Cut-Offs with Network Invariants

Antti Siirtola

Department of Information Processing Science
University of Oulu
PL 3000, 90014 University of Oulu, Finland
firstname.lastname@oulu.fi

Abstract—We consider the multi-parameterised process algebraic verification of safety properties. There is a cut-off result for such verification tasks, but it cannot be naturally applied to systems involving linear parts. We show how the limitation can be overcome by computing a network invariant for each linear part and including all the processes obtained during the computation in the system model.

Keywords—formal verification; parameterised verification; refinement; process algebra; cut-off; induction; network invariant

I. INTRODUCTION

We consider the parameterised verification problem, where parameters are sets and relations over these sets, typically used to denote sets of identities of replicated components and connections between the components. A specification and a system are given as (multiply) parameterised labelled transition systems (LTSs) [1], parameter values are represented as a set of valuations, which are functions mapping parameters to values, and correctness is understood as the traces refinement. The problem is important in practice, as probably all real-life systems can be naturally modelled as multiply parameterised finite-state machines.

The precongruence reduction is a technique for deriving cut-offs for parameterised verification tasks of the above kind [2], [3]. The result gives a bound (cut-off) for the size of parameters such that the system is correct for all the allowed parameter values if and only if the system if correct for all the allowed parameter values of size at most the bound. The advantages of the method are that it preserves both the truth and false, can be automated, and after applying it the remaining finitely many finite-state verification tasks can be solved using existing tools.

However, because the parameterised verification problem is undecidable in general [4], the method has its limitations. The major restriction is that a set of valuations must be downward closed, which intuitively means that the resulting specification-system family has to be closed under the removal of a replicated component. Therefore, the result does not naturally apply to linear, tree or ring systems of an arbitrary size, because removing a node from a chain, tree or ring breaks the structure. Certainly, one can always enlarge a set of valuations such that it becomes downward closed and the result can be applied, but then one has to check also broken chains, trees or rings where synchronisation between certain parts is lost, and it typically leads to false negative verification results.

On the other hand, systems involving linear parts can often be treated with methods based on inductive reasoning [5], [6]. A crucial part of these methods is the construction of a network invariant, which is an abstraction of a (partial) system of an arbitrary size and remains invariant under the addition of a replicated component. A process algebraic version of a network invariant method is known as a behavioural fixed point technique [7], [8]. It can be applied with existing tools, and it can handle parameterised systems having the shape of a chain or nearly so. However, it is also believed that the method does not work outside this domain [9].

Here, we propose combining the precongruence reduction with the behavioural fixed point method to be able to analyse multiply parameterised systems involving both linear and non-linear parts. The idea is that we first iteratively compute a behavioural fixed point for each linear part. The computation is not guaranteed to terminate, but when it does, the LTSs obtained during the computation capture the behaviour of the system from the viewpoint of every pair of connected components that reside in linear parts. Next, we add all these LTSs in the system model. It does not affect the behaviour of the system, but the topology of the system changes in such a way that chains become replaced by their transitive closures. It means that if we remove a replicated component, components that used to be connected are still connected to together. Hence, when we finally replace the original set of valuations with its smallest downward closed superset and apply the cut-off result, we are less likely to get false negative verification results.

To show that the method is useful in the verification of real systems, we apply it to taDOM+ tree-locking protocols used
in XML databases [10]. The protocols exhibit both linear and non-linear parts, because database nodes are organised in the form of a tree and the connections between concurrent transactions and nodes form a bipartite topology. Therefore, neither of the methods alone can be used to verify the protocols, but we show that the combination of the methods will do the job. A mutual exclusion property we consider is proved to the protocols also earlier, but using a theory tailored for them [11]. Here, for the first time, we show in detail how these protocols can be verified with a generic theory.

The contribution of the paper is a novel combination of a cut-off result and a network invariant method. We do not explicitly use network invariants to prove the system correct, but we use them to modify the system model in such a way that the cut-off result can be applied. In this sense, the probably closest related work is done by Creese who also considers multi-parameterised verification by combining a network invariant method with a cut-off result [12]. However, in her approach, a parameter-dependent network invariant is supplied first and after that data independence results of Lazić and Nowak [13], [14] are applied to reduce parameterised verification and invariance checking to a finite verification task. Because its not clear when a network invariant or a behavioural fixed point exists, comparing the application domain of the method with ours is difficult. However, our approach can be seen to be more automatic, because behavioural fixed points are computed from the basis of component LTSs, whereas a network invariant in Creese’s method has to be typically manually provided by the user.

Other approaches that enable parameterised verification by establishing cut-offs for parameter values have been proposed for systems composed of similar fixed-size processes [15], [16], systems of a ring topology [17], [18], [19], [20], [21], networks of homogeneous processes [22], and request-take-release (RTR) systems, where identical processes compete for an access to a fixed number of shared resources under a prioritised queue policy [23]. The results concerning the verification of safety properties of RTR systems [23] and systems with conjunctive guards considered by Emerson and Kahlon [15] can be obtained with (the cut-off result behind) our method, too [3]. Our approach can be applied to systems of a ring topology and networks of arbitrary size as well, but we do not know yet if it is possible to obtain results similar to those in [17], [18], [19], [20], [21], [22]. On the other hand, only the result of Emerson and Kahlon [15] supports multiple parameterised components and only the approach of Clarke et al. [22] allows the topology of a system to be parameterised, but none of the methods have both the properties. Therefore, neither of the cut-off results above can be coupled with a network invariant method, at least in our fashion.

In the next section, we introduce LTSs used as the model of computation and briefly review the related theory. After that, we introduce the cut-off result, which our method is based on, and in the following section, we equip it with the behavioural fixed point method. Finally, Section V provides the taDOM+ case study, and the paper concludes with discussion on future work.

II. LABELLED TRANSITION SYSTEMS

Before we introduce our model of computation, we briefly clarify some notions and notation used throughout the paper. If \( n \) is a non-negative integer and \( e_1, \ldots, e_n \) are any elements, then \( e := (e_1, e_2, \ldots, e_n) \) is called an \((n\text{-}tuple)\) and \( e[i] \) denotes the element \( e_i \) whenever \( i \in \{1, \ldots, n\} \). For any \( n\text{-}tuple \ e \), we write \(|e|_n \) for the dimension \( n \) of \( e \). Moreover, whenever \( e \) is an element, then \(|e|_e \) denotes the number of occurrences of \( e \) in \( e \). If \( S \) is any structure and \( e, e' \) are tuples of elements such that the dimensions of \( e \) and \( e' \) match, we write \( S[e'/e] \) for a structure obtained from \( S \) by substituting \( e'[1], \ldots, e'[|e|] \) for every occurrence of respectively \( e[1], \ldots, e[|e|] \).

If \( f \) is a (partial) function: \( A \to B \), then the domain of \( f \), denoted by \( \text{dom}(f) \), is the set of all the elements in \( A \) for which \( f \) is defined and the image of \( f \), denoted by \( \text{im}(f) \), refers to the set \( \{f(a) \mid a \in \text{dom}(f)\} \). Moreover, whenever \( C \) is a set, then \( f[C] \) denotes a (partial) function: \( A \cap C \to B \) with the domain \( \text{dom}(f) \cap C \) such that \( f[C](a) = f(a) \) for all \( a \in \text{dom}(f[C]) \).

We are now ready to present the model of computation, a labelled transition system (LTS) [1]. Intuitively, an LTS is a graph the vertices of which are called states, the edges are labelled by actions and they are called transitions, and one of the states is marked as the initial one. To introduce LTSs formally, we assume a countably infinite set \( \mathbb{A} \) of atoms. Tuples of atoms are called actions. The empty tuple (), also denoted by \( \tau \), is called the invisible action and the rest of the actions are visible. The set of all the visible actions is referred to by \( \mathbb{V} \).

Definition 1 (LTS). A labelled transition system is a four-tuple \( (S, \Sigma, R, \hat{s}) \), where \( S \) is a non-empty set of states, \( \Sigma \subseteq \mathbb{V} \) is a set of visible actions, \( R \subseteq S \times (\Sigma \cup \{\tau\}) \times S \) is a set of transitions and \( \hat{s} \in S \) is the initial state.

If \( L = (S, \Sigma, R, \hat{s}) \) is an LTS, the set \( S \) is called the state-space of \( L \) and the set \( \Sigma \), also denoted by \( \alpha(L) \), is said to be the alphabet of \( L \). An LTS is finite, if it has a finite state-space and a finite alphabet.

In the analysis of LTSs, we are usually interested in the sequences of visible actions reachable from the initial state. A finite alternating sequence of states and actions, \( s_1 \hat{a}_1 s_2 \ldots \hat{a}_{n-1} s_n \), of an LTS \( L \), is a path in \( L \) from \( s_1 \), if \( (s_i, \hat{a}_i, s_{i+1}) \) is a transition of \( L \) for every \( i \in \{1, \ldots, n-1\} \). A finite sequence of visible actions is a trace of \( L \), if there is a path in \( L \) from the initial state such that the sequence can be obtained from the path by removing all the states.
and the invisible actions. The set of all the traces of \( \mathcal{L} \), the traces of \( \mathcal{L} \) for short, is referred to by \( \text{tr}(\mathcal{L}) \).

An LTS \( \mathcal{L}_2 \) is a traces refinement of an LTS \( \mathcal{L}_1 \), denoted by \( \mathcal{L}_1 \preceq_{\text{tr}} \mathcal{L}_2 \), if \( \mathcal{L}_1 \) and \( \mathcal{L}_2 \) have the same alphabet and \( \text{tr}(\mathcal{L}_2) \subseteq \text{tr}(\mathcal{L}_1) \) [24]. If \( \mathcal{L}_1 \preceq_{\text{tr}} \mathcal{L}_2 \) and \( \mathcal{L}_2 \preceq_{\text{tr}} \mathcal{L}_1 \), denoted by \( \mathcal{L}_1 =_{\text{tr}} \mathcal{L}_2 \), then \( \mathcal{L}_1 \) and \( \mathcal{L}_2 \) are called traces equivalent or one says that \( \mathcal{L}_1 \) is traces equivalent to \( \mathcal{L}_2 \). Clearly, \( \preceq_{\text{tr}} \) is a preorder and \( =_{\text{tr}} \) an equivalence in the set of LTSs.

If both the specification and the system are modelled respectively as LTSs \( \mathcal{L}_{\text{Spec}} \) and \( \mathcal{L}_{\text{Sys}} \), and \( \mathcal{L}_{\text{Sys}} \) is a traces refinement of \( \mathcal{L}_{\text{Spec}} \), then the system cannot do anything more than the specification does. This way, it is not possible to prove that a system does something, but one can still show absence of some unwanted behaviour. Therefore, the traces refinement is applicable in proving so called safety properties [25] only.

A system modelled as an LTS is typically built of smaller LTSs representing its parts. Let \( \mathcal{L}_i = (S_i, \Sigma_i, R_i, \hat{s}_i) \) be an LTS for every \( i \in \{1, 2\} \). The parallel composition of LTSs \( \mathcal{L}_1 \) and \( \mathcal{L}_2 \), denoted by \( (\mathcal{L}_1 \parallel \mathcal{L}_2) \), is a four-tuple

\[
(S_1 \times S_2, \Sigma_1 \cup \Sigma_2, R_{\parallel}, (s_1, s_2)) ,
\]

where \( R_{\parallel} \) consists of all tuples \( ((s_1, s_2), a, (s'_1, s'_2)) \) such that

- \( a \in \Sigma_1 \cap \Sigma_2 \) and \( (s_i, a, s'_i) \in R_i \) for \( i \in \{1, 2\} \), or
- \( a \in (\Sigma_1 \cup \{\tau\}) \setminus \Sigma_j \) and \( i, j \) are different elements in \( \{1, 2\} \) such that \( (s_i, a, s'_i) \in R_i \) and \( s_j = s'_j \).

This definition results to a standard Hoare style parallel composition [24]. According to it, the LTSs can execute a common visible action \( a \) in the parallel composition if and only if both of them agree on its execution, whereas the invisible actions and the visible actions that are in the alphabet of an other LTS only are executed individually.

Before a system LTS is compared against a specification one, the actions irrelevant to the specification are hidden. Let \( \mathcal{L} = (S, \Sigma, R, \hat{s}) \) be an LTS and \( A \) a set of visible actions. The LTS \( \mathcal{L} \) after hiding \( A \), is a four-tuple \( (S, \Sigma \setminus A, R_A, \hat{s}) \), denoted by \( (\mathcal{L} \setminus A) \), where \( R_A \) consists of all tuples \( (s, a, s') \) such that

- \( a \notin A \) and \( (s, a, s') \in R \), or
- \( a = \tau \) and there is \( b \in A \) such that \( (s, b, s') \in R \).

Hence, \( (\mathcal{L} \setminus A) \) is obtained from \( \mathcal{L} \) by changing the actions in \( A \) to the invisible one.

It is easy to see that the structures obtained from LTSs by parallel composition and hiding are LTSs and the operators preserve finiteness. They have also other useful properties [24], [1]. For example, the parallel composition is commutative, associative and idempotent (i.e. \( \mathcal{L} \parallel \mathcal{L} =_{\text{tr}} \mathcal{L} \) for every LTS \( \mathcal{L} \)). Moreover, the traces refinement is preserved under the application of the operators. Hence, the traces refinement is a precongruence and the traces equivalence a congruence. The property is also referred to as compositionality.

Because of commutativity and associativity, the order in which LTSs are composed in parallel is insignificant from the viewpoint of traces equivalence. It means that we can generalise the parallel composition to any finite set of LTSs, provided we are interested in the alphabet and traces only. Let \( I = \{i_1, i_2, \ldots, i_n\} \) be a finite index set and \( \mathcal{L}_i = (S_i, \Sigma_i, R_i, \hat{s}_i) \) an LTS for every \( i \in I \). The parallel composition of LTSs in the set \( \{\mathcal{L}_i\}_{i \in I} \), denoted as \( (\mathcal{L}_{i_1} \parallel \mathcal{L}_{i_2} \parallel \cdots \parallel \mathcal{L}_{i_n}) \), is defined as

\[
(\mathcal{L}_{i_1} \parallel \mathcal{L}_{i_2} \parallel \cdots \parallel \mathcal{L}_{i_n}) ,
\]

where \( I \) is non-empty. In the case of empty \( I \), \( (\prod_{i \in I} \mathcal{L}_i) \) is defined as a single state LTS \( \{(()\}, \emptyset, \emptyset, ()\} \) with the empty alphabet.

### III. CUT-OFFS BY THE PRECONGRUENCE REDUCTION

The problem when modelling real systems and specifications as LTSs is that LTSs cannot naturally express parameterised systems. That is why we allow the use of two kinds of variables in LTSs. Type variables denote sets of atoms and they are used to represent sets of identities of system components of the same kind. Relation variables denote sets of tuples of atoms (i.e. sets of visible actions), and they are used to describe the topology of a system and relationships between its components. The sets of all the type and relation variables are denoted respectively by \( T \) and \( \mathcal{G} \).

Our extended LTS formalism allows specifications of the form

\[
(\forall y_1 \in \Xi_1 : U_1, \ldots, y_m \in \Xi_m : U_m) (Q_1) \parallel \cdots \parallel (Q_m) ,
\]

where \( m \) is a positive integer and for all \( i \in \{1, \ldots, m\} \),

- \( Q_i \) is a finite LTS whose behaviour depends on
- a tuple \( y_i \) of distinct atoms thought as parameters of \( Q_i \),
- \( U_i \) is a tuple of type variables of the same dimension as \( y_i \), and
- \( \Xi_i \) is a relation variable representing a relation over the values of type variables in the tuple \( U_i \).

Systems, in turn, are assumed to have the form

\[
(\forall x_{i_1} \in \Pi_1 : T_1, \ldots, x_{i_k} \in \Pi_k : T_k) (P_1) \parallel \cdots \parallel (P_k) \setminus \bigcup_{z \in \mathcal{V}} \Gamma ,
\]

where \( k \) is a positive integer, \( P_1, \ldots, P_k, x_1, \ldots, x_k, T_1, \ldots, T_k, \) and \( \Pi_1, \ldots, \Pi_k \) are defined analogously to respectively \( Q_1, \ldots, Q_m, y_1, \ldots, y_m, U_1, \ldots, U_m, \) and \( \Pi_1, \ldots, \Pi_m \).

- \( \Gamma \) is a finite set of visible actions which depends on
- a tuple \( z \) of distinct atoms thought as parameters of \( \Gamma \), and
- \( \mathcal{V} \) is a tuple of type variables of the same dimension as \( z \).

Hence, we allow multiple parameterised components and the behaviour of a single component may depend on multiple
parameters. Moreover, if a relation $G$ represented by a
relation variable $\Pi$ consists of $l \in \mathbb{Z}_+$ independent parts
$G_1, \ldots, G_l$ (i.e. $G = G_1 \times \ldots \times G_l$), it is allowed to represent
a replicated parallel composition $\|_{(x_1, \ldots, x_n) \in \Pi: (T_1, \ldots, T_n)} P$ in the form

$$
\|_{(x_{n_1}, \ldots, x_{n_2}) \in \Pi: (T_{n_1}, \ldots, T_{n_2})} \|_{(x_{n_1-1}, \ldots, x_{n_2}) \in \Pi: (T_{n_1-1}, \ldots, T_{n_2})} P,
$$

where $l \in \mathbb{Z}_+$ and $1 = n_1 < \ldots < n_l = n$, because nested
replicated parallel compositions can always be combined in one.

Parameter values are formally represented as a valuation. A
valuation is a function that maps type variables to disjoint,
finite, non-empty sets of atoms, and relation variables to
relations over these sets. A valuation $\phi$ is said to be
compatible (with the specification and the system of the
above form), if

- $\phi$ is defined for all the relation variables
  $\Xi_1, \ldots, \Xi_m, \Pi_1, \ldots, \Pi_k$ and all the type variables
  occurring in the tuples $U_1, \ldots, U_m, T_1, \ldots, T_k$,
- no atom occurring in the system or specification occurs
  in the image of $\phi$, and
- $\phi$ maps $\Xi_j$ to a subset of $\phi(U_j[1]) \times \ldots \times \phi(U_j[t])$
  and $\Pi_i$ to a subset of $\phi(T_i[1]) \times \ldots \times \phi(T_i[t])$
  for all $j \in \{1, \ldots, m\}$ and for all $i \in \{1, \ldots, k\}$.

Let $Spec$ be a specification of the form (1), $Sys$ a system
of the form (2) and $\phi$ a compatible valuation. When the
values of type and relation variables in $Spec$ and $Sys$ are
fixed using $\phi$, (standard) LTSs are obtained. Formally, an
instance of $Spec$ generated by $\phi$, denoted by $[Spec]_\phi$, is
defined as

$$
( \|_{a \in \phi(\Xi_1)} Q_1[a/y_1] ) \| \ldots \| ( \|_{a \in \phi(\Xi_m)} Q_m[a/y_m] ),
$$

which is clearly an LTS. Similarly, an instance of $Sys$
generated by $\phi$ is denoted by $[Sys]_\phi$ and defined as

$$
( \|_{a \in \phi(\Pi_1)} P_1[a/x_1] ) \| \ldots \| ( \|_{a \in \phi(\Pi_k)} P_k[a/x_k] ) \setminus \Lambda,
$$
where $\Lambda$ is the set $\bigcup_{a \in \phi(\Xi[1]) \times \ldots \times \phi(\Xi[|\Xi|])} \Gamma[a/\Xi]$ of
disable actions. Clearly, $[Sys]_\phi$ is an LTS, too.

We may now assume that a specification $Spec$ and a
system $Sys$ are given as variable equipped LTSs of the
respectively forms (1) and (2), and the parameter values are
given as a potentially infinite set $\Phi$ of valuations. There
is usually no need to restrict the values of type variables
anymore, because one typically wants to know whether
the system works correctly for any number of replicated
components. On the other hand, it makes sense to restrict the
values of relation variables as they determine the topology of
the system and relationships between the components, and
one is usually not interested in all the possible topologies
and arbitrary connections between the components. The
parameterised verification task we are interested in can be
now stated as a question whether $[Spec]_\phi \models tr \ [Sys]_\phi$ for
all compatible valuations $\phi \in \Phi$.

Without loss of generality, we may assume that the
valuations in $\Phi$ define values to exactly the type and relation
variables occurring in $Spec$ and $Sys$. However, without any
further restrictions the parameterised verification problem
defined this way cannot be algorithmically solved, because
the set $\Phi$ can be very complex. However, if $\Phi$ is regular
enough, then there is a cut-off result that establishes upper
bounds for the values of variables such that the system is
correct with respect to the specification for all the parameter
values if and only if it is correct with respect to the
specification for all the parameter values at most the cut-
offs. To represent the result, we need to define two relations
in the set of valuations.

Let $\phi$ and $\psi$ be valuations. We say that $\phi$ and $\psi$ are
isomorphic, denoted by $\phi \simeq \psi$, if the valuations can be
obtained from each other by bijective mapping of atoms, i.e.
$\phi$ and $\psi$ have the same domain and there is a bijection
g : $\mathbb{A}$ $\rightarrow \mathbb{A}$ such that

1) $\{g(a) \mid a \in \phi(T)\} = \psi(T)$ for all type variables $T$ in the
domain and

2) $\{(g(a_1), \ldots, g(a_n)) \mid (a_1, \ldots, a_n) \in \phi(\Pi)\} = \psi(\Pi)$
for all relation variables $\Pi$ in the domain.

Moreover, $\psi$ is said to be smaller than (or equal to) $\phi$, denoted
by $\psi \preceq \phi$, if $\phi$ and $\psi$ have the same domain,

1) $\psi(T) \subseteq \phi(T)$ for all type variables $T$ in the domain,
and

2) $\phi(\Pi) = \psi(\Pi) \cap (\bigcup_{A_1, \ldots, A_n} \im(\psi[\phi]) A_1 \times \ldots \times A_n)$
for all relation variables $\Pi$ in the domain.

In other words, $\psi$ is smaller than $\phi$ if it can be obtained
from $\phi$ by cutting down the values of type variables and
restricting the values of relation variables accordingly.

We say that a set $\Psi$ of valuations is isomorphism closed,
if whenever $\psi$ is a valuation in $\Psi$ and $\psi'$ is a valuation
isomorphic to $\psi$, then $\psi' \in \Psi$, too. Moreover, the set $\Psi$ is
downward closed, if whenever $\psi$ is a valuation in $\Psi$ and
$\psi'$ is a valuation smaller than $\psi$, then $\psi' \in \Psi$, too. If the
set $\Phi$ encoding the parameter values is both isomorphism
and downward closed and consists of valuations that define
values to precisely the type and relation variables occurring
in the specification and the system, then $\Phi$ can be finitely
expressed using the universal fragment of first order logic
and the parameterised verification problem becomes decide-
able through the following cut-off result [3].

**Theorem 1.** Let $Spec$ be a specification of the form (1),
$Sys$ a system of the form (2), and $\Phi$ an (infinite) isomor-
phicism and downward closed set of valuations that define
values precisely to type and relation variables occurring in
Spec and Sys. Then any maximal set $\Psi$ of non-isomorphic
Theorem 1 is not naturally applicable to systems with a linear, ring or tree topology, because then one has to check also broken chains, trees or rings where synchronisation between certain parts is lost, which typically leads to false negative verification results.

To show how to overcome the problem, let us assume that our system is of the form (2) and the relation variable \( \Pi_k \) represents a binary relation whose transitive closure is irreflexive. Hence, \( T_k \) is a 2-tuple \((T,T)\) for some type variable \( T \) and \( x_k \) is a 2-tuple \((l,r)\) for some distinct atoms \( l,r \). This is the case when \( \Pi_k \) represents for example a linear or tree topology and \( P_k \) captures the behaviour of the system from the viewpoint of adjacent components \( l \) and \( r \), which are of the type \( T \). The method is applicable to rings and other structures that contain loops as well, one just has to make sure that all the computations are performed for linear parts only.

To make sure that synchronisation between connected components is not lost when an intermediate node is removed, we try capture the behaviour of the system from the viewpoint any two components connected to each other. Let \( P_k^1 := P_k \) and

\[
P_k^{i+1} := (P_k^i[b/r] \parallel P_k[b/l]) \setminus \alpha(P_k)
\]

for every positive integer \( i \), where \( b/r \) is any atom not occurring in the alphabet of \( P_k \) and \( \alpha(P_k) \) denotes the set of all the visible actions not in the alphabet of \( P_k \). In other words, \( P_k^{i+1} \) represents the behaviour of the system from the viewpoint of components \( l \) and \( r \) connected to each other via \( i \in \mathbb{N} \) other similar components.

If \( P_k^1 =_{tr} P_k^2 \), then we know that the system looks the same from the viewpoint of two adjacent components as from the viewpoint of two components connected to each other via another similar component. Actually, by induction and compositionality, it implies that for every positive integer \( i \),

\[
P_k^{i+1} = (P_k^i[b/r] \parallel P_k[b/l]) \setminus \alpha(P_k)
\]

which means that the system looks the same from the viewpoint of any two components connected to each other.

Moreover, if \( \phi \) is a compatible valuation in \( \Phi \) and \( \phi_1 \) is a valuation otherwise like \( \phi \) except that \( \phi_1(\Pi_k) \) is the transitive closure of \( \phi(\Pi_k) \), then

\[
\llbracket((\forall x \in \Pi_1:T_1 \parallel \ldots \parallel x_k \in \Pi_k:T_k) \parallel P_1) \parallel \ldots \parallel (\forall x \in \Pi_l:T_l \parallel \ldots \parallel x_k \in \Pi_k:T_k)\rrbracket_{\phi_1} =_{tr} \llbracket((\forall x \in \Pi_1:T_1 \parallel \ldots \parallel x_k \in \Pi_k:T_k) \parallel P_1) \parallel \ldots \parallel (\forall x \in \Pi_l:T_l \parallel \ldots \parallel x_k \in \Pi_k:T_k)\rrbracket_{\phi}.
\]

That is because by commutativity and associativity of the parallel composition and the fact that \( L \parallel (L \setminus \Lambda) =_{tr} L \) for every LTS \( L \) and any set \( \Lambda \) of visible actions, the behaviour of an system instance does not change if we add instances of \( P_k \) that are obtained from those already contained in
the system using parallel composition and hiding. Therefore, without the risk of false verification results, we can replace the value of $P_k$ with its transitive closure. In other words, instead of $\Phi$, we use a set $\Phi_1$, which consists of all valuations $\phi_1$ such that $\phi_1$ is otherwise like a valuation $\phi \in \Phi$ but maps $P_k$ to the transitive closure of $\phi(P_k)$.

If there are no other linear parts, the set $\Phi_1$ is downward closed and typically also isomorphism closed. Otherwise, we can enlarge the set $\Phi_1$ to a minimal isomorphism and downward closed one and try to verify the system with the aid of Theorem 1. There is also a good chance to avoid false negative verification results, because now the behaviour of the system is explicitly described from the viewpoint of any two components of the type $T$ connected to each other, which implies that synchronisation between the components is not, at least totally, lost when some of them are removed.

However, if $P_k$ and $P_{k+1}$ are not traces equivalent, then we check whether $P_k$ and $P_{k+1}$ are. If they are, then, like above, we can prove that $P_{k+2} =_\text{tr} P_k$ for every non-negative integer $i$, which means that the system looks the same from the viewpoint of any two components connected to each other through one or more similar components. Now, we extend the system by adding a new component $P_{k+1} := P_k$ and a new relation variable $\Pi_{k+1}$, which represents pairs of components of the type $T$ connected through one or more similar components. Therefore, the system gets the form

$$\left( \bigcup_{x \in V} \Gamma \right),$$

where $x_{k+1} = x_k = (l, r)$ and $T_{k+1} = T_k = (T, T)$.

Like above, we can show that if $\phi$ is a compatible valuation in $\Phi$ and $\phi_1$ is a valuation otherwise like $\phi$ except that it maps $P_{k+1}$ to the transitive closure of $\phi(P_k)$ minus $\phi(P_k)$, then

$$\left( \bigcup_{x \in V} \Gamma \right),$$

Hence, without the risk of false verification results, we can replace the original system and the set $\Phi$ of valuations with the modified system and a set $\Phi_2$ which consists of all valuations $\phi \in \Phi$ extended to the set $\{\Pi_{k+1}\}$ such that $\Pi_{k+1}$ is mapped to the transitive closure of $\phi(P_k)$ minus $\phi(P_k)$.

The set $\Phi_2$ is not downward closed, but again, because of system modifications, if we remove a component of the type $T$ synchronisation between components is not (totally) lost. Hence, when we enlarge the set $\Phi_2$, to a minimal isomorphism and downward closed one, we are less likely to get false negative verification results.

In general, we compute LTSs (3) until we find a non-negative integer $c$ such that $P_{c+1} =_\text{tr} P_{c+1}$, which by above implies that $P_{c+1} =_\text{tr} P_{c+1}$ for every non-negative integer $i$. In other words, we compute a behavioural fixed point for the systems of the form (3). Such a fixed point does not exist in general, but when it does, it means that the system looks the same from the viewpoint of any two components of the type $T$ connected to each other via $c$ or more other similar components. Hence, the behaviour of any two connected components can be captured in $c + 1$ LTSs.

If a positive integer $c$ above exists, then the system gets the form

$$\left( \bigcup_{x \in V} \Gamma \right),$$

where for all $i \in \{1, \ldots, c\}$, $P_{k+i} := P_{k+1}$, $\Pi_{k+i}$ is a new relation variable, $x_{k+i} = x_k = (l, r)$ and $T_{k+i} = T_k = (T, T)$. Moreover, each valuation $\phi \in \Phi$ is extended to the set $\{\Pi_{k+1}, \ldots, \Pi_{k+c}\}$ such that the extended valuation $\phi'$ maps $\Pi_{k+i+1}$ to the set

$$\{(a_1, a_2) \mid \exists b: (a_1, b) \in \phi'(\Pi_{k+i}) \wedge (b, a_2) \in \phi'(\Pi_k)\}$$

for every $i \in \{0, \ldots, c - 2\}$, and the value of $\Pi_{k+c}$ is the transitive closure of $\phi(P_k)$ minus the values of $\Pi_{k, k+1}, \ldots, \Pi_{k+c-1}$. We write $\Phi'$ for the set of all such extended valuations. Like earlier, one can show that

$$\left( \bigcup_{x \in V} \Gamma \right),$$

whenever $\phi$ is a compatible valuation in $\Phi$ and $\phi'$ is its extended equivalent. Therefore, replacing the original system and the set $\Phi$ with the extended system and the set $\Phi'$ cannot lead to false verification results.

The set $\Phi'$ is not downward closed in general, which means we cannot apply Theorem 1 directly. On the other hand, we have appended the system description in such a way that the behaviour of the system from the viewpoint of any two components connected to each other is covered by some $P_i$, where $i \in \{k, \ldots, k+c\}$. Therefore, if we remove a component of the type $T$, synchronisation between components is not lost, at least totally. Hence, when we replace the set $\Phi'$ with its minimal isomorphism and downward closed superset, there is a good chance of avoiding false negative verification results.

V. Verifying taDOM+ Tree-Locking Protocols

We have applied the method to prove a mutual exclusion property for taDOM+ tree-locking protocols used in XML databases [10]. A taDOM+ protocol allows a tree-shape database to be accessed through transactions which perform sequences of read and write actions to nodes in the tree. To guarantee correct isolation between the transactions, there are several locks that enable a node, a node together with its children or a whole subtree to be locked for reading or writing. Additionally, there are weaker read and write
intention locks that should be requested on all the ancestors of a read or write locked node. After a transaction has successfully requested appropriate locks to a node and all its ancestors up to the root, it can access the node, its children or descendants depending on the mode of the lock on the node. When a transaction ends all its locks are released.

Currently, there are two taDOM+ protocols, namely taDOM2+ and taDOM3+ compliant with respectively Domain Object Model (DOM) levels 2 and 3 [10]. A taDOM+ protocol describes the lock modes, provides rules to update a lock when a transaction makes multiple requests on a node, and specifies compatible lock modes, i.e. the pairs of lock modes that can be used to lock a node by two different transactions. The most important part of these protocols is a lock manager that takes care of locking nodes and edges, but we consider only tree-locking part here.

The property we are interested in is known as repeatable-read [26] in the context of databases. It says that reading a node within a transaction should always give the same result unless the transaction itself has updated the node. In other words, if a transaction accesses a node, no other transaction should be able to write to the node before the transaction ends, and if a transaction starts writing to a node, no other transaction should be able to access the node before the transaction ends. The same property is earlier proved by Siirtola and Valenta [11] but using a theory tailored for the protocols. Here, for the first time, we show in detail how these protocols can be verified with a generic theory.

We consider the protocols parameterised by the number of transactions and the size and the shape of a tree. We use type variables T and N to refer to respectively sets of identifiers of transactions and nodes, and a relation variable <p to represent a tree over the nodes such that a node n1 is related to a node n2 by <p if and only if n1 is the parent of n2. We use also other relation variables to compose the protocol model, but their values can be derived from the values of T, N and <p.

A protocol model is created from several parts composed in parallel. Each part represents the behaviour of the system from the viewpoint of finitely many components. This kind of compositional modelling technique is natural and widely used in practice. It gives an abstraction (over-approximation) of the system if each part covers all the behaviours from the viewpoint of its alphabet [3]. Hence, such a model can be safely used to prove the system correct but not incorrect, in general.

Obviously, we have to model the protocol from the viewpoint of a transaction t and reading and writing to a node n2. Because a lock that entitles reading or writing to the node may be in any of its ancestors, the model must involve an ancestor n1 of n2 as well. Here, an ancestor of a node can also be the node itself, and the notion proper ancestor is used to refer to an ancestor other than the node itself. Actually, we need three different models, denoted by Tn,

\[ Tc \]  and \[ Ts \], depending on whether respectively n1 represents the same node as n2, n1 is the parent of n2, or n1 represents an other proper ancestor of n2. That is because lock modes have three kinds of scopes which ranges from a node to itself, from a node to its children, or from a node to all its descendants. Each of the models allows the transaction t to read and/or write to the node n2 repeatedly until it ends when the transaction sees the node n1 to be appropriately locked or an appropriate lock to be successfully requested on n1.

Because lock requests of a transaction t on a node n2 and its ancestors n1 are mutually dependent, we need to model t from the viewpoint of locking a node n2 and its parent n1, too. This behaviour is captured in an LTS \[ Tp \] in Fig. 1, where L denotes the set of all the lock modes and \[ prv \] denotes a function \[ L \rightarrow L \] that given the mode of the lock on a node returns the lock mode that must be requested on the parent [10], [11]. The actions \( (t, lockbeg) \) and \( (t, lockend) \) denote respectively that the transaction t starts and finishing locking a path of nodes. The action \( (t, end) \) represents the end of the transaction t and \( (t, n, lock, l_1, l_2) \) indicates a successful lock request of the transition t on a node n, where \( l_1 \in L \) is a requested lock mode and \( l_2 \in L \) a granted lock mode, which is never weaker than \( l_1 \). Hence, the LTS \[ Tp \] formally states if the transaction t requests a lock on n2, then based on the result of the request a lock is requested on the parent n1 as well, but it is also possible that only n1 or neither of the nodes are in the locking path.

Finally, as also lock requests of different transactions on the same node are mutually dependent, the protocol has to be modelled from this viewpoint as well. For that purpose, we introduce an LTS denoted by \[ Nd_2 \], which stores the lock of a transaction \( t_1 \) on a node n and restricts the requests of a transaction \( t_2 \) to those that result to a lock mode compatible with the mode of the lock of \( t_1 \). To correctly present the behaviour of the protocol in the presence of one transaction only, we also need an LTS, denoted by \[ Nd_1 \], which just stores a lock of a transaction t on a node n2.

We can now construct the protocol model by putting all
the parts in parallel. Hence, in our formalism, a protocol model, denoted by \( TDM \), is expressed as

\[
\begin{align*}
\forall (t, t_1, t_2, n_1, n_2) \in (T, T, T(n_1, n_2)) & \rightarrow T(n_1, n_2) \\
\forall (t, t_1, t_2, n_1, n_2) \in (T(n_1, n_2) \in \mathcal{N}(N,N)) & \rightarrow T(n_1, n_2) \\
\forall (t, t_1, t_2, n_1, n_2) \in (T(n_1, n_2) \in \mathcal{N}(N,N)) & \rightarrow T(n_1, n_2) \\
\forall (t, t_1, t_2, n_1, n_2) \in (T(n_1, n_2) \in \mathcal{N}(N,N)) & \rightarrow T(n_1, n_2)
\end{align*}
\]

where \(*_T\) and \(*_N\) are relation variables representing the same sets as respectively \( T \) and \( N \), \( \mathcal{N}(T) \) denotes the set of all pairs of different transaction identities, \( \mathcal{N}(N,N) \) represents the set of all pairs \( (n_1, n_2) \) of nodes such that \( n_1 = n_2 \), and \( \mathcal{N}(T) \) denotes the set of all pairs \( (n_1, n_2) \) of nodes such that \( n_1 \) is a proper ancestor of \( n_2 \) other than its parent.

The same modelling technique can be applied to the specification as well. However, now one has to additionally make sure that every illegal behaviour is blocked by some of the parts. This way we are guaranteed to get a precise formalisation of the specification [3].

To formalise the repeatable-read property, note that every illegal behaviour can be traced back to two transactions \( t_1, t_2 \) that write to a node \( n_3 \) simultaneously based on the locks they have on respectively some ancestors \( n_1, n_2 \). Therefore, we first capture the specification from this point of view in an LTS denoted by \( RR_2 \). However, to correctly present the specification also in the presence of one transaction only, we model it also from the viewpoint of a single transaction \( t \) that accesses a node \( n_2 \) based on the lock on an ancestor node \( n_1 \). This LTS is denoted by \( RR_1 \). The formal specification can be now created by composing all the parts in parallel, which results to

\[
RR := \begin{cases} \text{(t1,t2)} \in (T;T(T,T)) (n_1,n_2,n_3) \in \mathcal{N}(N,N,N) ; & \text{RR2} \\
\text{\hspace{1cm}} \forall (t, t_1, t_2, n_1, n_2, n_3) \in \mathcal{N}(N,N,N) ; & \text{RR1} \end{cases}, \tag{4}
\]

where relation variables \( \mathcal{N}(T) \) and \( \mathcal{N}(N,N) \) denote respectively the sets of all triplets \( (n_1, n_2, n_3) \) and pairs \( (n_1, n_2) \) of node identities such that \( n_1 \) and \( n_2 \) are ancestors of \( n_3 \).

Finally, before it makes sense to compare the protocol model against the formal specification, we hide actions not involved in the specification. We write \( LA \) for the set of all the locking actions a transaction \( t \) can perform on nodes \( n_1 \) and \( n_2 \). We also need to specify the allowed parameter values as a set of valuators. We denote this set by \( \Phi_{TDM} \) and it consists of all valuators \( \phi \) which define values to precisely the variables occurring in \( TDM \) and \( RR \) as explained above. The correctness of taDOM2+ tree-locking protocol can be now stated as a question whether

\[
\begin{align*}
\forall \phi \in \Phi_{TDM} & \rightarrow TDM \hspace{1cm} \bigcup_{(t, n, n_2) \in (T,N,N)} LA \phi \\
\end{align*}
\]

for all compatible valuators \( \phi \in \Phi_{TDM} \).

The set \( \Phi_{TDM} \) is clearly isomorphism closed but not downward closed, because \( \phi(<p) \) defines a tree topology for all \( \phi \in \Phi \). Therefore, Theorem 1 cannot be directly applied. The simplest solution is to try to replace the set \( \Phi_{TDM} \) with its smallest downward closed superset, but it results to a false negative verification result.

The reason is that when we pick a valuation \( \phi \in \Phi \), then \( (n, n') \in \phi(<a) \) (i.e. \( n \) is a proper ancestor of \( n' \) other than the parent) if and only if there are nodes \( n_1, n_2, \ldots, n_k \) such that \( n_1 = n_2 \) and \( <a \) denotes the set of all pairs \( (n_1, n_2) \) of nodes such that \( n_1 \) is a proper ancestor of \( n_2 \) other than its parent.

However, for valuators \( \phi' \) in the smallest downward closed superset of \( \Phi_{TDM} \) only the “if” direction holds. Hence, there may be a pair \( (n, n') \in \phi'(<a) \) but no \( n_1, n_2, \ldots, n_k \) such that \( n = n_1, (n_i, n_{i+1}) \in \phi'(<p) \) for all \( i \in \{1, \ldots, k-1\} \), and \( n_k = n' \). The problem is that the values of \( <a \) are used to instantiate the LTS \( Ts \), which represents the behaviour of a transaction from the viewpoint of accessing a node based on the lock on its proper ancestor other than the parent, so a transaction gets an access to \( n' \) if \( n \) is appropriately locked.

On the other hand, locking proceeds on the basis of the value of \( <p \), so if there is no path from \( n \) to \( n' \) according to \( \phi'(<p) \), it means that a transaction may lock \( n' \) without locking \( n \) at all. Therefore, when \( \Phi_{TDM} \) is enlarged to a minimal isomorphism and downward closed superset, it is possible that one transaction puts a subtree write lock on \( n \) to get a write access to \( n' \) and the other transaction locks \( n' \) directly for writing without locking \( n \), which violates the specification but is impossible in practice.

To overcome the problem, we should capture the behaviour of a transaction \( t \) from the viewpoint of locking a node \( n_2 \) and its arbitrary proper ancestor \( n_1 \). To do that, we apply the method presented earlier. Let \( Tp^1 := Tp \) and

\[
Tp^{i+1} := (Tp^i[b/n_2] \parallel Tp[b/n_1]) \chi \alpha(Tp)
\]

for every positive integer \( i \), where \( b \) is an atom not occurring in \( Tp \). For every non-negative integer \( i \), the LTS \( Tp^{i+1} \) represents the behaviour of a transaction \( t \) from the viewpoint of locking a node \( n_2 \) and its proper ancestor \( n_1 \) connected through \( i \) other nodes. Using a refinement checker FDR2, one can see that \( Tp^1 \) is not traces equivalent to \( Tp^2 \), but \( Tp^2 =_t^1 Tp^3 \), which by the previous section implies that \( Tp^2 =_t Tp^{i+2} \) for every non-negative integer \( i \). In other words, \( Tp^2 \) represents the behaviour of a transaction \( t \) from the viewpoint of locking a node \( n_2 \) and its proper ancestor \( n_1 \) connected through one or more other nodes. By above, it means that we can use

\[
TDM \parallel \left( \bigcup_{(t, n, n_2) \in (T,N,N)} LA \phi \right)
\]

in place of \( TDM \) without affecting the behaviour of the system.

Unfortunately, when we replace the set \( \Phi_{TDM} \) with its smallest downward closed superset, we still get a non-
realistic negative verification result. This time the problem is that $T_p^2$ is too abstract, because it does not take into account that a lock to be requested on a proper ancestor does not only depend on the lock requested on the node but also on the locks on the intermediate nodes. That is why we compose $Nd_1$, which stores a lock on a node $n_2$, in parallel with $T_p$. Note that it does not affect the behaviour of the system, because $Nd_1$ is included in the protocol model for every transaction and every node and the parallel composition operator is commutative, associative and idempotent.

Like above, we first compute $(T_p \parallel Nd_1)^1 := T_p \parallel Nd_1$ and then

$$(T_p \parallel Nd_1)^{i+1} := ((T_p \parallel Nd_1)[b/n_1] \parallel (T_p \parallel Nd_1)[b/n_2]) \cup \alpha(T_p \parallel Nd_1),$$

where $b$ is an atom not occurring in $T_p \parallel Nd_1$, for positive integers $i$ in an increasing order until computation converges.

Again, we find out that $(T_p \parallel Nd_1)^1$ is not traces equivalent to $(T_p \parallel Nd_1)^2$, but $(T_p \parallel Nd_1)^2 =_t (T_p \parallel Nd_1)^3$, which implies that we can use

$$TDM' := TDM \parallel \left( \begin{array}{c} \vdots \end{array} \right) (T_p \parallel Nd_1)^2 \right)$$

in place of $TDM$ without affecting the behaviour of the system.

Now, we replace the set $\Phi_{TDM}$ with its smallest downward closed superset and apply Theorem 1 again. Because $T$ and $N$ occur respectively at most two and three times in the nested tuples of type variables in the specification (4) and the protocol model (6), by Theorem 1 it is sufficient to check all the instances of the system generated by valuations that map $T$ and $N$ to sets of respectively at most two and at most three values. In other words, it is sufficient to check all the instances of the protocol model with at most two transactions and at three nodes.

We still need to determine a maximal set of compatible non-isomorphic valuations that map $T$ and $N$ to sets of size two and three, respectively. To see which valuations we should pick, note that one to three nodes can be organized into a forest in seven different ways and each edge can represent a link between a node and its parent or a link between a node and its other proper ancestor, which gives a total of 14 different forests. As each of these forests can be coupled with one or two transactions, there are altogether 28 valuations to consider per protocol. The instances of the specification and the system generated by these valuations were encoded in CSP modelling language. In each of the cases, the protocol model was found to be correct, which by Theorem 1 implies that

$$[RR]_ \phi \subseteq_{tr} [TDM'] \bigcup_{(t,n_1,n_2) \in (T,N,N)} LA]_ \phi$$

for all compatible valuations $\phi \in \Phi$. Hence, both the taDOM+ tree-locking protocols satisfy the repeatable-read property.

VI. Conclusions

We have shown how a cut-off result [2], [3] and a behavioural fixed point method [7], [8] can be combined to a multi-parameterised verification method that is more powerful than neither of the methods alone. The method is used to prove a mutual exclusion property to taDOM+ tree-locking protocols [10] used in XML databases.

An obvious topic for future research is implementing the method in a tool. From the theoretical side, interesting questions are when the computation of a behavioural fixed point is guaranteed to terminate and under which restrictions our approach is complete. The former question is already partially answered by Nazari and Thistle [27], and a starting point for the latter task is established by Emerson and Namjoshi [19], Emerson and Kahlon [17], [18] and Clarke et al. [22] who provide cut-off results to ring systems and networks of an arbitrary size. It will be interesting to see whether similar or strictly more general results can be achieved also in our framework.

Acknowledgement

The research is mainly funded by the Ministry of Education of Finland through Infotech Oulu Graduate School and partly by the department, and supported by Oulu University Scholarship Foundation and the Foundation of Tauno Tönning. We thank Michal Valenta and Pavel Strnad for discussions on taDOM+ protocols.

References


