct for every number of user components lower than the cutoff then it is correct for any number of users. We present an on-the-fly model checking algorithm which integrates computation of a cutoff with the verification itself. Symmetry reduction can be applied during the verification to tackle the state explosion of the model. Applying the algorithm we verify models of several previously published component-based systems.[1]

*The author has been supported by the grant No. 1ET400300504.
†The author has been supported by the grant No. 1ET408050503.
[1]Full version of the paper presented at the Conference on Component-Based Software Engineering 2008 (CBSE2008).

1 Introduction

Model-checking [11] is a formal verification technique which has received a wide attention in industry as it can be used to detect design errors early in the design life-cycle. Model checking is based on state space generation and as such can be directly applied to finite state systems. In case of infinite state systems more involved techniques have to be employed.

Component-based software development is an alternative to existing software development techniques. Component-based development proposes to assemble software systems from reusable components, which helps to significantly reduce development time and costs. On the other hand, interaction among components opens new issues relevant to the correctness of interaction.

An extensive study of component-based systems reveals that many systems are composed of a beforehand unknown number of components. A typical situation is the composition of one fixed component (control component) with an unknown number of identical components (user components). These systems are usually called Control-User systems. Formal verification of Control-User systems includes verification of every possible composition of the control component with any number of user components. Even if the composition of the control component with a specific number of user components is finite, the verification task itself is infinite.

It has been observed [14] that reachability properties are the most common properties arising in verification. Reachability is closely related to safety, expressing that no *unsafe* state is reachable in a system. This remains true also for Control-User systems.

In this paper we aim to use existing approaches to build-up two reachability verification algorithms. Our verification method is based on *cutoffs* [15, 19]. If a system is proved to be correct for every number of user components lower than the cutoff then it is correct for any number of users. First of the two introduced algorithms is suitable for efficient verification of a finite set of reachability properties. The algorithm iteratively computes the minimal cutoff and during the computation it finds a finite set of representatives of all reachable states. Representatives bear all information needed for deciding reachability and thus to verify given properties of the whole Control-User system it is enough to check these representatives. Advantages of the verification algorithm are that it works with the minimal cutoff and allows to verify many properties at the same time. Evaluation of the algorithm reveals that the cutoff is typically rather small and thus the verification itself is efficient. The experimental studies were conducted on several previously published case studies as well as on a tailored Control-User system.

The second (bounding) algorithm is proposed for computing the highest possible number of users which are simultaneously in the same state (part of a computation). The bounding algorithm provides answers to questions like *What is the maximal number of users simultaneously requiring the same services?* The number of users is called the *bound*. The algorithm has two main parts. In the first part the algorithm finds a candidate for the bound and proves that the bound is less or equal to the candidate. In the second part it computes the exact value of the bound. Experiments demonstrate that typically in the first part the algorithm efficiently finds a candidate which is equal to the bound. In both algorithms the effectiveness can be supported by symmetry reduction over the reachable state space.

The paper is structured as follows. Section 2 presents a model of Control-User system while Section 3 introduces several types of reachability properties. Section 4 highlights backward reachability for C-U systems. The algorithm for verification of reachability properties is described in Section 5 and its evaluation is in Section 6. The bounding algorithm and its evaluation can be found in Section 7. Related work is summarised in Section 8.

2 The Control-User System Model

We consider a class of parametrised systems where a system consists of a unique component - in the literature called the control component [21, 17] and an arbitrary number of components with an identical model - user components. An example of such a system with n users is in Figure 1 a). Components are executing concurrently with the interleaving semantics, capturing that a component can communicate with another component using the pairwise rendezvous synchronisation (a component can send a message iff the receiver is enabled).

As the formal model of a Control-User system we use a labelled Kripke structure (LKS) [9]. An LKS is a structure underlying many other formalisms capturing interactions between components like I/O automata [25], Component-Interaction Automata [34], or Extended Behavior Protocols [24].

Definition 2.1 (LKS)**.** *A* labelled Kripke structure *(or* LKS *for short) is a tuple* $(Q, I, Ap, L, \Sigma, \delta)$ *where* Q *is a set of* states, $I \subseteq Q$ *is a set of* initial states, Ap *is a set of* atomic propositions, $L : Q \to 2^{Ap}$ *is a* state-labelling function, Σ *is a finite set of* actions *and* $\delta \subseteq Q \times \Sigma \times Q$ *is a* transition relation.

We suppose that $\Sigma = \Sigma_{int} \cup \Sigma_{out} \cup \Sigma_{inp}$, where $\Sigma_{out} = \Sigma'_{out} \times \{!\}$, $\Sigma_{inp} = \Sigma'_{inp} \times \{?\}$. The alphabets Σ_{out} resp. Σ_{inp} represent output resp. input actions which can be used for pairwise rendezvous communication between LKSs. The alphabet Σ_{int} represents internal actions. We write $q \to q'$ if there is a label $l \in \Sigma$ such that $(q, l, q') \in \delta$, $\to^*$ is the transitive and reflexive closure of $\to$. A state q is reachable iff $in \to^* q$ for some $in \in I$. Let $q = (q_0, \cdots, q_n)$ be an $n+1$-tuple. Then $pr_i(q)$, $i = 0, \ldots, n$, denotes its $i+1$-th projection, $pr_i(q) = q_i$.

In this paper we restrict ourselves to systems in which the models of the control and user component are finite and have one initial state. Thus LKSs $C = (Q_C, \{in_C\}, Ap_C, L_C, \Sigma_C, \delta_C)$, $U = (Q_U, \{in_U\}, Ap_U, L_U, \Sigma_U, \delta_U)$ form a *Control-User model* (or *C-U model* for short) iff Q_C, Q_U, Σ_C, and Σ_U are finite.

Definition 2.2 (Composition)**.** *Let* $n \in \mathbb{N}_0$ *and for each* $i = 0, \ldots, n$ *let* $K_i = (Q_i, I_i, Ap_i, L_i, \Sigma_i, \delta_i)$ *be an LKS. Then* $K_0 \| \ldots \| K_n$ *denotes the asynchronous composition of* $K_0, \ldots, K_n$ *and is defined as the LKS*

$$(Q_0 \times \cdots \times Q_n,\ I_0 \times \cdots \times I_n,\ (Ap_0 \times \{0\}) \cup \cdots \cup (Ap_n \times \{n\}),\ L,\ \Sigma,\ \delta),$$

where the state-labelling function L *assigns to each state* $(q_0, \ldots, q_n)$ *the set* $\bigcup_{i=0}^{n} L_i(q_i) \times \{i\}$. *The alphabet* $\Sigma = \bigcup_{i=0}^{n} \Sigma_{i,int} \cup \{l \mid l \in \bigcup_{i=0}^{n} \Sigma'_{i,inp} \wedge l \in \bigcup_{i=0}^{n} \Sigma'_{i,out}\}$. *The transition relation* δ *is defined by the prescription* $(q, l, q') \in \delta$ *iff any of the next possibilities holds*

- $\exists 0 \leq i \leq n : \forall 0 \leq j \leq n, j \neq i : pr_j(q) = pr_j(q')$, *and* $(pr_i(q), l, pr_i(q')) \in \delta_i$,
- $\exists 0 \leq i, i' \leq n, i \neq i' : \forall 0 \leq j \leq n, i \neq j \neq i' : pr_j(q) = pr_j(q')$, $(pr_i(q), (l, ?), pr_i(q')) \in \delta_i$, *and* $(pr_{i'}(q), (l, !), pr_{i'}(q')) \in \delta_{i'}$.

A C-U system with n clients is modelled as the composition of $n+1$ LKSs where the first LKS stands for the control component while the others are identical and represent the users. A Control-User system with arbitrary many clients is modelled as the union of LKSs modelling systems with n clients, for all $n \in \mathbb{N}$.

Definition 2.3 ($C\|U^n$, $C\|U^\infty$)**.** *Let* C, U *be a C-U model,* $n \in \mathbb{N}$. *Then* $C\|U^n$ *denotes the composition* $C\|U\| \cdots \|U$ *of* C *and* n *copies of* U. $C\|U^\infty$ *is an infinite state LKS defined* $C\|U^\infty = (\bigcup_{n\in\mathbb{N}} Q_{C\|U^n}, \bigcup_{n\in\mathbb{N}} \{in_{C\|U^n}\}, \bigcup_{n\in\mathbb{N}} Ap_{C\|U^n}, \bigcup_{n\in\mathbb{N}} L_{C\|U^n}, \bigcup_{n\in\mathbb{N}} \Sigma_{C\|U^n}, \bigcup_{n\in\mathbb{N}} \delta_{C\|U^n})$.

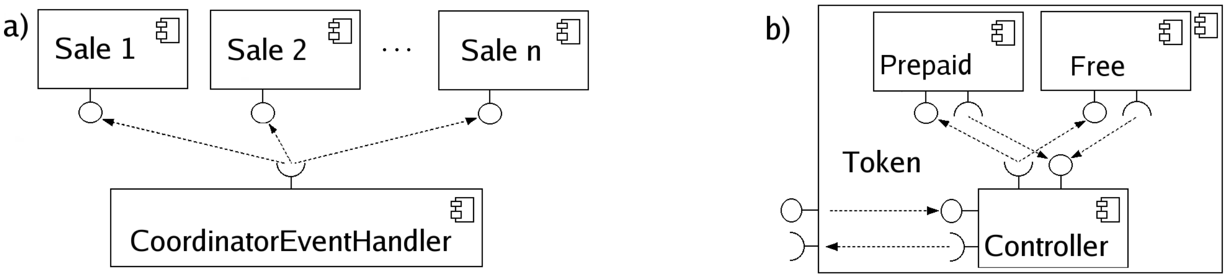

Figure 1: a) CoordinatorEventHandler with n Sales, b) Component Token.

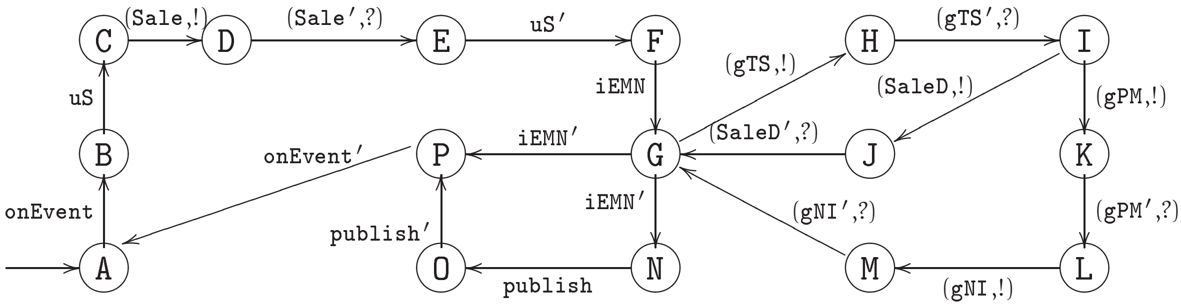

Figure 2: LKS C_{ex} modelling *CoordinatorEventHandler*.

The states of $C \| U^n$ or $C \| U^\infty$ are called *global states* while the states of C or U are called *local states*.

Example 2.4 (Coordinator System)**.** *As a running example we present a part of the* Common Component Modelling Example (CoCoME) *system [34].* Coordinator *is a part of the system that is used for managing express checkouts. For this purpose,* Coordinator *keeps a list of sales that were done within the last 60 minutes and decides whether an express cash desk is needed.* Coordinator *consists of two types of sub-components:* CoordinatorEventHandler *(control component) and* Sale *(user) see Figure 1 a). Anytime a new sale arrives,* CoordinatorEventHandler *creates a new instance of the* Sale *component and displays it in the list. Whenever a sale represented by an instance expires,* CoordinatorEventHandler *removes the instance from the list which causes its destruction.*

We use the models of CoordinatorEventHandler *and* Sale *as presented in [33]. In the models we abbreviate the name of the method* `getNumberofItems()` *to* `gNI`*,* `getPaymentMode()` *to* `gPM`*,* `getTimeofSale()` *to* `gTS`*,* `updateStatistics()` *to* `uS`*, and* `isExpressModeNeeded()` *to* `iEMN`*.*

The control component C_{ex} is depicted in Fig. 2. Its set of atomic propositions corresponds to the set of labels, $Ap_{C_{ex}} = \Sigma_{C_{ex}}$. For a state $q \in Q_{C_{ex}}$ the set $L_{C_{ex}}(q)$ contains all labels which are enabled in the state. For example $L_{C_{ex}}(\mathtt{A}) = \{\mathtt{onEvent}\}$ and $L_{C_{ex}}(\mathtt{I}) = \{(\mathtt{gMP}, !), (\mathtt{SaleD}, !)\}$.

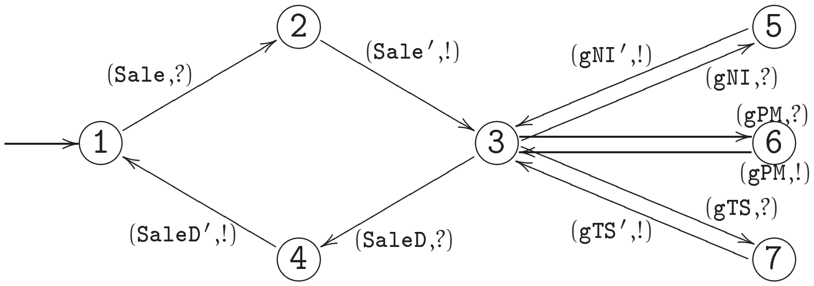

Figure 3: LKS U_{ex} modelling *Sale*.

The user component U_{ex} *is depicted in Fig. 3. Its atomic propositions are* $\mathrm{Ap}_{\mathrm{U}_{ex}} = \Sigma_{\mathrm{U}_{ex}} \cup \{activated, served\}$. *For* $q \in \{1, \dots, 7\}$ *the set* $L_{\mathrm{U}_{ex}}(q)$ *contains labels which are enabled in the state, for* $q \in \{3, 5, 6, 7\}$ *it moreover contains the atomic predicate activated, and for* $q \in \{2, 4, 5, 6, 7\}$ *the set* $L_{\mathrm{U}_{ex}}(q)$ *in addition contains served. For example* $L_{\mathrm{U}_{ex}}(2) = \{(\mathtt{Sale}', !), served\}$ *and* $L_{\mathrm{U}_{ex}}(7) = \{(\mathtt{gTS}', !), activated, served\}$.

Note 2.5. Note that systems with more than one type of users can be modelled as C-U models as well. More precisely, the C-U model can represent an arbitrary system with a control part and a finite number of distinct types of users. The model of a user is an LKS which in the initial state non-deterministically chooses one of the given behaviours and after that it behaves like the choosen type of user. For example in the model of *Token and its support* (part of the prototype implementation of a payment system for public Internet on airports [29]) a Token (client) first chooses whether it will behave as a client with prepaid or free access to Internet (for model of the component see Figure 1 b)).

3 l-symmetric Reachability Properties

We concentrate on verification of reachability properties of C-U models. Properties of interesting states are expressed as formulae of a propositional logic. Formulae are defined over a set of atomic propositions with the help of standard Boolean operators $\wedge, \vee, \neg$. A propositional formula is interpreted over a state of an LKS. A formula is *true* in a state iff after evaluating all atomic proposition assigned to the state as *true* and all others as *false* the result formula is *true*. In the following text we use a standard shortcut $\bigvee_{i \in \emptyset} \psi_i \equiv \mathit{false}$.

A reachability property (or RP for short) is a property capturing that a state satisfying a given propositional formula is reachable in the model $C \| U^n$ for some n. The *general reachability problem* for C-U models is formulated as:

Instance (reachability):
— *C-U model* C, U
— *sequence of formulae* $\{\varphi_n\}_{n\in\mathbb{N}}$*, where* φ_n *is a formula of the propositional logic over atomic propositions* $L_C\times\{0\}\cup L_U\times\{1\}\cup\cdots\cup L_U\times\{n\}$*.*
Problem: *Is there* $n\in\mathbb{N}$ *such that a state satisfying* φ_n *is reachable in* $C\|U^n$*?*

In the paper we are interested only in special types of reachability properties which make no distinction among users – so called *0-symmetric RP*, *1-symmetric RP*, etc. For a fixed $l\in\mathbb{N}_0$ and any $n\in\mathbb{N}$, an l-symmetric RP guarantees that if a state $q\in Q_{C\|U^n}$ satisfies φ_n, then there are l users which together with the control component ensure that the state q satisfies φ_n. An instance of the l-symmetric reachability problem is:
Instance (l-symmetric reachability):
— *C-U model* C, U*, number* $l\in\mathbb{N}_0$
— *sequence of* l*-symmetric formulae* $\{\varphi_n\}_{n\in\mathbb{N}}$*, where for each* $n\in\mathbb{N}$*:*

$$\varphi_n=\bigvee_{f:\{1,\ldots,l\}\to\{1,\ldots,n\}}\psi_{(1,f(1)),\ldots,(l,f(l))}.$$

Here ψ is a formula of the propositional logic over atomic propositions defined $L_C\times\{0\}\cup L_U\times\{1\}\cup\cdots\cup L_U\times\{l\}$, f is an injective function, and $\psi_{(1,f(1)),\ldots,(l,f(l))}$ is the formula which results from ψ if we substitute each atomic proposition (a,i) by $(a,f(i))$ leaving $(a,0)$ untouched. We say that ψ is the propositional formula underlying $\{\varphi_n\}_{n\in\mathbb{N}}$.

Example 3.1. l*-symmetric RPs of the C-U model* C_{ex}, U_{ex} *described in Example 2.4 are e.g. properties describing reachability of a state satisfying:*

1. Sale can send an event but CoordinatorEventHandler is not ready to accept it. *It is a 1-symmetric RP described by the sequence* $\{\varphi_n\}_{n\in\mathbb{N}}$ *with the underlying formula* $\psi=\bigvee_{act\in\{\mathtt{Sale'},\mathtt{SaleD'},\mathtt{gNI'},\mathtt{gMP'},\mathtt{gTS'}\}}((act,!),1)\wedge\neg((act,?),0)$.
2. Two Sales are able to send a response `Sale'` to CoordinatorEventHandler at the same time. *It is a 2-symmetric RP with the underlying formula* $\psi=((\mathtt{Sale'},?),0)\wedge((\mathtt{Sale'},!),1)\wedge((\mathtt{Sale'},!),2)$.
3. At least m Sales can be activated simultaneously. *It is an* m*-symmetric RP with the underlying formula* $\psi=(activated,1)\wedge\cdots\wedge(activated,m)$.
4. CoordinatorEventHandler can service (at least) m activated Sales simultaneously. *It is an* m*-symmetric RP with the underlying formula* $\psi=(served,1)\wedge(activated,1)\wedge\cdots\wedge(served,m)\wedge(activated,m)$.

4 Backward Reachability

Backward reachability is one of the methods used in verification of parametrised systems for checking reachability of critical states [4, 22, 23]. For a given LKS S and a set of its states Q the question is whether a state from Q is reachable in the structure S. For iteratively increasing values j, one generates the set of states from which Q can be reached by a sequence of transitions of the length at most j. The backward reachability procedure terminates in the first iteration where the generated set of states does not increase comparing to the set generated in the previous iteration or if the last generated set of states contains an initial state. For transition systems with an infinite number of states termination of backward reachability is not guaranteed. However, the termination can be guaranteed for special sets Q.

Lemma 4.1. *Let C, U be a C-U model, $A \subseteq Q_C$. Then the backward reachability in $C\|U^\infty$ starting with $Q = \bigcup_{i\in\mathbb{N}} A \times \underbrace{Q_U \times \cdots \times Q_U}_{i}$ terminates.*

Proof: For proof see Appendix.

5 Verification Algorithm

In this Section we present an algorithm which verifies a given l-symmetric RP. A naïve approach is to check all reachable states of the C-U model (more precisely, all reachable states of the LKSs $C\|U^1, C\|U^2, \ldots$) for validity of the given property. As the number of the reachable states is infinite, we first define a finite number of their representatives.

Definition 5.1 ($l+1$-tuple)**.** *Let C, U be a C-U model, $q \in Q_{C\|U^\infty}$ and $l \in \mathbb{N}_0$. Then an $l+1$-tuple $(q_C, q_1, \ldots, q_l) \in Q_{C\|U^l}$ is* assigned *to the state q iff the local state of C in q is q_C and there are l different users in q with local states $q_1, \ldots, q_l$.*

In a similar way we can assign an $l+1$-tuple to an $(l+i)+1$-tuple t' (t' is a state of $C\|U^{l+i}$).

$l+1$-tuples serve as an abstraction of the C-U model states where only the local states of l chosen users in an arbitrary order are maintained. Observe that for any fixed sequence $\{\varphi_n\}_{n\in\mathbb{N}}$ describing a l-symmetric RP a state q of $C\|U^i$ does not satisfy φ_i if and only if there is an $l+1$-tuple t assigned to q such that t (which is a state of $C\|U^l$) does not satisfy ψ (ψ is the propositional formula underlying $\{\varphi_n\}_{n\in\mathbb{N}}$). Hence if we find all $l+1$-tuples assigned to the reachable states of $C\|U^\infty$ (so called *reachable* $l+1$-tuples of $C \| U^\infty$) we can easily verify validity of the given l-symmetric RP[2]. Thus the core of our verification algorithm is a procedure for finding all reachable $l+1$-tuples of $C \| U^\infty$. As for each $l \in \mathbb{N}_0$ the number of $l+1$-tuples is finite there must exist a number k such that the set of all reachable $l+1$-tuples of $\{C \| U^n\}_{n \in \{l,\dots,k\}}$ is exactly the set of reachable $l+1$-tuples of $C \| U^\infty$. Let us call the number k *cutoff*.

The smallest number which can be a cutoff is the number $L = max(1, l)$. The algorithm which we propose searches for the minimal cutoff and returns all reachable $l+1$-tuples of $C \| U^\infty$. It iteratively traverses all reachable states of $C \| U^L, C \| U^{L+1}, \dots$ In the i-th iteration all $l+1$-tuples assigned to reachable states of $C \| U^{L+i}$ are computed and compared to those computed in the previous iteration. More precisely, as for any j the set of reachable $l+1$-tuples of $C \| U^j$ is a subset of the set of reachable $l+1$-tuples of $C \| U^{j+1}$, it is sufficient to compare their cardinality. Once there is no difference between the two sets, the number $L+i-1$ is the candidate for a cutoff. It can happen that there is an $l+1$-tuple assigned to a state reachable in $C \| U^{L+j-1}$ for some $j > i$ which is not covered yet. Therefore we need to verify whether $L+i-1$ is a cutoff.

To confirm that $L+i-1$ is a cutoff we run backward reachability in $C \| U^\infty$ from the set of its states to which a not yet covered $l+1$-tuple is assigned. If backward reachability finds that some of these states is reachable in $C \| U^\infty$, then the state must be reachable in $C \| U^k$ for some $k > L+i-1$ and the state is not reachable in $C \| U^k$ for $k \leq L+i-1$. Consequently $L+i-1$ is not a cutoff and we start the whole procedure with $L+i$. Otherwise $L+i-1$ is the minimal cutoff; the algorithm returns all reachable $l+1$-tuples of $C \| U^{L+i-1}$.

The pseudo-code of the algorithm is given in Fig. 4. The procedure FIND CUTOFF returns the first number k greater or equal to *Cutoff* such that the sets of reachable $l+1$-tuples of the LKSs $C \| U^k$ and $C \| U^{k+1}$ are the same. The set of all reachable $l+1$-tuples of $C \| U^j$ is monotonically increasing with increasing parameter j and at the same time the set of all possible $l+1$-tuples is finite. These two facts ensure that the procedure terminates.

The procedure BACKWARD REACHABILITY(T) first computes the set containing all states of $C \| U^\infty$ to which an $l+1$-tuple from T is assigned. By iterative searching of predecessors it decides reachability of states from T in $C \| U^\infty$.

Lemma 5.2. *The procedure* BACKWARD REACHABILITY *always terminates.*

[2]In fact, having all reachable $l+1$-tuples of $C \| U^\infty$ we can verify any l'-symmetric RP for $l' \leq l$.

```
proc REACHABLE l + 1-TUPLES(C, U, l)
  All_tuples := all l + 1-tuples
  Cutoff := MAX(l, 1);  Valid_cutoff := false
  while Valid_cutoff = false do
          FIND CUTOFF(Cutoff)
          BACKWARD REACHABILITY(All_tuples \ Reached_tuples)
          if Valid_cutoff = false then Cutoff := Cutoff + 1 fi
  od
  Cutoff = Cutoff − 1;
  return Reached_tuples
```

```
proc FIND CUTOFF(Cutoff)
  k := Cutoff;  Reached_tuples := ∅
  repeat  Old_tuples := Reached_tuples
           Reached_tuples := all l + 1-tuples assigned to reachable states in C||U^k
           if |Old_tuples| ≠ |Reached_tuples| then k := k + 1 fi
    until  |Old_tuples| = |Reached_tuples|
  Cutoff := k − 1
```

```
proc BACKWARD REACHABILITY(T)
  Q := {q∈ Q_{C||U∞}  | an l + 1-tuple assigned to q belongs to T}
  Q′ := ∅;  Reach := false
  while (Q ≠ Q′) ∧ (Reach = false) do
          Q := Q ∪ Q′
          Q′ :=  predecessors of Q in C||U∞
          if Q′ ∩ in_{C||U∞}  ≠ ∅ then Reach := true fi
  od
  if ¬Reach then Valid_cutoff := true fi
```

Figure 4: Algorithm for computing all reachable $l + 1$-tuples.

Lemma 5.3. *Let for a C-U model* C, U *and* $l \in \mathbb{N}_0$ *the following condition is true:*

> *An unreachable* $l{+}1$*-tuple of* $C\|U^\infty$ *is assigned to every unreachable* $(l{+}1){+}1$*-tuple of* $C\|U^\infty$. *(*)*

Then BACKWARD REACHABILITY(T), *where* T *is the set of all unreachable* $l + 1$*-tuples of* $C\|U^\infty$*, terminates after the first iteration.*

Proof: For proof of Lemma 5.2 and Lemma 5.3 see Appendix.

Example 5.4. *Let us inspect the computation of* REACHABLE $l{+}1$-TUPLES*(*C_{ex}*,* U_{ex}*, 1); where* C_{ex}*,*U_{ex} *is the C-U model from Example 2.4.*

In the first iteration of when $\mathsf{Reached\_tuples}_{k=0} = \emptyset$ *while-cycle* FIND CUTOFF*(1) is called and it iteratively computes the set* $\mathsf{Reached\_tuples}$*:*

$$
\begin{aligned}
\mathit{Reached\_tuples}_{k=1} = \quad & \{(\mathtt{x},1),(\mathtt{x},3) \mid \mathtt{x} \in \{\mathtt{A},\mathtt{B},\mathtt{C},\mathtt{G},\mathtt{N},\mathtt{O},\mathtt{P}\}\} \cup \\
& \{(\mathtt{x},3) \mid \mathtt{x} \in \{\mathtt{E},\mathtt{F},\mathtt{I},\mathtt{L}\}\} \cup \\
& \{(\mathtt{D},2),(\mathtt{H},7),(\mathtt{J},4),(\mathtt{K},6),(\mathtt{M},5)\}, \\
\mathit{Reached\_tuples}_{k=2} = \quad & \{(\mathtt{x},1),(\mathtt{x},3) \mid \mathtt{x} \in \{\mathtt{A},\dots,\mathtt{P}\}\} \cup \\
& \{(\mathtt{D},2),(\mathtt{H},7),(\mathtt{J},4),(\mathtt{K},6),(\mathtt{M},5)\}, \\
\mathit{Reached\_tuples}_{k=3} = \quad & \mathit{Reached\_tuples}_{k=2}.
\end{aligned}
$$

After that BACKWARD REACHABILITY*($\mathit{All\_tuples} \setminus \mathit{Reached\_tuples}_{k=3}$) is called.*

Iteration *0:* *Q contains all states of $C_{ex} \| U^{\infty}_{ex}$ to which a tuple from* $T = \{(\mathtt{x},2),(\mathtt{x},4),(\mathtt{x},5),(\mathtt{x},6),(\mathtt{x},7) \mid \mathtt{x} \in \{\mathtt{A},\dots,\mathtt{P}\}\} \setminus \{(\mathtt{D},2),(\mathtt{H},7),(\mathtt{J},4),(\mathtt{K},6),(\mathtt{M},5)\}$ *is assigned,*

Iteration *1:* *Q contains all states of $C_{ex} \| U^{\infty}_{ex}$ to which a tuple from* $T \cup \{(\mathtt{D},2,2),(\mathtt{H},7,7),(\mathtt{J},4,4),(\mathtt{K},6,6),(\mathtt{M},5,5)\}$ *is assigned,*

Iteration *2:* *Q is the same as for the iteration 2.*

Thus after 2 iterations BACKWARD REACHABILITY *terminates and returns that the found cutoff* 2 *is valid, consequently* REACHABLE $l+1$-TUPLES *returns the set $\mathit{Reached\_tuples}_{k=2}$.*

We should stress that even if the algorithm is presented as a procedure for finding all reachable $l+1$-tuples of $C \| U^{\infty}$ it is in fact a verification algorithm. As pointed out at the beginning of this Section, keeping the set of all reachable $l+1$-tuples of $C \| U^{\infty}$ one can decide validity of any l'-symmetric RP for an arbitrary $l' \leq l$.

Note 5.5 (Optimisation). There are several possible optimisations of the algorithm. We list the two most important.

- Symmetry reduction [10] in the algorithm decreases the number of both reachable states and tuples exponentially.
- On-the-fly approach to verification: as soon as an $l+1$-tuple is reached for the first time it is checked for validity of given reachability properties.

Note 5.6 (Verification). There is another important way how to use the presented algorithm, namely verification of an updated (modified) system. When updating a system we usually want to guarantee that the new system satisfies all important properties which the original system satisfies. The problem is that it is hard to enumerate all (important) properties of the original system. In such a case it is profitable to use the given algorithm and compute differences between the reachable $l+1$-tuples in the original and the updated system.

6 Evaluation

In order to test efficiency of the proposed algorithm we use several models of previously published real component-based C-U systems (R_I - R_{IV}), simplified real systems (S_I, S_{II}), and simple C-U systems proposed for evaluation of our algorithm (E_I, E_{II}). The inspected C-U models are: R_I - model of *Coordinator* (Example 2.4), R_{II} - model of *Token and its support* (Note 2.5), R_{III} - model of *Cash desk and its support* (Fractal model in [7]), R_{IV} - model of *Subject - Observer system* with n subjects (published in [32], model of the system [3]). S_I and S_{II} are models of *Comanche Web Server* with a Sequential resp. Multi Thread Scheduler and their clients, published in [1]. E_I is a system where the controller provides 5 services in parallel and users can use the services sequentially. E_{II} is a system where the controller provides 2 services and in all states it can receive or return any request, users use the services sequentially. Detailed description of all models and their characteristics are on the web page [2].

Table on Figures 5 and 6 displays for a given C-U model and a parameter $l \in \{1, 2, 3\}$ the number of states of $C \| U^k$ for maximal k for which the state space is generated in FINDCUTOFF (*States*), the minimal cutoff (*Cutoff*), and the number of iterations of backward reachability (*Iterations*). Based on experimental evaluation we conclude:

1. In all cases the while-cycle in REACHABLE $l+1$ -TUPLES was performed only once - the *Cutoff* computed in FINDCUTOFF was the valid minimal cutoff.

2. For each of the models and every $l \geq 2$ the condition *(*)* holds. It means that for each of the models and every $l \geq 2$ to arbitrary unreachable $l+1$-tuple of $C \| U^\infty$ is assigned an unreachable $2+1$-tuple of $C \| U^\infty$. Thus for each model and an arbitrary $l \leq 2$ BACKWARD REACHABILITY terminates after the first iteration.

3. The minimal cutoff for $l+1$-tuples is typically the minimal cutoff for $0+1$-tuples plus l.

7 Bounding Algorithm

When analysing C-U systems we are often interested in the highest possible number of users which are simultaneously in the same state (situation, part of a computation). For instance, we can ask how many users have started a communication with the control component and have not finished it yet, or how many users are demanding the same

model	l	*States*	*Cutoff*	*Iterations*
R_I	1	144	2	2
	2	332	3	1
	3	748	4	1
R_{II}	1	145 125	4	2
	2	1 091 875	5	1
	3	7 821 875	6	1
R_{III}	1	297 108	4	2
	2	1 706 103	5	1
	3	8 957 952	6	1
R_{IV}	1	$48\,320^n$	4	2
	2	$266\,240^n$	5	1
	3	$1\,425\,920^n$	6	1

Figure 5: Evaluation of the verification algorithm.

model	l	*States*	*Cutoff*	*Iterations*
S_I	1	1 126	2	2
	2	4 910	3	1
	3	20 830	4	1
S_{II}	1	7 514	2	2
	2	142 476	3	1
	3	2 672 672	4	1
E_I	1	4 051	6	2
	2	9 276	7	1
	3	19 080	8	1
E_I	1	69	3	1
	2	245	4	1
	3	312	5	1

Figure 6: Evaluation of the verification algorithm.

service. This can be described by a sequence of reachability properties $\{P_m\}_{m\in\mathbb{N}}$ such that for each m the property $P_m = \{\varphi_n^m\}_{n\in\mathbb{N}}$ is an m-symmetric RP expressing that *It is possible to reach a global state in which at least* m *users are in the same specified setting*.

Example 7.1. *Motivations can be found e.g. in Example 3.1, properties 3 and 4:* What is the maximal number of Sales which can be activated simultaneously? *or* What is the maximal number of activated Sales which can CoordinatorEventHandler service simultaneously?

Lemma 7.2. *Let* $\{P_m\}_{m\in\mathbb{N}}$ *be a sequence where every* P_m *is an* m*-symmetric RP with the underlying formula* ψ_m*. Let the implication* $\psi_j \Rightarrow \psi_i$ *be true for every* $j > i$*. Then if* $C\|U^\infty$ *satisfies* P_j *then it also satisfies* P_i *for each* $i \leq j$. *(**)*

A sequence of RPs satisfying the condition of Lemma 7.2 is denoted *integrated sequence of RPs*. Note that an equivalent to the condition (**) is: if $C\|U^\infty$ does not satisfy P_i then it does not satisfy P_j for any $i \leq j$.

For a given integrated sequence of RPs we propose an algorithm for computing a *bound* which is the number b such that P_b is satisfied and P_{b+1} is not satisfied (if it exists). In practise we are often given a value *Max* and the question is whether the *bound* is at most *Max* and only if this is the case we want to know the exact value of *bound*. The task which we study can be described as

Instance:

– *C-U model* C, U, *number Max* $\in \mathbb{N}$

– *sequence* $\{P_m\}_{m\in\{1,\dots,Max\}}$, *where* P_i *is an* i*-symmetric RP satisfying (**)*

Problem: *Compute*

$$\mathsf{bound} \stackrel{\text{def}}{=} \begin{cases} 0 & \textit{if } P_1 \textit{ is not satisfied}, \\ b & \textit{if } P_b \textit{ is satisfied and } P_{b+1} \textit{ is not satisfied}, 1 \leq b < \mathit{Max}, \\ \mathit{Max} & \textit{if } P_{\mathit{Max}} \textit{ is satisfied}. \end{cases}$$

From the Section 5 it follows that for checking the property P_i from the sequence $\{P_m\}_{m\in\{1,\dots,Max\}}$ it suffice to compute all $i+1$-tuples reachable in $C\|U^\infty$. A trivial bounding algorithm, using the algorithm given in the previous section, first finds all $1+1$-tuples reachable in $C\|U^\infty$ and checks P_1, then it finds $2+1$-tuples reachable in $C\|U^\infty$ and checks P_2, etc. However, this approach is not effective enough, for explanation see item 3 in Section 6. This motivate us to propose another algorithm which instead of

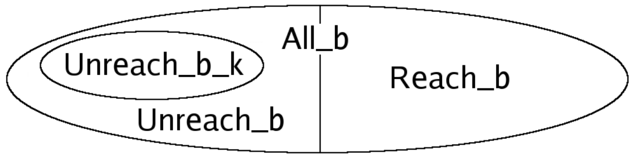

Figure 7: The sets of $b+1$-tuples $\mathtt{All\_b}$, $\mathtt{Unreach\_b}$, $\mathtt{Reach\_b}$, and $\mathtt{Unreach\_b\_k}$.

```
1  proc BOUND(C, U, Max, {P_i}_{1≤i≤Max})
2    k := 0; Unreach_New = ∅
3    repeat k := k + 1
4          Unreach_Old := Unreach_New
5          Reach_New := REACHABLE l + 1-TUPLES(C, U, k)
6          Unreach_New := all k+1-tuples \ Reach_New
7          if IS_k − 1_BOUND() then return k-1 fi
8          Changes = Unreach_New \ GENERATE_POSSIBLE_NEW(Unreach_Old)
9      until (Changes = ∅) ∨ (k ≥ Max)
10   b := LAST_SATISFIED(Unreach_Old)
11   return VALIDATEBOUND(b, Unreach_Old)
```

```
proc IS_k − 1_BOUND
  Old_Satisfied = New_Satisfied;
  New_Satisfied = IS_P_k_SATISFIED(Reach_New)
  if Old_Satisfied ∧ (¬New_Satisfied) then return true
                                       else return false
```

Figure 8: Algorithm for computing the bound.

computing all $i+1$-tuples reachable in $C\|U^\infty$ for each i over-approximate the sets of tuples.

The core idea is that if we have a high integer b, we can choose a small integer k and under-approximate the set $\mathtt{Unreach\_b}$ of all unreachable $b{+}1$-tuples of $C\|U^\infty$ by the set $\mathtt{Unreach\_b\_k}$ of all $b{+}1$-tuples to which is assigned an unreachable $k{+}1$-tuple of $C\|U^\infty$. The profit is that to compute $\mathtt{Unreach\_b\_k}$ instead of $\mathtt{Unreach\_b}$ it is necessary to find all reachable $k+1$-tuples instead of all reachable $b+1$-tuples. The set $\mathtt{Unreach\_b\_k}$ serves also as an over-approximation of all reachable $b+1$-tuples of $C\|U^\infty$. Let us denote $\mathtt{All\_b}$ the set of all possible $b+1$-tuples of $C\|U^\infty$. Then the set of all reachable $b+1$-tuples of $C\|U^\infty$ can be over-approximated $\mathtt{Reach\_b\_k} = \mathtt{All\_b} \setminus \mathtt{Unreach\_b\_k}$ of all $b{+}1$-tuples to which is not assigned an unreachable $k+1$-tuple of $C\|U^\infty$.

The pseudo-code of the algorithm is given in Fig. 8. The algorithm in the first cycle (lines 3-9) iteratively finds the minimal k such that to each unreachable $k+1$-tuple of $C\|U^\infty$ is assigned an unreachable $(k-1)+1$-tuple of $C\|U^\infty$. For every value of $l \leq k$

the algorithm tests whether $l-1 = bound$ (line 7). If the *bound* is greater than $k-1$, then in the next step (line 10) the algorithm computes the maximal $b \in \{k, \dots, Max\}$ such that a $b+1$-tuple from $\mathtt{Reach\_b\_k}$ satisfies the property P_b. The inclusion $\mathtt{Reach\_b\_k} \supseteq \mathtt{Reach\_b}$ implies that the property P_b may but must not be satisfied. On the other hand if $b < Max$ then the property P_{b+1} can not be satisfied. Thus the number b is a maximal value which can be the bound and for us it is a candidate for the bound. Consequently the algorithm computes which of the values $k, \dots, b$ is the bound (line 11).

The procedure REACHABLE $l+1$-TUPLES is described in Section 5. The procedure IS_$k-1$_BOUND computes, using the set of all reachable k-tuples of $C \| U^{\infty}$ *Reach_New*, whether $k-1$ is the bound. GENERATE_POSSIBLE_NEW generates all $k+1$-tuples to which is assigned a $(k-1)+1$-tuple in the input set. The procedure LAST_SATISFIED returns the maximal number b from the set $\{k, \dots, Max+1\}$ such that the intersection of all tuples unsatisfying P_b and $\mathtt{Reach\_b\_k}$ is not empty. The procedure gradually tests $b = k, \dots, Max+1$. The procedure VALIDATEBOUND tests which of the numbers $k, \dots, b$ is the result. If $b = Max$ then it is the valid result, else it firstly checks whether b is the bound and after that it tests $k, k+1$, etc.

Evaluation

As noted in item 2 in Section 6 for each examined model and an arbitrary $k \geq 2$ the condition *(*)* holds. Consequently for any of the presented models and arbitrary b, k, where $b \geq k \geq 2$, equalities $\mathtt{Unreach\_b\_k} = \mathtt{Unreach\_b}$ and $\mathtt{Reach\_b\_k} = \mathtt{Reach\_b}$ hold. Thus the procedure BOUND for all studied models and any sequence of described properties found the correct bound in the while-cycle (lines 3-9) - if it is less then 3. If the bound is greater of equal to 3, than the algorithm computes b as the bound (line 10) and than it successfully verifies, that P_b is satisfied. Our experience with verification of different properties of component-based systems is that a good choice for the value *Max* is $|Q_C| + 1$ as usually if the bound is finite then it is at most $|Q_C|$.

8 Related work

Many papers address parametrised systems and their verification, we relate our contribution to the previously published results in several aspects.

Computational model We consider control-user systems of the form $C \| U^n$ with finite models of C and U. Components communicate using the pairwise rendezvous synchronisation, and there are no variables in the model. Similar models are studied in [6, 8, 21].

Two other approaches to modelling C-U system can be found in the literature. The first one is a model containing a parametrised number of identical finite state components (modelling users) with a finite set of global variables (modelling states of the control component) where an individual user can make a transition if the global variables satisfy a Boolean guard, see [12, 26, 27, 28]. The second approach is to model a C-U system with a finite number of control states (modelling the control component), infinite set of data values $\mathbb{N}_0^{|Q_U|}$ (corresponding to the number of users in each state from Q_U), and appropriate synchronisation, see [4].

Cutoff Verification methods based on a cutoff have been successfully applied to several types of properties of various parametrised systems. In [21] an approach implicitly based on a cutoff was used for proving that verification of l-symmetric reachability properties is decidable but the algorithm runs in triple exponential time and thus it is impractical. Other papers [15, 16, 19, 20] propose algorithms which are more efficient however these are not applicable to our model of control-user systems. In our previous work [31] we studied verification of LTL_X properties using a cutoff. This algorithm is incomplete and for several l-symmetric properties with $l > 0$ does not terminate.

Verification Among a number of approaches to verification of parametrised systems [4, 6, 8, 12, 18, 19, 20, 21, 22, 23, 26, 27, 28] there are several fully automatic techniques which can be used for systems which we study in the paper.

Several of the approaches are based on (backward or forward) reachability analysis [4, 22, 23, 26, 30]. The paper [4] presents verification of reachability properties for general types of systems using backward reachability analysis. The authors prove that for a general type of systems and a special type of reachability properties (including l-symmetric properties) the algorithm terminates, but the number of iterations in which it terminates is not known. The symbolic backward and forward reachability analysis for general rich assertional languages using regular sets and acceleration is performed in [23]. The paper [22] extends [4]. It studies general program model and using a transitive closure generation and acceleration of actions it performs a reachability analysis on those systems. In [26] the author uses a tree based decision procedure which generates predecessors to solve this problem. The main purpose of the paper is to prove a decidability of verification of safety properties for a broad class of systems and thus the paper does not contain any experimental results. Authors in [30] study backward analysis using the local backward reachability algorithm.

Another approach to verification of safety properties works with invisible invariants [5, 20, 27, 28]. This incomplete approach is based on computing an inductive assertion. Our approach can be seen in some aspects as a restriction of this approach (inductive assertion is exactly determined by reachable $l + 1$-tuples). Contrary to the mentioned algorithms our algorithm always terminates.

Papers [8, 12] propose an algorithm based on generating abstract finite model of infinite users (an over-approximation of the model of the system with U^∞). Paper [12] moreover studies in which cases the algorithm returns a valid counterexample.

The technique presented in [6] takes each instance of a parametrised system as an expression of a process algebra and interprets this expression in modal mu-calculus, considering a process as a property transformer. The result is an infinite chain of mu-calculus formulae and technique solves the verification problem by finding the limit of a chain of formulae.

To sum it up, our verification algorithm is complete (contrary to [5, 8, 12, 27, 28, 31]), it computes a cutoff for l-symmetric RPs (not only checks an l-symmetric RP, contrary to [4, 6, 22, 23, 26, 30]), and the found cutoff is minimal (contrary to [21, 31]). Experiments demonstrate that the number of backward reachability iterations is typically very small (contrary to [4, 22, 23, 26, 30]) but steps of backward reachability in our algorithm are usually quite complex. As it was mentioned our verification algorithm computes the cutoff for l-symmetric RPs. Consequently our verification algorithm is important especially whenever one needs to find several types of possible errors in the system. The algorithm computes the reachable $l + 1$-tuples and after (or during) this computation it verifies which of the given properties are fulfilled.

As far as we know there is no other algorithm for verification of integrated sequences of RPs. Experiments show that the algorithm BOUND typically estimates the value of the bound correctly and thus is very efficient.

9 Conclusions

The paper studies systems composed of a control component and a dynamic number of user components (Control-User systems). Such type of systems is often of use in component-based systems e.g. when a central component provides services to unspecified number of clients. Safety properties are used to express that the system cannot enter an undesired configuration. The complexity of verification of reachability problems for Control-User systems stems from the fact that we want to guarantee the correctness for every possible number of users communicating with the control component. Though the problem of verification of symmetric safety properties on C-U system is decidable [21, 4], the state-space explosion is highly limiting factor for practical usage of verification techniques on those systems.

The paper presents two verification algorithms. The first algorithm solves the problem whether a given C-U model satisfies a given (finite) set of l-symmetric properties. The second algorithm is for computing the largest number of users which can be at the same time in a specific situation, state (so called *bound*).

The algorithms are evaluated on several C-U models of component-based systems (see also [2]). Characteristics of these models confirm practical usability of both algorithms as only instances with very low number of user components have to be explored during the algorithm.

An open question is whether similar approaches can be used to verify a wider class of reachability properties.

10 Acknowledgement

The authors thank Tomáš Poch for valuable comments. The authors would also like to acknowledge Ivana Vařeková and Jan Hutař for their contributions.

References

[1] http://fractal.objectweb.org/tutorial/index.html.

[2] http://anna.fi.muni.cz/coin/CUmodels/.

[3] A. Poetzsch-Heffter, J. Aldrich, M. Barnett, D. Giannakopoulou, G. T. Leavens, and N. Sharygina. Challenge Problem SAVCBS'07, May 2007. http://www.eecs.ucf.edu/ leavens/SAVCBS/2007/challenge.shtml.

[4] P. A. Abdulla, K. Cerans, B. Jonsson, and Y. Tsay. General decidability theorems for infinite-state systems. In *LICS'96*. IEEE Computer Society, 1996.

[5] T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. D. Zuck. Parameterized verification with automatically computed inductive assertions. In *CAV'01*, 2001.

[6] S. Basu and C. R. Ramakrishnan. Compositional analysis for verification of parameterized systems. *Theor. Comput. Sci.*, 354(2):211–229, 2006.

[7] L. Bulej, T. Bures, T. Coupaye, M. Decky, P. Jezek, P. Parizek, F. Plasil, T. Poch, N. Rivierre, O. Sery, and P. Tuma. *The Common Component Modeling Example: Comparing Software Component Models*, volume 5153, chapter CoCoME in Fractal. LNCS, 2008.

[8] M. Calder and A. Miller. An automatic abstraction technique for verifying featured, parameterised systems. *Theoretical Computer Science*, 2008. To appear.

[9] S. Chaki, E. M. Clarke, J. Ouaknine, N. Sharygina, and N. Sinha. State/event-based software model checking. In *IFM'04*, April 2004.

[10] E. M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1-2):77–104, 1996.

[11] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, USA, January 2000.

[12] E. M. Clarke, M. Talupur, and H. Veith. Proving ptolemy right: The environment abstraction principle for model checking concurrent systems. In *TACAS'08*, pages 33–47, 2008.

[13] L. E. Dickson. Finiteness of the odd prerfect and primitive abundant numbers with r distinct prime factors. *Amer. Journal Math.*, 35:413–422, 1913.

[14] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Property specification patterns for finite-state verification. In *FMSP*, pages 7–15. ACM Press, 1998.

[15] E. A. Emerson and V. Kahlon. Reducing model checking of the many to the few. In *CADE'00*, volume 1831 of *LNCS*, pages 236–254, 2000.

[16] E. A. Emerson and V. Kahlon. Exact and efficient verification of parameterized cache coherence protocols. In *CHARME'03*, LNCS, pages 247–262, 2003.

[17] E. A. Emerson and V. Kahlon. Model checking guarded protocols. In *LICS'03*, pages 361–370. IEEE Computer Society, 2003.

[18] E. A. Emerson and V. Kahlon. Rapid parameterized model checking of snoopy cache coherence protocols. In *TACAS'03*, LNCS, pages 144–159, 2003.

[19] E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In *POPL'95*, pages 85–94, New York, NY, USA, 1995. ACM.

[20] P. Fontaine and E. P. Gribomont. Decidability of invariant validation for paramaterized systems. In *TACAS'03*, LNCS, pages 97–112, 2003.

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

[22] B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In *TACAS'00*, volume 1785 of *LNCS*, pages 220–234, 2000.

[23] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *Theor. Comput. Sci.*, 256:93–112, 2001.

[24] J. Kofroň. *Behavior Protocols Extensions*. PhD thesis, Charles University in Prague, 2007.

[25] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, September 1989.

[26] M. Maidl. A unifying model checking approach for safety properties of parameterized systems. In *CAV'01*, volume 2102, pages 311–323, 2001.

[27] A. Pnueli, S. Ruah, and L. D. Zuck. Automatic deductive verification with invisible invariants. In *TACAS'01*, volume 2031, pages 82–97, 2001.

[28] A. Pnueli and L. D. Zuck. Model-checking and abstraction to the aid of parameterized systems. In *VMCAI'03*, page 4, 2003.

[29] Component reliability extensions for fractal component model. http://kraken.cs.cas.cz/ft/public/public_index.phtml.

[30] T. Rybina and A. Voronkov. Using canonical representations of solutions to speed up infinite-state model checking. In *CAV '02*. LNCS, 2002.

[31] P. Vařeková, P. Moravec, I. Černá, and B. Zimmerova. Effective-Verification of Systems with a Dynamic-Number of Components. In *SAVCBS'07*, pages 3–13. ACM Press, 2007.

[32] P. Vařeková and B. Zimmerova. Solution of challenge problem. In *SAVCBS'07*. ACM Press, 2007.

[33] B. Zimmerova and P. Vařeková. Reflecting creation and destruction of instances in CBSs modelling and verification. In *MEMICS'07*, Znojmo, Czech Republic, 2007.

[34] B. Zimmerova, P. Vařeková, N. Beneš, I. Černá, L. Brim, and J. Sochor. *The Common Component Modeling Example: Comparing Software Component Models*, volume 5153, chapter Component-Interaction Automata Approach (CoIn), pages 146–176. LNCS, 2008.

A Appendix

A.1 Proof of Lemma 4.1

Lemma 1. *Let* C, U *be a C-U model,* $A \subseteq Q_C$*. Then backward reachability in* $C\|U^\infty$ *starting with* $Q = \bigcup_{i\in\mathbb{N}} A \times \underbrace{Q_U \times \cdots \times Q_U}_{i}$ *terminates.*

Proof: To prove the lemma we first define an auxiliary object: C-U protocol and its states. Then we transform the problem of backward reachability for Q in the LKS $C\|U^\infty$ to backward reachability for the set of states $A \times \mathbb{N}_0^{|Q_U|}$ in the C-U protocol $\mathcal{C}\|\mathcal{U}^\infty$ and we prove that backward reachability for $A \times \mathbb{N}_0^{|Q_U|}$ in the C-U protocol $\mathcal{C}\|\mathcal{U}^\infty$ terminates. As a consequence we demonstrate that the original backward reachability terminates as well.

Definition A.1 (C-U protocol)**.** *A* Control-User protocol *(C-U protocol for short) is a 5-tuple* $(Q, S, I, \delta, \Sigma)$*, where* Q *is a finite set of* control states, S *is a finite ordered set of* user states, $I \subseteq (Q \times \mathbb{N}_0^{|S|})$ *is a nonempty set of* initial states, Σ *is a finite set of* actions, *and* $\delta \subseteq (Q \times \mathbb{N}_0^{|S|}) \times \Sigma \times (Q \times \mathbb{N}_0^{|S|})$ *is a* transition relation. *Elements of* $(Q\times\mathbb{N}_0^{|S|})$ *are called* states *of the C-U protocol. The reachability relation is defined in a similar way as for C-U models.*

W.l.o.g. we can assume that the user states of a C-U protocol are named $\{1, \dots, |Q_U|\}$ and $in_U = 1$. For a global state $q \in Q_{C\|U^x}$, $x \in \mathbb{N} \cup \{\infty\}$, and for each $j \in Q_U$ the symbol $\#j(q)$ denotes the number of users which are within the global state q in the local state j and $\#0(q)$ denotes the state of the control component in the global state q.

Definition A.2 (C-U protocols $\mathcal{C}\|\mathcal{U}^n$, $\mathcal{C}\|\mathcal{U}^\infty$)**.** *Let* C, U *be a C-U model and* $n \in \mathbb{N}$. *We define the C-U protocol* $\mathcal{C}\|\mathcal{U}^n$ *as a tuple* $(Q_C, Q_U, (in_C, n, 0, \dots, 0), \delta, \Sigma_{C\|U^n})$. *Here* $(q^1, a, q^2) \in \delta$ *iff there is* $(r^1, a, r^2) \in \delta_{C\|U^n}$*, such that for each* $i \in \{1, 2\}$ *it holds* $q^i = (\#0(r^i), (\#1(r^i), \dots, \#|Q_U|(r^i)))$.

C-U protocol assigned to the C-U model is $\mathcal{C}\|\mathcal{U}^\infty = (Q_C, Q_U, \bigcup_{n\in\mathbb{N}} I_{\mathcal{C}\|\mathcal{U}^n}, \bigcup_{n\in\mathbb{N}} \delta_{\mathcal{C}\|\mathcal{U}^n}, \bigcup_{n\in\mathbb{N}} \Sigma_{\mathcal{C}\|\mathcal{U}^n})$.

The C-U protocol $\mathcal{C}\|\mathcal{U}^n$ is an abstraction of the LKS $C\|U^n$. The abstraction of the state set $Q_{C\|U^x}$ is the state set $Q_C \times \mathbb{N}^{|Q_U|}$. A state $q \in Q_{C\|U^n}$ is in the set $Q_C \times \mathbb{N}_0^{|Q_U|}$ represented by the tuple $(\#0(q), (\#1(q), \dots, \#|Q_U|(q)))$.

Let $q_1, q_2 \in Q_{C\|U^\infty}$ satisfy $(\#0(q_1), (\#1(q_1), \dots, \#|Q_U|(q_1))) = (\#0(q_2),(\#1(q_2), \dots, \#|Q_U|(q_2)))$. Then if a state q_1' is a successor of q_1 in $C\|U^\infty$ then a global state q_2' with the same state of the control component as in q_1' and with the same numbers of users in local states as in q_1' is a successor of q_2 in $C\|U^\infty$.

Observation 1. *Let* C, U *be a C-U model and let* $A \subseteq Q_C$. *Then backward reachability in* $C\|U^\infty$ *starting with the set* $\{q \mid \#0(q) \in A\}$ *of global states modelling the control component in a state from* A *terminates in the* i*-th step iff backward reachability in* $\mathcal{C}\|\mathcal{U}^\infty$ *starting with* $A \times \mathbb{N}_0^{|Q_u|}$ *terminates in the* i*-th step.*

Definition A.3. *A subset* X *of the state set of a C-U protocol is* upward closed *iff for every state* $(q_c, i_1, \ldots, i_{|Q_u|}) \in X$ *and every tuple* $(j_1, \ldots, j_{|Q_u|})$ *such that* $i_1 \leq j_1, \ldots, i_{|Q_u|} \leq j_{|Q_u|}$, *the state* $(q_c, j_1, \ldots, j_{|Q_u|})$ *belongs to* X *as well.*

For any $A \subseteq Q_C$ the set $A \times \mathbb{N}_0^{|Q_U|}$ is upward closed. As the set of vectors of k natural numbers with the component-wise ordering is partially ordered (Dickson's lemma [13]) we can use results from [4] and conclude that for every upward closed set backward reachability in a C-U protocol terminates. As a consequence of Observation 1 we have that the backward reachability in $C\|U^\infty$ starting with $X = \bigcup_{i\in\mathbb{N}} A \times \underbrace{Q_U \times \cdots \times Q_U}_{i}$ terminates as well.

A.2 Proof of Lemma 5.2

Let us first study an alternative procedure BACK_REACH_ALTERNATIVE(T), where T is a set of $l+1$-tuples. The procedure works with a modified C-U model consisting of the original user model and a new control component model C_{new}. C_{new} models the composition of the original control component and l user components. Let Q' be the set of all states of $C_{new}\|U^\infty$ in which C_{new} is in the local state equal to a tuple in T ($Q' = \bigcup_{i\in\mathbb{N}} T \times \underbrace{Q_U \times \cdots \times Q_U}_{i}$). The procedure BACK_REACH_ALTERNATIVE(T) computes using backward reachability whether a state from Q' is reachable in $C_{new}\|U^\infty$. If it is so, then T contains an $l{+}1$-tuple reachable in $C\|U^\infty$, otherwise T does not contain such an $l{+}1$-tuple. Lemma 4.1 guarantees that BACK_REACH_ALTERNATIVE(T) always terminates.

Lemma 2. *The procedure* BACKWARD REACHABILITY *always terminates.*

Proof: At first we prove that BACKWARD REACHABILITY(T) always terminates after less or equal number of transitions than the BACK_REACH_ALTERNATIVE(T).

Let $Q'_0 = Q'$, where Q' is defined in the previous paragraph, and Q'_j is the set of states from which Q'_0 is reachable in j or less transitions in $C_{new}\|U^\infty$.

Let Q_0 be the set of all the states of $C\|U^\infty$ to which is assigned a tuple from T, and Q_j is the set of states from which Q_0 is reachable in j or less transitions in $C\|U^\infty$.

Let us assume that the statement does not hold. Let $i \in \mathbb{N}$ be such that $Q'_i = Q'_{i+1}$ and $Q_i \subset Q_{i+1}$. If $q \in Q_{i+1} \setminus Q_i$ then there is a state $q' \in Q_0$ that is reachable from q in $C\|U^\infty$ by a sequence of transitions of length $i+1$. As $q' \in Q_0$ it to q' is assigned an $l+1$-tuple from T and thus there exists a state $\overline{q}' \in Q'_0$ such that the number of users (including the users composed in the new control component) in any local state is the same as in the state q', and the state of the control component is the same too. The definition of transition relations implies that there is a state $\overline{q} \in Q'_{i+1} = Q'_i$ such that the number of users in any state is the same as in the state q and the state of the control component is the same too. Since $\overline{q} \in Q'_i$ a state $\overline{q}'' \in Q'_0$ is reachable from $\overline{q}$ by a sequence of transitions of length less than $i+1$. Because $\overline{q}'' = (q_{C_{new}}, q_0, \dots) \in Q'_0$ the local state $q_{C_{new}} \in T$. Thus there is a state $q'' \in Q_0$ such that the number of users in any local state is the same as in the state q'' and the state of the control component is the same too. Hence from the state q, the state q'' must be reachable by a sequence of transitions of the length less than $i+1$ and consequently q is contained in the set Q_i.

BACKWARD REACHABILITY(T) always terminates after at most the number of steps than the BACK_REACH_ALTERNATIVE(T). Moreover we know that the procedure BACK_REACH_ALTERNATIVE always terminates. Thus Lemma 5.2 holds.

A.3 Proof of Lemma 5.3

Lemma 3. *Let for a C-U model* C, U *and* $l \in \mathbb{N}_0$ *the following condition is true:*

> *An unreachable* $l+1$*-tuple of* $C\|U^\infty$ *is assigned to every unreachable* $(l+1)+1$*-tuple of* $C\|U^\infty$. *(*)*

Then BACKWARD REACHABILITY(T)*, where* T *is the set of all unreachable* $l+1$*-tuples of* $C\|U^\infty$*, terminates after the first iteration.*

Proof: The set Q from which backward reachability starts is in this case the set containing all states of $C\|U^\infty$ to which an $l+1$-tuple from T is assigned. The set of states generated in the first iteration of backward reachability is the set Q together with the set of predecessors of the states in Q. Consequently it is necessary to prove that in this case any predecessor of a state in Q is in the set Q. Let $q \in Q$ and $(q' = (q'_c, q'_1, \ldots, q'_n), l, q = (q_c, q_1, \ldots, q_n)) \in \delta_{C\|U^\infty}$. Because $q \in Q$, an unreachable $l+1$-tuple must be assigned to q, denote such a tuple $t = (q_c, q_{i_1}, \ldots, q_{i_l})$. There are four possibilities:

- (q', l, q) models a communication of the control component. The $l+1$-tuple $t' = (q'_c, q_{i_1}, \ldots, q_{i_l})$ is assigned to q' and because $(t', l, t) \in \delta_{C\|U^\infty}$ the tuple t' is unreachable in $C\|U^\infty$. Consequently $q' \in Q$.
- (q', l, q) models a communication of a user j. If $j \notin \{i_1, \ldots, i_l\}$ then t is assigned to q' and thus $q' \in Q$. Else $j \in \{i_1, \ldots, i_l\}$ and $j = i_x$ for some $x \in \{1, \ldots, l\}$. Then the tuple $t' = (q_c, q_{i_1}, \ldots, q_{i_x - 1}, q'_{i_x}, q_{i_x+1}, \ldots, q_{i_l})$ is assigned to q' and because $(t', l, t) \in \delta_{C\|U^\infty}$ the tuple t' is unreachable in $C\|U^\infty$. Consequently $q' \in Q$.
- (q', l, q) models a communication of the control component and user j.
  - Let $j \notin \{i_1, \ldots, i_l\}$ then the $(l+1)+1$-tuple $u' = (q'_c, q_{i_1}, \ldots, q_{i_l}, q'_j)$ is assigned to q'. Moreover it holds
  $$(u', l, u = (q_c, q_{i_1}, \ldots, q_{i_l}, q_j)) \in \delta_{C\|U^\infty}.$$
  Because t is an unreachable $l+1$-tuple, u must be an unreachable $(l+1)+1$-tuple, thus u' must be an unreachable $(l+1)+1$-tuple too. Hence and from the assumption of the lemma we obtain that to the state q' is assigned an unreachable $l+1$-tuple and consequently $q' \in Q$.
  - In the other case $j \in \{i_1, \ldots, i_l\}$ and $i_x = j$ for some $x \in \{1, \ldots, l\}$. Then the tuple $t' = (q'_c, q_{i_1}, \ldots, q_{i_x-1}, q'_{i_x}, q_{i_x+1}, \ldots, q_{i_l})$ is assigned to the state q' and because $(t', l, t) \in \delta_{C\|U^\infty}$ the tuple t' is unreachable in $C\|U^\infty$. Thus $q' \in Q$.
- (q', l, q) models a communication of two users j_1, j_2.
  - Let $j_1, j_2 \notin \{i_1, \ldots, i_l\}$ then t is assigned to q' and thus $q' \in Q$.
  - Let $j_1 \in \{i_1, \ldots, i_l\}$ and $i_x = j_1$ for $x \in \{1, \ldots, l\}$, and $j_2 \notin \{i_1, \ldots, i_l\}$. In such a case the $(l+1)+1$-tuple
  $$u' = (q_c, q_{i_1}, \ldots, q_{i_x-1}, q'_{i_x}, q_{i_x+1}, \ldots, q_{i_l}, q'_{j_2})$$
  is assigned to q'. Moreover it holds
  $$(u', l, u = (q_c, q_{i_1}, \ldots, q_{i_l}, q_{j_2})) \in \delta_{C\|U^\infty}.$$

Because t is an unreachable $l+1$-tuple, u must be an unreachable $(l+1)+1$-tuple, thus u' must be an unreachable $(l+1)+1$-tuple too. Hence and from the assumption of the lemma we obtain that to the state q' is assigned an unreachable $l+1$-tuple and consequently $q' \in Q$.

- Let $j_1, j_2 \in \{i_1, \ldots, i_l\}$ and $i_x = j_1$, $i_y = j_2$ for some $x, y \in \{1, \ldots, l\}$. Without lost of generality we assume that $x < y$. Then the tuple $t' = (q'_c, q_{i_1}, \ldots, q_{i_x-1}, q'_{i_x}, q_{i_x+1}, \ldots, q_{i_y-1}, q'_{i_y}, q_{i_y+1}, \ldots, q_{i_l})$ is assigned to the state q' and because $(t', l, t) \in \delta_{C\|U^\infty}$ the tuple t' is unreachable in $C\|U^\infty$. Thus $q' \in Q$.