

Automated Computing of the Maximal Number of Handled Clients for Client-Server Systems

P. Vařeková and I. Vařeková and I. Černá^{1,2}

*Faculty of Informatics, Masaryk University
Brno, Czech Republic*

Abstract

In many real software systems like Client-Server systems, one can identify a stable part (server, instance handler, control component) and a number of uniform components of the same type (clients, instances, users). When analysing performance and correctness of these systems we need to answer questions like "What is the maximal possible number of clients which can be handled simultaneously?" or more generally "What is the maximal possible number of clients which are in the some special situation when the control component is in a particular state?". In the paper we propose an automated technique solving such questions. For Client-Server systems we reduce the problem of finding the upper bound on the number of handled clients to the formal verification of reachability properties in infinite state transition systems. For the verification task we propose an efficient and fully automated algorithm which combines several techniques proposed in existing literature. Applying the algorithm we verify models of several previously published systems.

Keywords: Client-Server systems, formal verification, infinite state systems, component-based system analysis.

1 Introduction

The Client-Server computing model is a popular concept and many Internet and database applications are based on this model. Although this concept can be applied to many different kinds of applications, the architecture remains fundamentally the same. It distinguishes client applications from server applications where a client can communicate only with the server. In Client-Server systems the number of clients changes in time but the number of servers is fixed. This fact causes that there can be large differences between the behaviour of the system with different number of clients. Therefore it is important to understand how the behaviour of the system depends on the number of deployed clients. One of the many important questions is *What is the maximal possible number of clients which are in the same*

¹ Email: {xvareko1,xvareko2,cerna}@fi.muni.cz

² The authors have been supported by grant No. 1ET400300504 and No. 1ET408050503.

special situation when the control component is in a particular state? From this information we can deduce which features of the system will be affected when the number of deployed clients increases or how many servers are necessary in a system if the expected number of clients is known.

In the paper we propose an algorithm which answers the type of questions which is mentioned above. The algorithm comes out from formal verification. The Client-Server system under the study is described using a formal language. We model separately the client part and the server part of the system. The model of the server can be composed from any fixed number of subparts modelling individual servers in the original system. The algorithm works with the model composed of an arbitrary number of client models and one model of the server. For this model, in the literature called *parameterised system model*, the problem of finding the maximal number of clients can be reduced to the problem: *Compute the maximal k such that a state in which k clients and the control component are in a specific states simultaneously is reachable in the model*. Note, that our algorithm can be used for any system consisting of two types of components – a stable component and an unknown number of uniform components of the same type.

The proposed algorithm is based on techniques from the parametrised systems verification, consequently we use the terminology from this field [10,8]. Therefore the systems which we study in the paper are denoted as *control-user systems*. The part of the system which is unique (server in Client-Server systems) we call *control component* and the parts of the system with an identical model (clients in Client-Server systems) we denote *user components* or users. The algorithm is based on an over-approximation of the set of reachable states in a similar way as the algorithm presented in [19]. The main improvement comparing to [19] is a reduction of the computational time and space required by the algorithm. In the paper we demonstrate its efficiency on a case study. Section 2 introduces the Control-User model. Section 3 presents and formalises the problem which we solve. Section 4 describes the algorithm while the following section its evaluation. Section 7 summarises the results and outlines the aims for future work.

2 The Control-User System Model

In the paper we consider systems which consist of a unique control component and an arbitrary number of user components with an identical model. An example of such a system with n users (Tokens) is in Figure 1. Components in the systems 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 a formal model of a Control-User system we use a labelled Kripke structure. A labelled Kripke structure is a model underlying many formalisms capturing interactions between parts of the system like I/O automata [14], Component-Interaction Automata [20,5], or Extended Behavior Protocols [13].

Definition 2.1 A *labelled Kripke structure (LKS)* is a 6-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*, $L : Q \rightarrow 2^{Ap}$ 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 $q \rightarrow q'$ if there is a label $l \in \Sigma$ such that $(q, l, q') \in \delta$, \rightarrow^* is the transitive and reflexive closure of \rightarrow . A state q is reachable iff $in \rightarrow^* q$ for some $in \in I$. Let $q = (q_0, \dots, q_n)$ be an $n + 1$ -tuple. Then $pr_i(q)$, $i = 0, \dots, n$, 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) if and only if Q_C , Q_U , Σ_C , and Σ_U are finite.

Definition 2.2 Let $n \in \mathbb{N}_0$ and for each $i = 0, \dots, n$ let $K_i = (Q_i, I_i, Ap_i, L_i, \Sigma_i, \delta_i)$ be an LKS. Then $K_0 \parallel \dots \parallel K_n$ denotes the asynchronous composition of K_0, \dots, K_n which is modelled by the LKS

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

where the state-labelling function L assigns to each state (q_0, \dots, q_n) the set $\bigcup_{i=0}^n L_i(q_i) \times \{i\}$. The alphabet $\Sigma = \bigcup_{i=0}^n \Sigma_{i,int} \cup \{l \mid l \in \bigcup_{i=0}^n \Sigma'_{i,inp} \wedge l \in \bigcup_{i=0}^n \Sigma'_{i,out}\}$. For every $q, q' \in (Q_1 \times \dots \times Q_n)$ and $l \in \Sigma$ the transition $(q, l, q') \in \delta$ iff it holds

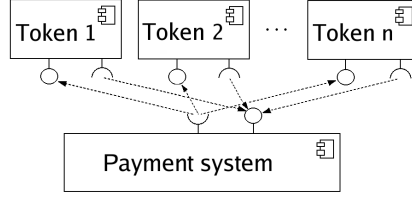
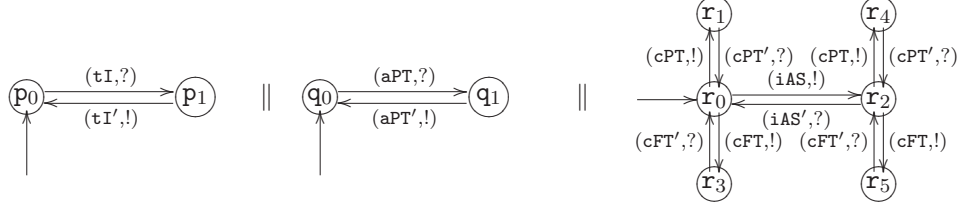
- $\exists i \in \{0, \dots, n\}: \forall j \in \{0, \dots, n\}, j \neq i :$
 $pr_j(q) = pr_j(q')$, and $(pr_i(q), l, pr_i(q')) \in \delta_i$, or
- $\exists i, i' \in \{0, \dots, n\}, i \neq i' : \forall j \in \{0, \dots, n\}, i \neq j \neq i' :$
 $pr_j(q) = pr_j(q')$, $(pr_i(q), (l, ?), pr_i(q')) \in \delta_i$, 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 Let C, U be a C-U model, $n \in \mathbb{N}$. Then $C \parallel U^n$ denotes the composition $C \parallel U \parallel \dots \parallel U$ of C and n copies of U . $C \parallel U^\infty$ is an infinite state LKS defined $C \parallel U^\infty = (\bigcup_{n \in \mathbb{N}} Q_{C \parallel U^n}, \bigcup_{n \in \mathbb{N}} \{in_{C \parallel U^n}\}, \bigcup_{n \in \mathbb{N}} Ap_{C \parallel U^n}, \bigcup_{n \in \mathbb{N}} L_{C \parallel U^n}, \bigcup_{n \in \mathbb{N}} \Sigma_{C \parallel U^n}, \bigcup_{n \in \mathbb{N}} \delta_{C \parallel U^n})$.

Note 1 For simplicity we allow that there can be a communication between clients. The proposed algorithm can be used for such systems, but we did not find any interesting example of a CU-system which uses this type of communication.

Example 2.4 As a running example we present a part of a model of the prototype implementation of a payment system for public Internet access on airports [1]. It models a system where clients (represented as instances of the component Token) of several air-carriers can have access to the Internet. Clients have to authenticate themselves or pay for the service before the system permits them to establish communication with the Internet. There are two types of sessions - the prepaid session and free session (for clients with a valid fly ticket for first class or business class).

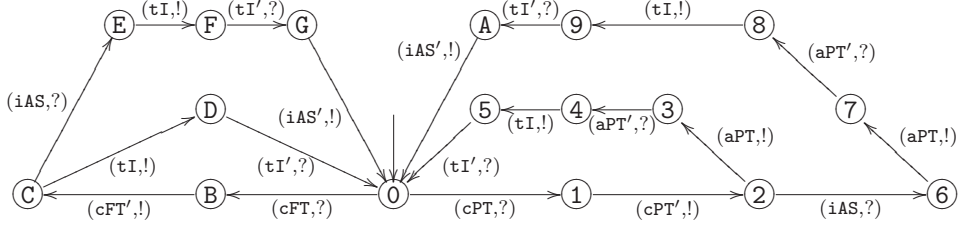

 Fig. 1. Internet payment system with n Tokens, modelled by $C_{ex} \parallel U_{ex}^n$.

 Fig. 2. LKS C_{ex} modelling *Internet payment system*.

In the model, to each method, e.g. `createPrepaidToken()`, is assigned a tuple of action names: `createPrepaidToken` denotes the call of the method, and `createPrepaidToken'` the return from the method. These two determine the beginning and the end of the method's execution. In the models we abbreviate the name of the method `tokenInvalidated()` to `tI`, `adjustPrepaidTime()` to `aPT`, `createPrepaidToken()` to `cPT`, `invalidateAndSave()` to `iAS`, and `createFreeToken()` to `cFT`.

The payment system (control component) models three independent methods and hence its model consists of three independent parallel parts. The first of them is able to receive an information that a Token is invalid (the terminating phase of an Internet session is successfully finished). The second is for changing the prepaid time during the invalidation of the Token. The last is for creating a Token (it represents the start of an Internet session) and for starting of invalidation of the Token (it represents termination of the session). An LKS model of the control component C_{ex} is depicted in Figure 2. Its set of atomic propositions $Ap_{C_{ex}}$ is $\{\text{active}\}$. The only state in which the atomic proposition does not hold is (p_0, q_0, r_0) .

The Token (user component) models a client of the system. At first a client must be initialised and then it models an initialised Internet session. There are two possibilities how to terminate an existing session - a client (Token) terminates the session or the system terminates the session (because the prepaid time is used up or fly ticket becomes invalidated). If the client terminates the session, then after some time the control component detects it and sends to the Token the action `InvalidateAndSave`. After that, if the session is prepaid, the Token changes the prepaid time (actions `adjustPrepaidTime`, `adjustPrepaidTime'`). Subsequently the Token announces that it is invalidated (actions `tokenInvalidated`, `tokenInvalidated'`) and returns `InvalidateAndSave'`. If the system terminates the session, if the session is prepaid, the system changes the prepaid time (actions `adjustPrepaidTime`, `adjustPrepaidTime'`). Subsequently the Token announces that it is invalidated (actions `tokenInvalidated`, `tokenInvalidated'`).

The user component U_{ex} is depicted in Figure 3. Its atomic propositions are $Ap_{U_{ex}} = \{\text{finishing_Session}, \text{free_Session}, \text{prepaid_Session}, \text{served}\}$. For each $q \in \{\mathbf{C} - \mathbf{G}\}$ the set $L_{U_{ex}}(q)$ contains `free_Session`, for $q \in \{\mathbf{2} - \mathbf{A}\}$ it moreover contains


 Fig. 3. LKS U_{ex} modelling *Token*.

prepaid_Session, for $q \in \{3 - 5, 7 - A, D - G\}$ it moreover contains finishing_Session, for $q \in \{0, 2, 4, C\}$ it moreover contains unserved (the method A was called, but the complement A' has not been received yet). For example $L_{U_{ex}}(C) = \{\text{free_Session}, \text{unserved}\}$ and $L_{U_{ex}}(7) = \{\text{prepaid_Session}, \text{finishing_Session}\}$.

3 The bounding problem

In this section we formally describe the problem which we solve in the paper and we show its motivation on the model of a payment system which is introduced in the previous section.

3.1 Motivation

During analysis of C-U systems we are often interested in the *maximal possible number of users* which can be simultaneously in the same state when the control component is in a particular situation.

Example 3.1 For the model of payment system from Example 2.4 we show several interesting properties of the described form.

1) It is clear that this C-U system should satisfy that the number of clients which can be connected to the Internet simultaneously is not bounded, or it is bounded by an integer greater than a number of possible clients (e.g. 10^{10}).

2) If the number of Tokens which use an initialised session increases, than obviously the amount of time that is necessary for finishing the session (from starting to accomplishing the finishing phase) may increase too. It is because the number of initialised Tokens which want to finish their session at the same time may increase and thus the time necessary for starting the finishing phase may increase too.

The interesting question is: If a client successfully starts the finishing phase, then the time necessary to accomplishing the finishing phase may increase too? This time may increase iff the number of clients which can be in the middle of a session is not bounded. It can happen that the other clients in the finishing phase can be served before the selected client and thus the selected client must wait.

3) Because there is always a possibility that for some unpredictable reason the payment system shuts down without any log message, it is important to know how many Tokens can be in the middle of a communication with the payment system simultaneously. A Token is in the middle of a communication if it has sent a request and is waiting for a respond or if the system sent a request to the Token and it is waiting for a response.

Consequently for the model of the payment system it is interesting to know the maximal number of Tokens which are simultaneously in the following situations:

- 1) are using (arbitrary/prepaid/free) session?
- 2) started finishing of their (arbitrary/prepaid/free) session, but the finishing phase is not completed?
- 3) are in the middle of a communication with the payment system?

3.2 l -symmetric reachability properties

In this part we define auxiliary terms which are used in the formal definition of the problem solved in the paper - reachability properties of C-U model and l -symmetric reachability properties.

Formulae of the propositional logic 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*. A reachability property (or RP for short) is a property capturing that a state satisfying a given propositional formula is reachable in the system. The *general reachability problem* for C-U models can be formulated as:

Instance (reachability):

- a C-U model C, U
- $\{\varphi_n\}_{n \in \mathbb{N}}$, where φ_n is a formula of the propositional logic over $Ap_{C \parallel U^n}$.

Problem:

Is there $n \in \mathbb{N}$ such that a state satisfying φ_n is reachable in $C \parallel U^n$?

There are special sets of reachability properties which make no distinction among users – so called *0-symmetric RP*, *1-symmetric RP*, etc. For fixed $l \in \mathbb{N}_0$ and any $n \in \mathbb{N}$, an l -symmetric RP (or l -SRP) guarantees that if a state $q \in Q_{C \parallel U^n}$ 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):

- a C-U model C, U , a number $l \in \mathbb{N}_0$
- a sequence of l -symmetric formulae $\{\varphi_n\}_{n \in \mathbb{N}}$, where for each $n \in \mathbb{N}$:

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

Here ψ is a formula of the propositional logic over atomic propositions $L_C \times \{0\} \cup L_U \times \{1\} \cup \dots \cup L_U \times \{l\}$, f is an injective function, and $\psi_{(1, f(1)), \dots, (l, f(l))}$ 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 3.2 l -symmetric RPs of the C-U model from Example 2.4 (inspired by the properties from Example 3.1) are e.g. properties describing reachability of a

state satisfying:

- (i) m (or more) Tokens are connected to the payment system simultaneously. It is an m -SRP with the underlying formula $\psi = (\text{prepaid_Session}, 1) \vee (\text{free_Session}, 1) \wedge \dots \wedge (\text{prepaid_Session}, m) \vee (\text{free_Session}, m)$.
- (ii) m Tokens are finishing their free session simultaneously and the payment system is active. It is an m -SRP and the underlying formula $\psi = (\text{active}, 0) \wedge (\text{finishing_Session}, 1) \wedge (\text{free_Session}, 1) \wedge \dots \wedge (\text{finishing_Session}, m) \wedge (\text{free_Session}, m)$.
- (iii) m Tokens can be finishing their prepaid session simultaneously and the payment system is active. It is an m -SRP and the underlying formula $\psi = (\text{active}, 0) \wedge (\text{finishing_Session}, 1) \wedge \dots \wedge (\text{finishing_Session}, m)$.
- (iv) m Tokens are serviced by the payment system simultaneously. It is an m -SRP and the underlying formula $\psi = \neg(\text{unserved}, 1) \wedge \dots \wedge \neg(\text{unserved}, m)$.
- (v) m prepaid Tokens are serviced by the payment system simultaneously. It is an m -SRP and the underlying formula $\psi = \neg(\text{unserved}, 1) \wedge (\text{prepaid_Session}, 1) \wedge \dots \wedge \neg(\text{unserved}, m) \wedge (\text{prepaid_Session}, m)$.

3.3 Instance of the problem

The question "what is the maximal possible number of users which can be simultaneously in the same state when the control component is in a particular situation" can be described using a sequence of l -symmetric 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 simultaneously in the same state when the control component is in a particular situation.*

If for some $j > i$ it holds that j users together with the control component are in the specified setting, it usually means that i of the j users are together with the control component in the particular setting to. Thus the sequence of underlying formulae ψ_1, ψ_2, \dots of P_1, P_2, \dots often satisfies that $\psi_j \Rightarrow \psi_i$ is true for every $j > i$. The next lemma clearly follows from the definitions:

Lemma 3.3 *Let $\{P_m\}_{m \in \mathbb{N}}$ be a sequence where every P_m is an m -symmetric RP with an underlying formula ψ_m . Let the implication $\psi_j \Rightarrow \psi_i$ be true for every $j > i$. Then if $C \parallel U^\infty$ satisfies P_j then it also satisfies P_i for each $j > i$. (*)*

Note that an equivalent to the condition (*) is the requirement: if $C \parallel U^\infty$ does not satisfy P_i then it does not satisfy P_j for any $j > i$. Consequently if a sequence of RP $\{P_m\}_{m \in \mathbb{N}}$ satisfies the assumption of Lemma 3.3 then maximally one number b satisfies that P_b holds and P_{b+1} does not hold. Thus in such cases one of the following conditions is fulfilled:

- a) none of properties $\{P_m\}_{m \in \mathbb{N}}$ holds,
- b) there exists $b \in \mathbb{N}$ such that P_m for $m \leq b$ holds and P_m for $m > b$ does not hold
- c) each of properties $\{P_m\}_{m \in \mathbb{N}}$ holds,

Thus the problem which we solve in the paper can be transformed to the task to find the number b such that P_b is satisfied and P_{b+1} is not satisfied (if it exists) for a given sequence of RPs. We call this value *bound*. In practise we often have

a number Max and the question is whether $bound$ is at most Max . Only if this is true we want to know the exact value of $bound$. The task which we study can be described as

Instance:

- a C-U model C, U , number $Max \in \mathbb{N}$
- a sequence $\{P_m\}_{m \in \{1, \dots, Max\}}$, where P_m is an m -symmetric RP satisfying (*)

Problem: Compute

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

4 Verification algorithm

Verification of k -symmetric properties is decidable [10,3] and there are several algorithms for verification of k -symmetric properties. Thus it is possible to solve the bounding problem using some of the previously published verification algorithms and iteratively verify properties $P_1, P_2, \dots, P_{\min(Max, bound+1)}$. If the algorithm is based on backward reachability [3,11,12,15,18] the number of steps used for verification of an k -symmetric RP is linear in k and the number of conditions needed for describing the set of backward reachable states is exponential in k . In case of algorithms based on cutoffs [10,19], invisible invariants [4,9,16,17], or abstractions [6,7] the space complexity of verification of a satisfied k -symmetric RP is exponential in k . Thus the proposed solution based on iterative verification of properties is acceptable only when $bound$ or Max is small. In other cases the approach is not efficient.

The main purpose of this paper is to propose an algorithm which efficiently finds $bound$ for C-U models of real Client-Server systems. The algorithm which we describe is an improved version of the algorithm from [19]. For a comparison of the two algorithms see Related work.

To describe the algorithm we first define so called $k+1$ -tuples. $k+1$ -tuples serve as an abstraction of a C-U model state where the local states of exactly k chosen users in an arbitrary order are maintained.

Definition 4.1 Let C, U be a C-U model, and $k \in \mathbb{N}_0$. Then a $k+1$ -tuple $(q_C, q_1, \dots, q_k) \in Q_C \parallel U^k$ is assigned to the state $q \in Q_C \parallel U^\infty$ iff the local state of C in q is q_C and there are k different users in q with local states q_1, \dots, q_k .

In a similar way we can assign a $k+1$ -tuple to a $(k+i)+1$ -tuple t .

Example 4.2 For example to the tuple $((p_1, q_0, r_3), 0, 2, D, G, 1, A, B)$ where C_{ex}, U_{ex} is described in Example 2.4 are assigned $3+1$ -tuples $((p_1, q_0, r_3), 0, 2, D), ((p_1, q_0, r_3), D, 0, G), ((p_1, q_0, r_3), B, A, D)$, etc.

The paper [19] proposes the procedure REACHABLE $l+1$ -TUPLES which for input parameters C, U, k computes all $k+1$ -tuples reachable in $C \parallel U^\infty$. For the space reasons we do not describe the procedure here.

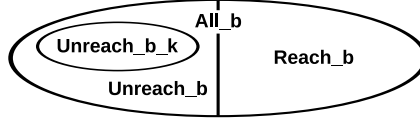


Fig. 4. The sets of $b+1$ -tuples All_b , $Unreach_b$, $Reach_b$, and $Unreach_b.k$.

4.1 Verification using $k+1$ -tuples

Let $P_k = \{\varphi_n\}_{n \in \mathbb{N}}$ be a fixed sequence describing a k -symmetric RP, with the underlying formula ψ . For an arbitrary $i \in \mathbb{N}_0$ a state $q \in Q_{C \parallel U^i}$ satisfies φ_i iff there is a $k+1$ -tuple t assigned to q such that t (which is a state of $C \parallel U^k$) satisfies $\varphi_k = \psi$. Hence if we find all $k+1$ -tuples assigned to the reachable states of $C \parallel U^\infty$ (so called *reachable $k+1$ -tuples* of $C \parallel U^\infty$ denoted $Reach_k$) we can easily verify the validity of a given k -symmetric RP. Consequently we can easily make the procedure $IS.k - 1_BOUND$ which from the sets of all reachable states $Reach_k$, $Reach_{k+1}$ computes whether k is *bound*.

Procedure 1 $IS.k - 1_BOUND(Reach_k, Reach_{k+1}, P_k, P_{k+1}, k)$

$Reach_k = REACHABLE\ l + 1\text{-TUPLES}(k)$
if not exists $t \in Reach_k$ satisfying P_k **then return false**
 $Reach_k = REACHABLE\ l + 1\text{-TUPLES}(k+1)$
if exists $t \in Reach_{k+1}$ satisfying P_{k+1} **then return false**
return true

4.2 Over-approximation of reachable $k+1$ -tuples

The computational time and space used in $REACHABLE\ l + 1\text{-TUPLES}$ for computation of the set $Reach_b$ is exponential in b . Thus it is inefficient for a high integer b . On the other hand as it is written in [19] the set can be efficiently over-approximated. The main idea is that if we have a high integer b , we can choose a small integer k and under-approximate the set $Unreach_b$ of all unreachable $b+1$ -tuples of $C \parallel U^\infty$ by the set $Unreach_b.k$ of all $b+1$ -tuples to which is assigned an unreachable $k+1$ -tuple of $C \parallel U^\infty$ (see Figure 4). The profit is that to compute $Unreach_b.k$ instead of $Unreach_b$ it is necessary to find $Reach_k$ instead of $Reach_b$. The set $Unreach_b.k$ can also be used for computing an over-approximation of all reachable $b+1$ -tuples of $C \parallel U^\infty$. Let us denote All_b the set of all possible $b+1$ -tuples of $C \parallel U^\infty$. Then the set of all reachable $b+1$ -tuples of $C \parallel U^\infty$ can be over-approximated using the set $Reach_b.k = All_b \setminus Unreach_b.k$ of all $b+1$ -tuples to which is not assigned an unreachable $k+1$ -tuple of $C \parallel U^\infty$ (see Figure 4).

This over-approximation of $Reach_b$ can be used to over-approximate *bound* and to compute a candidate for the *bound* - the minimal number x such that that $Reach_x.k$ contains an $x+1$ -tuple satisfying P_x and $Reach_{x+1}.k$ does not contain an $(x+1) + 1$ -tuple satisfying P_{x+1} . From the fact that $Reach_{x+1} \subseteq Reach_{x+1}.k$ it follows that *bound* must be smaller or equal to x . This idea is used in the procedure $LAST_SATISFIED$. Several optimisations of the procedure are possible and are employed in our implementation.

Procedure 2 LAST-SATISFIED($Unreach_k, k, Max, \{P_i\}_{k \leq i \leq Max}$)

```

 $x := k + 1$ 
repeat
   $Reach\_k\_x :=$  all tuples which does not contain a tuple in  $Unreach\_k$ 
  if does not exists  $t \in Reach\_k\_x$  satisfying  $P_k$  then return  $x - 1$ 
   $x := x + 1$ 
until  $x \geq Max$ 
return  $Max$ 
    
```

4.3 Verification of a candidate for bound

In some cases we need to check whether a k -symmetric property (where k is a high number) is satisfied and we know that it is highly probable that the property is satisfied. In these cases, verification using the set of all $k+1$ -tuples is inefficient and verification using an over-approximation of the reachable $k+1$ -tuples is inapplicable. We propose an efficient algorithm for this task. The algorithm is based on the following observation:

A state q to which is assigned $(q_C, q_1, \dots, q_k) \in Reach_k$ is often reachable by a path, which has $k+1$ parts such that for some injective function $i : \{1, \dots, k\} \rightarrow \{1, \dots, k\}$ it holds:

- 1: In the first part, only user 1 and the control component perform actions. User 1 is in the state $q_{i(1)}$ after this part of the path.
- 2: In the second part, only user 2 and the control component perform actions. User 2 is in the state $q_{i(2)}$ after this part of the path.
- ...
- k : In the $k - th$ part, only user k and the control component perform actions. User k is in the state $q_{i(k)}$ after this part of the path.
- $k+1$: In the $k + 1 - th$ part, only the control component and users $k + 1, \dots$, perform actions. The control component is in the state q_C after this part of the path.

$$\underbrace{p_1 \rightarrow p_2 \rightarrow \dots \rightarrow p_{i_1}}_{\text{user 1 and control comp.}} \rightarrow \underbrace{p_{i_1+1} \rightarrow \dots \rightarrow p_{i_2}}_{\text{user 2 and control comp.}} \rightarrow \dots \rightarrow \underbrace{p_{i_{k-1}+1} \rightarrow \dots \rightarrow p_{i_k}}_{\text{user } k \text{ and control comp.}} \rightarrow \underbrace{p_{i_k+1} \rightarrow \dots \rightarrow p_{i_{k+1}}}_{\text{users } k+1, \dots, \text{ and control comp.}} = q$$

Example 4.3 Consider the C-U model from Example 2.4. A state to which is assigned 3+1-tuple $((p_1, q_0, r_1), 2, 0, D)$ is reachable by the path:

$$\begin{aligned}
 & ((p_0, q_0, r_0), 0, 0, 0, 0) \xrightarrow{\text{cFT}} ((p_0, q_0, r_1), 0, 1, 0, 0) \xrightarrow{\text{cFT}'} ((p_0, q_0, r_0), 0, 2, 0, 0) \xrightarrow{\text{iAS}} \\
 & ((p_0, q_0, r_3), 0, 2, B, 0) \xrightarrow{\text{tI}} ((p_0, q_0, r_0), 0, 2, C, 0) \xrightarrow{\text{tI}'} ((p_1, q_0, r_0), 0, 2, D, 0) \xrightarrow{\text{cFT}} \\
 & ((p_1, q_0, r_1), 0, 2, D, 1)
 \end{aligned}$$

We generalise this observation. For a value $parallel \in \{1, \dots, k\}$ the sign $Path_{parallel}^k$ denotes the set of all paths in $C \parallel U^\infty$ such that each path in $Path_{parallel}^k$ has $k + 2 - parallel$ parts (see Figure 5):

- 1: In the first part, only users $1, \dots, parallel$ and the control component perform actions.
- ...
- $k + 1 - parallel$: In the $k + 1 - parallel$ -th part, only users $k + 1 - parallel, \dots, k$ and the control component perform actions.
- $k + 2 - parallel$: In the $k + 2 - parallel$ -th part, only users $k + 2 - parallel, \dots$ and the control component perform actions.

part	component															
	control comp.	1	2	...	<i>parallel</i>	<i>parallel+1</i>	<i>parallel+2</i>	...	<i>k-parallel</i>	<i>k+1-parallel</i>	<i>k+2-parallel</i>	...	<i>k-1</i>	<i>k</i>	<i>k+1, k+2, ...</i>	
1-st	✓	✓	✓	...	✓	×	×	...	×	×	×	...	×	×	×	
2-nd	✓	-	✓	...	✓	✓	×	...	×	×	×	...	×	×	×	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	
<i>k+1-parallel</i> -th	✓	×	×	...	×	×	×	...	×	✓	✓	...	✓	✓	×	
<i>k+2-parallel</i> -th	✓	×	×	...	×	×	×	...	×	×	✓	...	✓	✓	✓	

 Fig. 5. Illustration of parts of a path in $Path_{parallel}^k$.

Example 4.4 Consider the C-U model from Example 2.4 and $2 + 1$ -tuple $((p_1, q_0, r_3), B, 0)$. No state of the form $((p_1, q_0, r_3), B, 0, q'_1, \dots, q'_f)$ or $((p_1, q_0, r_3), 0, B, q'_1, \dots, q'_f)$ is reachable by a path in $Path_1^2$. It is because $Path_1^2$ contains only paths which can be divided into three parts – in its first part only user 1 and the control component perform actions, in its second part only user 2 and the control component perform actions, and in the third part only users $3, \dots$, and the control component perform actions. Thus if it exists a path in $Path_1^2$ satisfying these conditions, then the first state of its third part satisfies that the user 1 is in state B and the user 2 is in the state 0 (or vice versa), all other users are in state 0, and the control component is in state (p_0, q_0, r_3) or (p_0, q_0, r_5) . Consequently in the next steps of the path no user with name greater than 1 is able to start its prepaid or free session. Hence the control component can not move to the state (p_1, q_0, r_3) .

On the other hand, $((p_1, q_0, r_3), 0, B, D)$ is reachable by the finite path in $Path_2^2$:

$$((p_0, q_0, r_0), 0, 0, 0) \xrightarrow{cFT} ((p_0, q_0, r_3), 0, 0, B) \xrightarrow{cFT'} ((p_0, q_0, r_0), 0, 0, C) \xrightarrow{tI} ((p_1, q_0, r_0), 0, 0, D) \xrightarrow{cFT} ((p_1, q_0, r_3), 0, B, D)$$

For a given value *parallel* we denote $Reach_k\widehat{parallel}$ the set of all $k + 1$ -tuples (q_C, q_1, \dots, q_k) such that for some $f \geq k$ and an injective function $i : \{1, \dots, k\} \rightarrow \{1, \dots, k\}$ a state $(q_C, i(q_1), \dots, i(q_k), q'_1, \dots, q'_f)$ is reachable by a path in $Path_{parallel}^k$. This set is an under-approximation of the set $Reach_k$ and therefore it can be used for verification that a k -symmetric formula is satisfied. A computation of the set $Reach_k\widehat{parallel}$ can be divided into steps:

1-st step: A computation of all possible states $(q_C, i(q_1), \dots, i(q_k), q'_1, \dots, q'_f)$ reachable after the first part of a path in $Path_{parallel}^k$. This part can be computed if we traverse the state space of $C\|U^{parallel}$.

i -th step for $i \in \{2, \dots, k + 1 - parallel\}$: A computation of the set of all possible states $(q_C, i(q_1), \dots, i(q_k), q'_1, \dots, q'_f)$ reachable after the i -th part of a path in $Path_{parallel}^k$ from the set of states computed in the $i - 1$ -th step. It can be computed if we several times traverse the state space of $C\|U^{parallel}$ with changed sets of initial states.

$k + 2 - parallel$ -th step: A computation of the set of all possible states $(q_C, i(q_1), \dots, i(q_k), q'_1, \dots, q'_f)$ reachable after $k + 2 - parallel$ -th part of a path

in $Path_{parallel}^k$ from the set computed in the previous step. It can be computed if we find all reachable $parallel+1$ -tuples in $C\|U^\infty$ with a different set of initial states.

To sum it up, in the computation it is enough for each $q \in Q_{C\|U^{parallel}}$ to generate the state space in $C\|U^{parallel}$ reachable from q and to compute $Reach_{-}(parallel-1)$ in a model $C\|U^\infty$ with a different set of initial states. Thus the number of traversed states during this computation is constant in k and exponential in $parallel$. The number of traversed states during a computation of $Reach_k$ is exponential in k , thus for $parallel \ll k$ the required space necessary for computing $Reach_k \hat{=} parallel$ is smaller than the required space necessary for computing $Reach_k$. Thus in cases described at the beginning of this subsection it is effective to search whether a tuple satisfying the k -symmetric property is in $Reach_k \hat{=} 1$, then in $Reach_k \hat{=} 2, \dots, Reach_k \hat{=} k$. All these facts are used in the procedure IS_SATISFIED. Several optimisations of the procedure are possible and are employed in our implementation.

Procedure 3 IS_SATISFIED(C, U, P_k, k)

```

i := 1; n := CUTOFF_FOR_L+1_TUPLES(C, U, 0)
repeat
  if a tuple satisfying  $P_k$  is in  $Reach_k \hat{=} i$  then return true
  i := i + 1
until i > k
 $Reach_k := REACHABLE\ l+1\text{-TUPLES}(C, U, k)$ 
if exists  $t \in Reach_k$  satisfying  $P_k$  then return true
else return false
    
```

4.4 Algorithm BOUND

Procedure 4 BOUND($C, U, Max, \{P_i\}_{1 \leq i \leq Max}$)

```

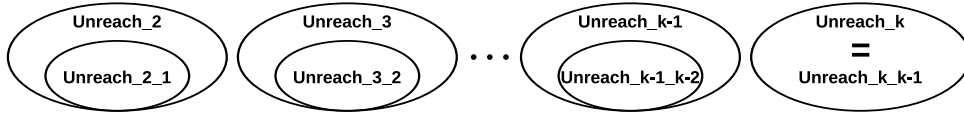
1: k := 0;  $Unreach_k := \emptyset$ ;  $Reach_{k-1} := \emptyset$ 
2: repeat
3:   k := k + 1
4:    $Reach_{k-1} := Reach_k$ 
5:    $Reach_k := REACHABLE\ l+1\text{-TUPLES}(C, U, k)$ 
6:    $Unreach_{k-1} := Unreach_k$ 
7:    $Unreach_k := \text{all } k+1\text{-tuples} \setminus Reach_k$ 
8:   if IS_k-1_BOUND( $Reach_{k-1}, Reach_k, P_{k-1}, P_k, k$ ) then return k - 1
9:    $Changes = Unreach_k \setminus GENERATE\_POSSIBLE\_NEW(Unreach_{k-1})$ 
10: until ( $Changes = \emptyset$ )  $\vee$  (k  $\geq Max$ )
11: b := LAST_SATISFIED( $Unreach_{k-1}, k, Max, \{P_i\}_{k \leq i \leq Max}$ )
12: return VALIDATEBOUND( $C, U, \{P_i\}_{1 \leq i \leq Max}, k, b$ )
    
```

Procedure 5 VALIDATEBOUND($C, U, \{P_i\}_{1 \leq i \leq Max}, k, b$)

```

if IS_SATISFIED( $C, U, P_b, b$ ) then return b
x := k
repeat
  if  $\neg$  (IS_SATISFIED( $C, U, P_x, x$ )) then return x - 1
  x := x + 1
until x  $\geq b$ 
return b - 1
    
```

The bounding algorithm which we propose using the previously described procedures computes for an input C-U system, a sequence of formulas, and Max bound. This algorithm has two main parts. In the first part the algorithm (lines 1-10) computes all reachable $k+1$ -tuples for increasing k until it decides that the k is large enough and the over-approximations $\{Reach_{b,k}\}_{b \in \{1, \dots, Max\}}$ of the sets $\{Reach_b\}_{b \in \{1, \dots, Max\}}$ are sufficient. During each iteration after computing the


 Fig. 6. The value k such that $Changes = \emptyset$.

Property form Example 3.2	Max	Bound	States-RT	States-VB	States-VB[19]
(i)	10	10	$1 \cdot 10^6$	88	10^{10}
(i)	10^{10}	10^{10}	$1 \cdot 10^6$	88	-
(ii)	10^{10}	1	$2 \cdot 10^3$	0	0
(iii)	10	10	$1 \cdot 10^6$	89	10^{11}
(iii)	10^{10}	10^{10}	$1 \cdot 10^6$	89	-
(iv)	10^{10}	4	$1 \cdot 10^6$	88	$6 \cdot 10^3$
(v)	10^{10}	4	$1 \cdot 10^6$	88	10^4

Fig. 7. Evaluation of the verification algorithm.

set $Reach_k$ for any $k > 1$ it computes whether $k - 1$ is *bound* (using the previously computed set of reachable states $Reach_{k-1}$, $Reach_k$). The situation in which the algorithm decides that k is large enough is when all unreachable k -tuples contain an unreachable $k - 1$ -tuple. In other words it is exactly when $Unreach_{k,k-1} = Unreach_k$ and this is exactly when $Reach_{k,k-1} = Reach_k$.

If *bound* is not found in the first phase of the algorithm (*bound* is greater than the value of k after the repeat-until cycle) then the algorithm continues (lines 11, 12). It uses the previously computed set of unreachable k -tuples and from the sets $Reach_{k,k}$, $Reach_{k+1,k}$, ..., $Reach_{Max,k}$ the procedure `LAST_SATISFIED` compute an over-approximation and a candidate for *bound* b . After that the procedure `VALIDATEBOUND` computes, which of the numbers $\{k+1, \dots, b\}$ is *bound*. `VALIDATEBOUND` firstly checks whether b is *bound* and if it is not *bound* then it iteratively checks whether *bound* is $k + 1$, $k + 2$, ..., $b - 1$.

5 Evaluation

Table in Figure 7 displays characteristics of the proposed algorithm `BOUND` for the model of the payment system from Example 2.4, properties from Example 3.2 and a comparison with the original algorithm published in [19]. These characteristics are: *Max*, *bound*, the number *States-RT* of states generated by `REACHABLE` $l + 1$ -TUPLES in the procedure `BOUND`, the number *States-VB* of states generated by `VALIDATEBOUND` in the procedure `BOUND` (sign "0" appears iff the procedure is not called), the number *States-VB[19]* of states generated by the original procedure `VALIDATEBOUND` from [19]³ (sign "-" appears iff if the value is greater than 10^{20} , sign "0" appears iff the procedure is not called).

We applied the algorithm on several other C-U models of Client-Server systems and another types of component-based systems from [2]. Based on the experimental evaluation we conclude:

lines 1-10: Computational time and space of lines 1-10 of the procedure `BOUND` is very similar to the complexity of computing $Reach_k$, where k is the highest

³ The original procedure `VALIDATEBOUND` uses on-the-fly method based on breadth-first search.

number for which the set $Reach_k$ is computed in the procedure BOUND. For all models from [2] the value k is less than or equal to 3. Thus computational time and space of this part of the algorithm is for all studied models similar as the computational time and space of REACHABLE $l + 1$ -TUPLES for input parameter less than or equal to 3. If $bound$ is less than 3 the procedure BOUND computes $bound$ in the repeat-until cycle.

line 12: For all studied models the time and space required by the procedure VALIDATEBOUND are much smaller than the time and space required by the procedures in lines 1-10. Moreover an optimisation of the procedure enables us to compute whether for a given C-U model and a property the value $bound$ is in all cases equal to Max , for all studied models and properties.

Generally, for all studied examples the time and space used in BOUND is similar to the time and space of computing $Reach_3$ no matter how big $bound$ is.

5.1 Comparison with the original algorithm

The algorithm proposed in Section 4 is an extension of the algorithm [19]. The original technique uses the same algorithm for computing of the over-approximation of $bound$ as the algorithm described in this paper (lines 1-11 of the Algorithm BOUND), but for computing of $bound$ from the over-approximation (Algorithm VALIDATEBOUND, line 12) it does not use any optimised technique. Thus the original algorithm effectively solves only a part of the problem – finding the over-approximation of $bound$. The second part of the problem – to prove that the over-approximation is $bound$ – is ineffective in the original algorithm. Experiments show that the computational space of the original algorithm is usually exponential in $bound$ (see Figure 7).

6 Related work

As it is discussed in Section 4 there are several techniques which can be used for solving the problem studied in this paper (e.g. [12,9,17,6,7]). However our experience shows that the computational space of those algorithms is exponential in the value $bound$ and thus their usage for the problem described in this paper is ineffective. To the best of our knowledge, there is no algorithm which can be effectively used for computing of the exact solution of the presented problem. The technique [19] is discussed in Section 4.5.

7 Conclusions

In the paper we propose an automated technique for the analysis of Client-Server systems solving questions like "What is the maximal possible number of clients which can be handled simultaneously?" or "What is the maximal possible number of clients which want to be handled simultaneously and are in some special situation?". The algorithm first finds an over-approximation of the maximal number and then using this approximation efficiently computes the correct result. Using the algorithm we verified models of several previously published systems and for all the systems the over-approximation found in the first part of the algorithm was the correct $bound$.

In future, we aim to finish the implementation of the presented algorithms and evaluate the approach on a large number of realistic case studies. We also aim at studying whether the proposed technique can be used for another problems.

8 Acknowledgement

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

References

- [1] [Http://kraken.cs.cas.cz/ft/public/public_index.phtml](http://kraken.cs.cas.cz/ft/public/public_index.phtml), Accessed May 2008.
- [2] [Http://anna.fi.muni.cz/coin/CUmodels/](http://anna.fi.muni.cz/coin/CUmodels/), Accessed May 2008.
- [3] Abdulla, P. A., K. Cerans, B. Jonsson and Y. Tsay, *General decidability theorems for infinite-state systems*, in: *LICS'96* (1996), pp. 313–321.
- [4] Arons, T., A. Pnueli, S. Ruah, J. Xu and L. D. Zuck, *Parameterized verification with automatically computed inductive assertions*, in: *CAV'01*, LNCS **2102**, 2001, pp. 221–234.
- [5] Brim, L., I. Černá, P. Vařeková and B. Zimmerova, *Component-Interaction Automata as a Verification-Oriented Component-Based System Specification*, ACM SIGSOFT Software Engineering Notes **31** (2006), pp. 1–8.
- [6] Calder, M. and A. Miller, *An automatic abstraction technique for verifying featured, parameterised systems*, Theoretical Computer Science (2008), to appear.
- [7] Clarke, E. M., M. Talupur and H. Veith, *Proving ptolemy right: The environment abstraction principle for model checking concurrent systems*, in: *TACAS'08*, LNCS, 2008, pp. 33–47.
- [8] Emerson, E. A. and V. Kahlon, *Model checking guarded protocols*, in: *LICS'03* (2003), pp. 361–370.
- [9] Fontaine, P. and E. P. Gribomont, *Decidability of invariant validation for parameterized systems*, in: *TACAS'03*, LNCS **2619**, 2003, pp. 97–112.
- [10] German, S. M. and A. P. Sistla, *Reasoning about systems with many processes*, J. ACM **39** (1992), pp. 675–735.
- [11] Jonsson, B. and M. Nilsson, *Transitive closures of regular relations for verifying infinite-state systems*, in: *TACAS'00*, LNCS **1785**, 2000, pp. 220–234.
- [12] Kesten, Y., O. Maler, M. Marcus, A. Pnueli and E. Shohar, *Symbolic model checking with rich assertional languages*, Theor. Comput. Sci. **256** (2001), pp. 93–112.
- [13] Kofroň, J., “Behavior Protocols Extensions,” Ph.D. thesis, Charles University in Prague (2007).
- [14] Lynch, N. A. and M. R. Tuttle, *An introduction to input/output automata*, CWI Quarterly **2** (1989), pp. 219–246.
- [15] Maidl, M., *A unifying model checking approach for safety properties of parameterized systems*, in: *CAV'01*, LNCS **2102**, 2001, pp. 311–323.
- [16] Pnueli, A., S. Ruah and L. D. Zuck, *Automatic deductive verification with invisible invariants*, in: *TACAS'01*, LNCS **2031**, 2001, pp. 82–97.
- [17] Pnueli, A. and L. D. Zuck, *Model-checking and abstraction to the aid of parameterized systems*, in: *VMCAI'03*, 2003, p. 4.
- [18] Rybina, T. and A. Voronkov, *Using canonical representations of solutions to speed up infinite-state model checking*, in: *CAV '02* (2002).
- [19] Vařeková, P. and I. Černá, *Model checking of control-user component-based parametrised systems*, in: *CBSE'08*, LNCS **5282**, 2008.
- [20] Zimmerova, B., P. Vařeková, N. Beneš, I. Černá, L. Brim and J. Sochor, “The Common Component Modeling Example: Comparing Software Component Models, Component-Interaction Automata Approach (CoIn),” LNCS 5153, 2008 pp. 146–176.