REIHE INFORMATIK
TR-2009-004

# Reachability in Tree-Like Component Systems is PSPACE-Complete

Mila Majster-Cederbaum and Nils Semmelrock*

Department of Computer Science
University Mannheim
Mannheim, Germany

**Abstract.** The reachability problem in component systems is PSPACE-complete. We show here that even the reachability problem in the subclass of component systems with "tree-like" communication is PSPACE-complete. For this purpose we reduce the question if a Quantified Boolean Formula (QBF) is true to the reachability problem in "tree-like" component systems.

## 1 Introduction

In component-based modeling techniques the size of the global state space of a system is in the worst case exponential in the number of its components. This problem is often referred to as the effect of state space explosion. Thus, checking properties of a component-based system by exploring the state space very quickly becomes inefficient.

As a formal model for component-based systems we consider here interaction systems [7], a formalism for component-based modeling which offers in general an arbitrary degree of synchronization. Reachability in general interaction systems was proved to be PSPACE-complete [12] similar to results in 1-safe Petri nets [5].

Tree-like component systems are component systems where the communication structure forms a tree. This is an important class of systems which has been early studied e.g. in [8, 4] and more recently e.g. in [3, 10].

Here we show that even in the subclass of "tree-like" interaction systems the reachability problem (and therefore proving deadlock-freedom as well) is PSPACE-complete. We also sketch that deciding progress in "tree-like" interaction systems is PSPACE-complete.

## 2 Interaction Systems

Interaction systems are a component-based formalism, i.e. a system is composed of subsystems called components. Components are put together by some kind of glue-code. Interaction systems are defined in two layers. The first layer, the interaction model, describes the interfaces of the components and the glue-code of a system by connecting the interfaces of the components. The second layer describes the behavior of the components, which is here given in form of labeled transition systems.

* Corresponding author. E-mail: nsemmelr@informatik.uni-mannheim.de

**Definition 1** *Let $K = \{1, \ldots, n\}$ be a finite set of components. For each $i \in K$ let $A_i$ be a set of ports such that $\forall_{i,j \in K}\, i \neq j \Rightarrow A_i \cap A_j = \emptyset$. An interaction model is a tuple $IM := (K, \{A_i\}_{i \in K}, C)$, where $C$ is a set, such that*

*a)* $\forall c \in C : c \subseteq \bigcup_{i \in K} A_i$, *b)* $\forall c \in C \forall i \in K : |c \cap A_i| \leq 1$ *and*
*c)* $\bigcup_{c \in C} c = \bigcup_{i \in K} A_i$.

*The elements of $C$ are called connectors. Let for $c \in C$ and $i \in K$ $i(c) := c \cap A_i$ be the set of ports of $i$ which participate in $c$, i.e. $|i(c)| \leq 1$.*

*Let $T_i = (Q_i, A_i, \rightarrow_i, q_i^0)$ for $i \in K$ be a labeled transition systems with a set of states $Q_i$, a transition relation $\rightarrow_i \subseteq Q_i \times A_i \times Q_i$ and an initial state $q_i^0 \in Q_i$. A transition system $T_i$ for $i \in K$ models the behavior of component $i$. We will write $q_i \xrightarrow{a_i}_i q_i'$ instead of $(q_i, a_i, q_i') \in \rightarrow_i$.*

**Definition 2** *An interaction system is a tuple $Sys := (IM, \{T_i\}_{i \in K})$. The behavior of $Sys$ is given by the transition system*
*$T = (Q_{Sys}, C, \rightarrow, q^0)$ where*

*a)* $Q_{Sys} := \prod_{i \in K} Q_i$ *is the state space,*
*b)* $q^0 = (q_1^0, \ldots, q_n^0) \in Q_{Sys}$ *is the initial state and*
*c)* $\rightarrow \subseteq Q_{Sys} \times C \times Q_{Sys}$ *is the transition relation with $q \xrightarrow{c} q'$ iff for all $i \in K$ $q_i \xrightarrow{i(c)}_i q_i'$ if $i(c) \neq \emptyset$ and $q_i = q_i'$ otherwise.*

**Definition 3** *Let $Sys$ be an interaction system and $T = (Q_{Sys}, C, \rightarrow, q^0)$ the associated global transition system. A global state $q \in Q_{Sys}$ is called reachable iff there is a path that leads from the initial state $q^0$ to $q$ in $T$.*

As mentioned we focus on a structural constraint on interaction systems. More precisely we look at the important class of interaction systems such that the glue-code describes a tree-like communication pattern, i.e. components never form a cycle with respect to their connectors.

A tree-like communication structure induces an important class of component-based systems. Interesting systems belong in this class, e.g. hierarchical systems or networks build by a master-slave operator [8]. For this reasons, this class of component systems has been studied intensely e.g. [2–4, 10].

**Definition 4** *Let $IM = (K, \{A_i\}_{i \in K}, C)$ be an interaction model. The interaction graph $G^* = (K, E)$ of $IM$ is an undirected graph with $\{i, j\} \in E$ iff there is a connector $c \in C$ with $i(c) \neq \emptyset$ and $j(c) \neq \emptyset$.*

*An interaction model $IM$ is called tree-like iff the associated interaction graph $G^*$ is a tree. An interaction system $Sys$ is called tree-like if its associated interaction model is tree-like.*

**Remark** *Note, that a tree-like interaction system with a set $C$ of connectors implies that $\forall_{c \in C} |c| \leq 2$.*

**Example 5** *Consider the dining philosophers problem with n philosophers and n forks. The philosophers respectively the forks are labeled with* $0, \ldots, n-1$. *The philosophers are placed in order around a table such that between philosopher* $i$ *and* $i+1$ *(we assume modulo n arithmetics) fork i is placed. We construct an interaction system such that each philosopher and each fork corresponds to a component. Let* $i \in \{0, \ldots, n\}$ *then philosopher i is modeled by component* $Phil_i$ *with the set of ports* $A_{Phil_i} := \{t\_left_i, t\_right_i, p\_left_i, p\_right_i\}$ *such that the ports are modeling, from left to right: "take left fork", "take right fork", "put left fork back on the table" and "put right fork back on the table". Fork i is modeled by component* $Fork_i$ *with the set of ports* $A_{Fork_i} := \{take_i, release_i\}$.

*The following connectors describe the synchronization between the philosophers and the forks, corresponding to the seating order.*

$$take\_left_i := \{t\_left_i, take_i\} \qquad take\_right_i := \{t\_right_i, take_{i-1}\}$$
$$put\_left_i := \{p\_left_i, release_i\} \qquad put\_right_i = \{p\_right_i, release_{i-1}\}$$

*Consider the problem for* $n = 3$ *philosophers, then the set K of components is given by* $K = \{Phil_0, Phil_1, Phil_2, Fork_0, Fork_1, Fork_2\}$ *and the set C of connectors by* $C := \{take\_left_i, take\_right_i, put\_left_i, put\_right_i | i = 0, \ldots, 2\}$. *The interaction model is given by* $IM = (K, \{A_i\}_{i \in K}, C)$. *The corresponding interaction graph* $G^*$ *for IM is given in Figure 1.*

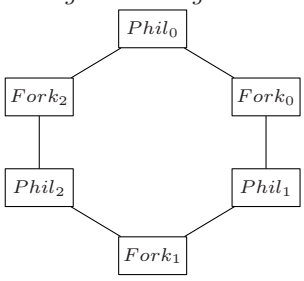



**Fig. 1.** Interaction graph $G^*$ for the interaction model $IM$.

## 3 QBF Reduction to Tree-Like Interaction Systems

We will show that reachability in tree-like interaction systems is PSPACE-complete. The PSPACE-hardness will be proved by a reduction from QBF [6]. PSPACE-hardness of reachability in general interaction systems was shown by a reduction from reachability in 1-safe Petri nets [12]. To show the PSPACE-hardness of reachability in 1-safe Petri nets a reduction from QBF was used [5].

### 3.1 Reduction

Reachability in tree-like interaction systems is in PSPACE. Given a tree-like interaction system and a global state $q$ one can guess a sequence of connectors (because PSPACE=NPSPACE) and check in linear space if it leads from the initial state $q^0$ to $q$. To prove the PSPACE-hardness we present a reduction from Quantified Boolean Formulas (QBF) to the reachability problem in tree-like interaction systems.

**QBF** An instance of QBF [6] is given as a well-formed quantified Boolean formula where its variables $x_1, \ldots, x_n$ are all bound and distinct. Without loss of generality we look at QBF instances over the grammar

$$P ::= x | \neg P | P \wedge P | \exists x.P.$$

In the following we will assume that a QBF formula is built over this grammar. Let $H$ be a QBF then the question is if $H$ is true. The language TQBF is defined as the set of true QBF instances and is well known to be PSPACE-complete.

**RIST** Let $IST$ be the class of tree-like interaction systems. For $Sys \in IST$ let $Reach(Sys) \subseteq Q_{Sys}$ be the set of reachable states. Let

$$RIST := \bigcup_{Sys \in IST} (\{Sys\} \times Q_{Sys}) .$$

For $(Sys, q) \in RIST$ we want to decide if $q$ is reachable in $Sys$. Let $TRIST \subseteq RIST$ be the set of true $RIST$ instances, i.e.

$$TRIST = \bigcup_{Sys \in IST} (\{Sys\} \times Reach(Sys)) .$$

In the following we will introduce for a QBF $H$ a tree-like interaction system $Sys_H$ and a global state $q^t$ such that

i) $H \in TQBF \Leftrightarrow (Sys_H, q^t) \in TRIST$ and
ii) the size of $Sys_H$ is polynomial in the size of $H$.

The idea for the construction of $Sys_H$ can be sketched as follows: the interaction system basically simulates the evaluation of the formula $H$ based on the syntax tree of $H$. The subformulas of $H$ are the components of the system and the interaction model the propagation of truth values from the leaves of the syntax tree, i.e. the variables upwards. We now describe in detail how $Sys_H$ is constructed:

**Components** Let $H$ be a QBF with variables $x_1, \ldots, x_n$ and $K_2 = \{x_i' | x_i \text{ is a variable in } H\}$. The set of components $K_2$ is needed to avoid cycles in the interaction graph. Generally, there may be several occurrences of a variable $x_i$ in $H$. Let $x_i$ occur $k_i$ times for $i = 1, \ldots, n$ as a subformula in $H$, then we assume that the $j$th occurrence of variable $x_i$ is renamed in $H$ as $x_i^j$ for $j \in \{1, \ldots, k_i\}$.

Let $K_H = K_1 \cup K_2 \cup \{H'\}$ be a set of components, such that $K_1 = \{P | P \text{ is a subformula of } H\}$. The component $H'$ is an auxiliary component which simplifies the definition of the behavior of the components in $K_1$.

Given a truth assignment to the variables, subformulas are assigned true or false. Therefore, when we mention an assignment to a component in $K_1 \cup K_2$ we refer to the assignment of the subformula that is modeled by this component.

In the following we will give the port sets of the components. Many ports, in different components, serve the same purpose and only differ in their subscripts. Once, such a port is introduced it will not be explicitly explained again.

**Port sets of components modeling variables** For $i = 1, \ldots, n$ and $j = 1, \ldots, k_i$ the component $P = x_i^j \in K_1$ represents the $j$th occurrence of variable $x_i$ in $H$. The set $A_P$ of ports is given by

$$A_P := \{\mathfrak{a}_P, t_P, f_P, r_P t\} \cup \{r_P x_l t, r_P x_l f | l = 1, \ldots, n\}.$$

- $\mathfrak{a}_P$ abbreviates "activate $P$" and starts the evaluation of $P$.
- $t_P$ respectively $f_P$ confirm that currently true respectively false is assigned to P.
- $r_P x_l t$ abbreviates "$P$ receives instruction to set $x_l$ true". If $l = i$ then true is assigned to $P$. For $i \neq l$ $r_P x_l t$ has no effect on $P$. The same applies to $r_P x_l f$ setting $x_l$ to false.
- $r_P t$ has the function to assign true to $P$.

**Port sets for negated formulas** A component modeling a negation, i.e. a subformula of the form $P = \neg P_1$ has the following set of ports $A_P$

$$\begin{aligned} A_P := \{&\mathfrak{e}_P^1, \mathfrak{a}_P, sub_P^1 t, sub_P^1 f, t_P, f_P, r_P t, s_P^1 t\} \cup \\ &\{r_P x_l t, r_P x_l f, s_P^1 x_l t, s_P^1 x_l f | l = 1, \ldots, n\}. \end{aligned}$$

- $\mathfrak{e}_P^1$ abbreviates "evaluate the first subformula of $P$" and evaluates the subformula $P_1$.
- $sub_P^1 t$ (abbreviates "subformula 1 is true") respectively $sub_P^1 f$ affirm that $P_1$ was evaluated true respectively false.
- According to the structure of a negation $f_P$ (abbreviates "$P = \neg P_1$ is false") is enabled if $P_1$ was evaluated true. Conversely $t_P$ is enabled if $P$ was evaluated false.
- Like above $r_P x_l t$ models that $P$ receives the instruction to set $x_l$ true. On the other hand $s_P^1 x_l t$ ("set $x_l$ true in the first subformula of $P$") models that $P$ itself sends the instruction to set $x_l$ to true to $P_1$. The same applies to $s_P^1 x_l f$ if $x_l$ needs to be set to false.
- $s_P^1 t$ has the function to set the truth assignment of $P$'s subformula $P_1$ to true.

**Port sets for conjunctions** The component that models a conjunction, i.e. a subformula of the form $P = P_1 \wedge P_2$ has the set of ports

$$\begin{aligned} A_P := \{&\mathfrak{a}_P, \mathfrak{e}_P^1, \mathfrak{e}_P^2, sub_P^1 t, sub_P^1 f, sub_P^2 t, sub_P^2 f, t_P, f_P, r_P t, s_P^1 t, s_P^2 t\} \cup \\ &\{r_P x_l t, r_P x_l f, s_P^1 x_l t, s_P^1 x_l f, s_P^2 x_l t, s_P^2 x_l f | l = 1, \ldots, n\}. \end{aligned}$$

This is the only formula that has two direct subformulas. $P = P_1 \wedge P_2$ needs to evaluate $P_1$ and $P_2$, therefore there are ports $\mathfrak{e}_P^1$ and $\mathfrak{e}_P^2$. Similarly there are $sub_P^1 t$, $sub_P^1 f$, $sub_P^2 t$, $sub_P^2 f$ for actually receiving the truth values of $P_1$ and $P_2$. Likewise, $s_P^1 x_l t$ and $s_P^2 x_l t$ model that $P$ needs to set $x_l$ to true in its first and second subformula and respectively $s_P^1 x_l f$ and $s_P^2 x_l f$ for false.

**Port sets for existentially quantified formulas and associated component $x'_i$** In the interaction system $Sys_H$ a component for a subformula of the form $P = \exists x_i.P_1$ with $i = 1, \ldots, n$ needs to have access to the current truth assignment of the variable $x_i$. For this purpose the set of components $K_2$ was introduced. Let $x_i$ be the variable that is quantified by the subformula $P = \exists x_i.P_1$. The component $x'_i$ models the truth assignment of $x_i$. The set of ports $A_{x'_i}$ is given by

$$A_{x'_i} := \{rx_it, rx_if, t_{x_i}, f_{x_i}\}.$$

$t_{x_i}$ respectively $f_{x_i}$ affirm that in the current state of $x'_i$ is true respectively false. $rx_it$ assigns $x'_i$ true. Analogously $rx_if$ switches the assignment to false.

The port set $A_P$ for $P = \exists x_i.P_1$ is given by

$$A_P := \{\mathfrak{a}_P, \mathfrak{e}^1_P, sub^1_P t, sub^1_P f, t_P, f_P, x_it, x_if, sx_it, sx_if, r_P t, s^1_P t\} \cup \{r_P x_l t, r_P x_l f, s^1_P x_l t, s^1_P x_l f | l = 1, \ldots, n\}.$$

$\mathfrak{a}_P$, $\mathfrak{e}^1_P$, $sub^1_P t$, $sub^1_P f$, $t_P$ and $f_P$ act corresponding to the corresponding ports of the other components specified above. $x_it$ confirms that true is assigned to $x_i$ and $sx_it$ sets $x_i$ to true if the current assignment is false. On the other hand $x_if$ confirms that false is assigned to $x_i$ and $sx_if$ assigns false to $x_i$ if thats is not the case.

**Port set for the auxiliary component $H'$** Given the syntax tree for $H$, whose root is labeled $H$, $H'$ can be interpreted as a direct dummy predecessor formula of $H$ without any logical operator. The set of ports $A_{H'}$ is given by

$$A_{H'} := \{\mathfrak{e}^1_{H'}, sub^1_{H'} t, sub^1_{H'} f, s^1_{H'} t, end_{H'}\}.$$

All ports but $end_{H'}$ act exactly as the ports described above. It will be shown that the formula $H$ is in TQBF iff the component associated with $H$ is evaluated true, i.e. $sub^1_{H'} t$ can interact eventually. When the evaluation of the QBF $H$ has been simulated, i.e. $H'$ reached a state that represent the fact that $H$ was evaluated true or false, then the port $end_{H'}$ becomes enabled. This only assures that the behavior of $H'$ does not deadlock.

**Connectors** We will now define a set $C$ of connectors. Let $P \in K_1 \cup \{H'\}$ be a subformula, not an occurrence of a variable. $P$ can have one direct subformula which is $P_1$ or two direct subformulas $P_1$ and $P_2$. If $P$ needs the truth value of $P_k$, $k \in \{1, 2\}$, to be evaluated then the evaluation in $P_k$ needs to be activated. This is realized by the synchronization of $\mathfrak{e}^k_P$ and $\mathfrak{a}_{P_k}$. Furthermore $P$ can ask $P_k$ for its current truth value. These interactions are realized by

$$eval\_P \to P_k := \{\mathfrak{e}^k_P, \mathfrak{a}_{P_k}\} \qquad P\_ask\_P_k\_true := \{sub^k_P t, t_{P_k}\}$$
$$P\_ask\_P_k\_false := \{sub^k_P f, f_{P_k}\}$$

for $k \in \{1, 2\}$. These connectors already connect all components in $K_1 \cup \{H'\}$ and result in an interaction graph that is related to the syntax tree of the QBF $H$.

If $P$ needs all occurrences of variable $x_i$ to be set to true or false a direct interaction with the components that model these variables would lead to a cycle in the associated interaction graph. Therefore, $P$ passes this information to its subformulas, i.e. $s^k_P x_i t$ in $P$ has to synchronize with $r_{P_k} x_i t$ in $P_k$ where $P_k$ is a direct subformula of $P$. Let $i \in \{1, \ldots, n\}$. The following connectors, for $k \in \{1, 2\}$, realize the synchronizations needed to propagate the information to switch a variable.

$$set\_x_i\_true\_P \to P_k := \{s^k_P x_i t, r_{P_k} x_i t\}$$
$$set\_x_i\_false\_P \to P_k := \{s^k_P x_i f, r_{P_k} x_i f\}$$

If the QBF $H$ is true, we need all components to be in one fixed state – this will be a state that models the assignment true. In fact, the component $H'$ will observe if $H$ is true and reach a fixed state. To assure that all components can reach a fixed state, a similar technique like above is used. A component can set the truth assignment of the components that represent its subformulas to true by the following connector for $k \in \{1, 2\}$.

$$set\_P_k\_true\_P \to P_k := \{s^k_P t, r_{P_k} t\}$$

Consider a subformula of the form $P = \exists x_i.P_1 \in K_1$ and the associated component $x'_i \in K_2$. The component representing $P$ can assign $x'_i$ the truth value true or false and can ask $x'_i$ whether the current truth assignment is true or false. This is realized by

$$set\_x'_i\_true := \{s x_i t, r x_i t\} \qquad ask\_true_{x'_i} := \{x_i t, t_{x'_i}\}$$
$$set\_x'_i\_false := \{s x_i f, r x_i f\} \qquad ask\_false_{x'_i} := \{x_i f, f_{x'_i}\}$$

IF $H'$ reaches a state that indicates that $H$ was evaluated true or false, i.e. the simulation of the evaluation of $H$ is finished then the unary connector $evaluated := \{end_{H'}\}$ becomes enabled.

Let $C$ be the set of connectors given by

$\{eval\_P \to P_k, P\_ask\_P_k\_true, P\_ask\_P_k\_false | P \in K_1 \cup \{H'\}$ with succ. $P_k\} \cup$
$\{set\_x'_i\_true, set\_x'_i\_false, ask\_true_{x'_i}, ask\_false_{x'_i} | x'_i \in K_2\} \cup$
$\{set\_P_k\_true\_P \to P_k | P \in K_1 \cup \{H'\}$ with succ. $P_k\} \cup$
$\{set_x_i_true_P \to P_k, set_x_i_false_P \to P_k | P \in K_1$with succ. $P_k, i \in \{1, \ldots, n\}\} \cup$
$\{evaluated\}$.

So far we have the interaction model $IM_H := (K_H, \{A_P\}_{P \in K_H}, C)$. This way any QBF formula $H$ over the grammar, given above, can be mapped to an interaction model $IM_H$.

**Remark** *The interaction graph $G^*_H$, associated to $IM_H$, is a tree, as it is constructed along the syntax tree and augmented with the components $H'$ and $x'_i$ for $i = 1, \ldots, n$ without forming cycles.*

**Example 6** *Consider the formula* $H = \neg\exists x_1.(x_1 \wedge \neg x_1)$*. The associated interaction graph* $G^*_H$ *of* $IM_H$ *is given in Figure 2 where components with highlighted frames denote components that do not model subformulas of* $H$*.*

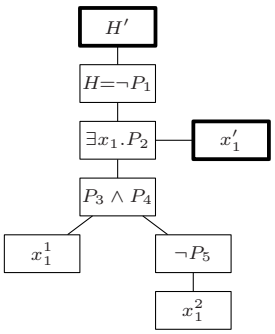
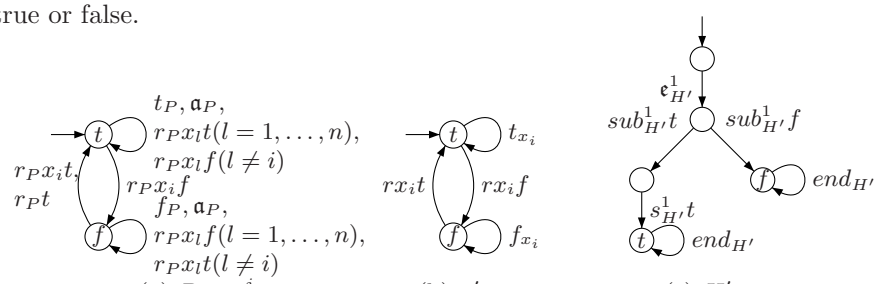



**Fig. 2.** Interaction graph $G^*_H$ of $Sys_H$.

**Local Behavior** The local behavior of the components is given by labeled transition systems. Every system has one state labeled $t$ and one labeled $f$. These states model the fact, that either true respectively false was assigned to this component or it was evaluated true respectively false. The initial state will be denoted by an ingoing arrow.

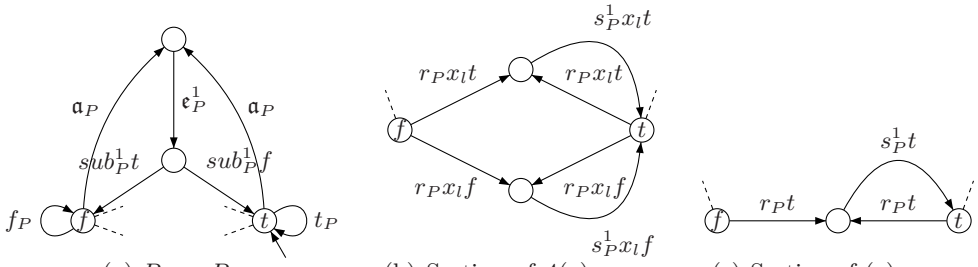
Figure 3(a) depicts the transition system of the component modeling the $j$th occurrence of variable $x_i$. Figure 3(b) gives the local behavior of a component $x'_i \in K_2$. The behavior of $H'$ is given in 3(c). The transition systems for a variable $x^j_i$ and a $x'_i \in K_2$ are self-explanatory. If in $T_{H'}$ the port $\mathfrak{e}^1_{H'}$ is performed, i.e. component $H$ needs to be evaluated, then $T_{H'}$ waits to perform either $sub^1_{H'}t$ or $sub^1_{H'}f$. This ports can only be performed if $T_H$ reaches its state labeled $t$ respectively $f$. It will be shown that this indicates whether the associated QBF is true or false.




(a) $P = x^j_i$ (b) $x'_i$ (c) $H'$

**Fig. 3.** Transition systems $T_{x^j_i}$ for a component $x^j_i$ (a), $T_{x'_i}$ for $x'_i$ (b) and $T_{H'}$ for the component $H'$ (c).

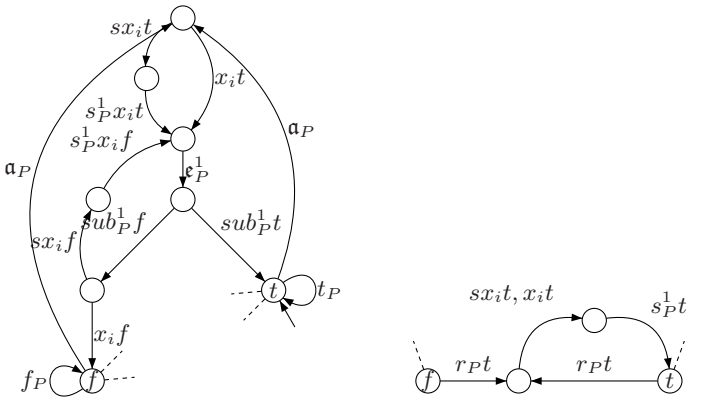
In Figure 4 the transition system for a component of the form $P = \neg P_1$ is pictured. Note, that for better readability, the transition system in Figure 4(a) is not completely displayed. In system 4(a) the transitions and states pictured in Figure 4(b) and 4(c) have to be included between the states labeled $t$ and $f$ for $l = 1, \ldots, n$.




(a) $P = \neg P_1$ (b) Section of 4(a) and 5(a), for $l \in \{1, \ldots, n\}$ (c) Section of (a)

**Fig. 4.** Main section of the transition systems $T_{\neg P_1}$ (a), part of $T_{\neg P_1}$ for $l \in \{1, \ldots, n\}$ (b), part of $T_{\neg P_1}$ (c).

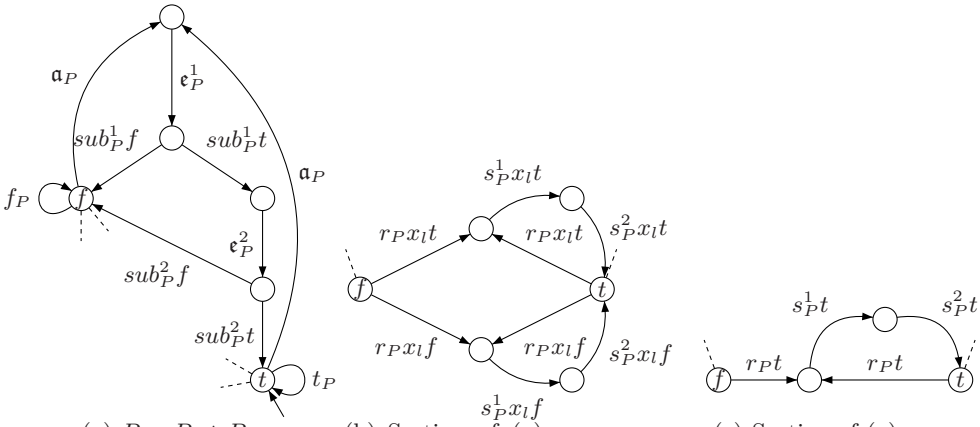
In Figure 5 the transition system for a component of the form $P = \exists x_i.P_1$ is pictured. For better readability, the transition system in Figure 5(a) is not completely displayed. In system 5(a) the transitions and states pictured in Figure 4(b) and 5(b) have to be included between the states labeled $t$ and $f$ for $l = 1, \ldots, n$.




(a) $P = \exists x_i.P_1$ (b) Section of 5(a)

**Fig. 5.** Main sections of the transition system $T_{\exists x_i.P_1}$ (a), part of $T_{\exists x_i.P_1}$ (b).

In Figure 6 the transition system for a component of the form $P = P_1 \wedge P_2$ is pictured. Note, that the transition system in Figure 6(a) is not completely displayed. The transitions and states pictured in Figure 6(b) and 6(c) have to be included between the states labeled $t$ and $f$ for $l = 1, \ldots, n$.




(a) $P = P_1 \wedge P_2$ (b) Section of (a), for $l \in \{1, \ldots, n\}$ (c) Section of (a)

**Fig. 6.** Transition system $T_{P_1 \wedge P_2}$.

The resulting interaction system is denoted $Sys_H := (IM_H, \{T_P\}_{P \in K_H})$.

**Theorem 7** *Let $H$ be a QBF over the grammar $P ::= x|\neg P|P \wedge P|\exists x.P$ and $Sys_H$ the associated interaction system obtained from the reduction. Let $q^t$ be the global state in which all components are in their state labeled $t$, then*

$$H \in TQBF \Leftrightarrow (Sys_H, q^t) \in TRIST.$$

The proof of Theorem 7 is presented in the Appendix.

## 4 QBF Reduction to Progress in tree-like Interaction Systems

By minor modification of the reduction given above it is possible to show the PSPACE-completeness of the progress property in tree-like interaction systems. At first we give some definitions to introduce progress in interaction systems and then give an overview, why it is PSPACE-complete to decide this property in tree-like interaction systems. In general interaction systems progress is PSPACE-complete [12], i.e. progress in tree-like interaction systems is in PSPACE.

**Definition 8** *Let $Sys$ be an interaction system and $T = (Q_{Sys}, C, \rightarrow, q^0)$ the associated global transition system. A global state $q \in Q_{Sys}$ is called a deadlock if no connector is enabled in $q$, i.e. there is no $c \in C$ and $q' \in Q_{Sys}$ such that $q \xrightarrow{c} q'$. A system $Sys$ is free of deadlocks if there is no reachable state $q \in Q_{Sys}$ such that $q$ is a deadlock.*

**Definition 9** *Let $Sys$ be a deadlock-free interaction system. A run of $Sys$ is an infinite sequence $\sigma$*

$$q^0 \xrightarrow{c_1} q^1 \xrightarrow{c_2} q^2 \ldots,$$

*with $q^l \in Q_{Sys}$ and $c_l \in C$ for $l \geq 1$.*

**Definition 10** *Let $Sys$ be a deadlock-free interaction system with components $K$. $k \in K$ may progress in $Sys$ if for every run $\sigma$ $k$ participates infinitely often in $\sigma$.*

We modify $Sys_H$ as follows:

We introduce an additional component called *pro* with the set of ports $A_{pro} := \{t_{pro}\}$ and the behavior given by the transition system $T_{pro}$ in Figure 7. The idea is to embed *pro* in $Sys_H$ such that $t_{pro}$ will participate infinitely often in every run $\sigma$ iff $H$ is true.

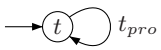



**Fig. 7.** Transition system $T_{pro}$ for the component *pro*.

In addition we modify the component $H'$ as follows. The set of ports $A_{H'}$ of the component $H'$ is now given by

$$A_{H'} := \{\mathfrak{e}^1_{H'}, sub^1_{H'}t, sub^1_{H'}f, s^1_{H'}t, end\_true_{H'}, end\_false_{H'}\},$$

i.e. $end_{H'}$ is removed and the ports $end\_true_{H'}$ and $end\_false_{H'}$ are added. The modified behavior of $H'$ is given by the transition system $T_{H'}$ in Figure 4.

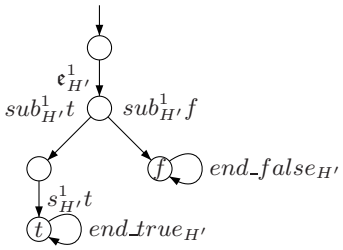



**Fig. 8.** Modified transition system $T_{H'}$ for the component $H'$.

In addition, the connector *evaluated* is removed von the set $C$ of connectors and the two following connectors are added.

$$evaluated\_true := \{end\_true_{H'}, t_{pro}\},$$
$$evaluated\_false := \{end\_false_{H'}\}.$$

It is easy to see that the connector *evaluated_true* is the only connector that is enabled if the state $q^t$ is reached. In this case, *evaluated_true* will perform infinitely often, i.e. the component *pro* will participate infinitely often. Therefore the component *pro* may progress iff $H$ is true.

## 5 Related Work

Apart from classical techniques as partial order reduction or abstraction for handling the complexity issue of reachability, approaches have been investigated that establish sufficient conditions that can be tested in polynomial time and ensure the desired property. For general component systems this is pursued e.g. in [1, 9, 11, 13]. For tree-like component systems [2–4, 10] have followed this approach and in particular established conditions that ensure deadlock-freedom.

## References


1. Paul Attie and Hana Chockler. Efficiently Verifiable Conditions for Deadlock-Freedom of Large Concurrent Programs. In *Proceedings of VMCAI'05*, LNCS 3385, pages 465–481, 2005.
2. H. Baumeister, F. Hacklinger, R. Hennicker, A. Knapp, and M. Wirsing. A Component Model for Architectural Programming. In *Proceedings of FACS'05*, volume 160 of *ENTCS*, pages 75–96. Elsevier, 2006.
3. Marco Bernardo, Paolo Ciancarini, and Lorenzo Donatiello. Architecting Families of Software Systems with Process Algebras. *ACM Trans. on Software Engineering and Methodology*, 11:386 – 426, October 2002.
4. Stephen D. Brookes and A. W. Roscoe. Deadlock Analysis in Networks of Communicating Processes. *Distributed Computing*, 4:209–230, 1991.
5. Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity Results for 1-safe Nets. In *Theoretical Computer Science*, pages 326–337. Springer-Verlag, 1995.
6. M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, January 1979.
7. Gregor Gössler and Joseph Sifakis. Composition for Component-Based Modeling. *Sci. Comput. Program.*, 55(1-3):161–183, 2005.
8. Charles A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International series in computer science. Prentice-Hall International, Englewood Cliffs, NJ [u.a.], 1985.
9. M. Majster-Cederbaum, M. Martens, and C. Minnameier. A Polynomial-Time Checkable Sufficient Condition for Deadlock-Freedom of Component-Based Systems. *Lecture Notes in Computer Science*, 4362/2007:888–899, 2007.
10. Mila Majster-Cederbaum and Moritz Martens. Compositional Analysis of Deadlock-Freedom for Tree-Like Component Architectures. In *EMSOFT '08: Proceedings of the 7th ACM international conference on Embedded software*, pages 199–206, New York, NY, USA, 2008. ACM.
11. Mila Majster-Cederbaum, Moritz Martens, and Christoph Minnameier. Liveness in Interaction Systems. *Electron. Notes Theor. Comput. Sci.*, 215:57–74, 2008.
12. Mila E. Majster-Cederbaum and Christoph Minnameier. Everything Is PSPACE-Complete in Interaction Systems. In *ICTAC*, pages 216–227, 2008.
13. Christoph Minnameier and Mila Majster-Cederbaum. Cross-Checking – Enhanced Over-Approximation of the Reachable Global State Space of Component-Based Systems, 2009. submitted for puplication.

# A Appendix - Correctness

Before we prove Theorem 7, we need some preliminaries.

**Recursive Algorithm** There is a straightforward, recursive algorithm called *eval* to determine whether a QBF $P$ given over the grammar above is in TQBF.

1. $eval(P)$
2. $\quad if(P = x)$
3. $\quad\quad$ return $value(x)$
4. $\quad if(P = \neg P')$
5. $\quad\quad$ return $\neg eval(P')$
6. $\quad if(P = P' \wedge P'')$
7. $\quad\quad$ return $eval(P') \wedge eval(P'')$
8. $\quad //\ P = \exists x.P'$ is the only remaining possibility
9. $\quad$ return $eval(P'_{x=true}) \vee eval(P'_{x=false})$

In line 9 $P'_{x=true}$ denotes the subformula $P'$ with $true$ assigned to the variable $x$. In line 3 $value(x)$ returns the truth value that is assigned to $x$. This is possible because every variable $x$ in $H$ is bound by an existential quantifier and therefore a truth value is assigned in line 9. Obviously, $H \in TQBF \Leftrightarrow eval(H) = true$.

We assume, that in line 7 and 9 the conjunction respectively the disjunction is called in sequence from left to right. In addition, we assume, that $eval(P'')$ is not called in line 7 if $eval(P')$ is evaluated $false$ and $eval(P'_{x=false})$ is not called in line 9 if $eval(P'_{x=true})$ is evaluated $true$. These assumptions imply a deterministic, unique execution of $eval(H)$.

The execution of $eval(H)$ for a QBF $H$ can be described uniquely by a sequence over

- "$call\_eval(P)$": subformula $P$ is called by $eval$
- "$eval(P) = true$": subformula $P$ was evaluated $true$ by $eval$
- "$eval(P) = false$": subformula $P$ was evaluated $false$ by $eval$

For a QBF $H$ let $Seq_H$ be this sequence and $Seq_H(i)$ the $i$th word in $Seq_H$ for $i = 1, \ldots, length(Seq_H)$, where $length(Seq_H)$ is the number of words in $Seq_H$. It is clear, that $H \in TQBF$ iff the last entry of $Seq_H$ is "$eval(H) = true$" and "$eval(H) = false$" otherwise.

**Example**: Consider the QBF $H = \neg\exists x_1.(x_1 \wedge \neg x_1)$ with its subformulas abbreviated as in Figure 2, then $Seq_H$ is given by

- $call\_eval(H)$
- $call\_eval(P_1)$
- $call\_eval(P_2)$ (true is assigned to $x_1$)
- $call\_eval(P_3)$
- $eval(P_3) = true$
- $call\_eval(P_4)$
- $call\_eval(P_5)$
- $eval(P_5) = true$
- $eval(P_4) = false$
- $eval(P_2) = false$
- $call\_eval(P_2)$ (false is assigned to $x_1$)
- $call\_eval(P_3)$
- $eval(P_3) = false$
- $eval(P_2) = false$
- $eval(P_1) = false$
- $eval(H) = true$

**Mapping the words of $Seq_H$ to $C$** Let $H \in QBF$ and $Sys_H$ be the associated tree-like interaction system. We treat the associated interaction graph $G^*$ as a rooted tree with component $H'$ as the root. In these terms, if we speak of a successor, a predecessor or a subtree spanned by a component, we refer to components with respect to $G^*$. Let $C' \subseteq C$ be the subset of connectors given by:

$$\{eval\_P \to P_k, P\_ask\_P_k\_true, P\_ask\_P_k\_false | P \in K_1 \cup \{H'\} \text{ with succ. } P_k\}$$

and $S$ the set of words that can possibly occur in $Seq_H$, given by

$$\{\text{“}call\_eval(P)\text{”}, \text{“}eval(P) = true\text{”}, \text{“}eval(P) = false\text{”} | P \text{ is a subformula of } H\}.$$

Define the function $f : S \to C'$ by

- $f(\text{“}call\_eval(P)\text{”}) = eval\_P' \to P$,
- $f(\text{“}eval(P) = true\text{”}) = P'\_ask\_P\_true$ and
- $f(\text{“}eval(P) = false\text{”}) = P'\_ask\_P\_false$.

where $P'$ is the predecessor of $P$ and $P' = H'$ if $P = H$.

**Lemma 11** *Let $\tilde{\sigma}$ be a trace of $Sys_H$, such that $\tilde{\sigma}$ is infinite or ends in a deadlock. Let $\sigma$ be the sequence obtained by removing the connectors in $C \setminus C'$ and let $\sigma(i)$ be the $i$th connector in $\sigma$ for $i = 1, \ldots, length(\sigma)$, where $length(\sigma)$ is the length of $\sigma$. Then $length(\sigma) = length(Seq_H)$ and*

$$\bigvee_{i=1,\ldots,length(\sigma)} f(Seq_H(i)) = \sigma(i).$$

Before we prove Lemma 11 by induction, we need some observations which follow from invariants of algorithm *eval* that are easy to show. In the following we will refer repeatedly to the structure of the transition systems given in Figure 3, 4 and 5 and the connectors given on page 6. We assume the induction hypothesis to be true.

**Observation 12** *Consider $\sigma(i)$ to be performed and let $Seq_H(i{+}1) =$ “$eval(P) = true$” where $P'$ is the predecessor of $P$, then $P'$ waits to perform $P'\_ask\_P\_true$.*

*The same applies for $Seq_H(i+1) =$ “$eval(P) = false$” where $P'$ waits to perform $P'\_ask\_P\_false$.*

***Proof:***
*There is $1 \leq j \leq i$ with $Seq_H(j) =$ “$call\_eval(P)$”, i.e. if subformula $P$ is evaluated true then it is assured that $P$ was called previously. Let $j$ be maximal for this property. For $j + 1 \leq k \leq i$ $Seq_H(k) \notin \{$“$call\_eval(P)$”, “$eval(P) = true$”, “$eval(P) = false$”$\}$, i.e. $P$ is not involved in between. For $\sigma$ follows that $\sigma(j) = f(\text{“}call\_eval(P)\text{”}) = eval\_P' \to P$, i.e. $P'$ reached a state in which it waits for $P'\_ask\_P\_true$ or $P'\_ask\_P\_false$. Since these connectors were not performed for $j+1 \leq k \leq i$, it is assured, that after $\sigma(i)$ has been performed, $P'$ still waits to perform $P'\_ask\_P\_true$ or $P'\_ask\_P\_false$.*

□

**Observation 13** *Consider $\sigma(i)$ to be performed and let $Seq_H(i+1) =$ "call_eval(P)". Let component $P'$ be the predecessor of component $P$, then $P$ will eventually reach a state in which it waits to perform eval_$P' \rightarrow P = f(Seq_H(i+1))$.*

***Proof:***
*There are six cases for $P$:*

1. *$P$ waits to perform eval_$P' \rightarrow P$, i.e. $P$ is in a state labeled $t$ or $f$, then eval_$P' \rightarrow P$ is enabled.*
2. *$P$ waits to perform set_$\tilde{P}$_true_$P \rightarrow \tilde{P}$ for a successor $\tilde{P}$ of $P$. It is easy to see that this is only possible if $f($"$eval(H) = true$"$) = H'$_ask_H_true performed which is not the case.*
3. *$P$ waits to perform eval_$P \rightarrow \tilde{P}$. For reaching this state either eval_$P' \rightarrow P$, $P$_ask_$\tilde{P}$_$false$ (if $P = \exists.x_l\tilde{P}$) or $P$_ask_$\bar{P}'$_$false$ (if $P = \bar{P} \wedge \tilde{P}$) was performed. Let this be the case for $f(Seq_H(j))$ with $j < i$ ($j$ maximal). Due to the structure of $Seq_H$, $f(Seq_H(j+1)) =$ eval_$P \rightarrow \tilde{P}$ would be the next connector to be performed, i.e. $P$ can not stay in a state waiting to perform eval_$P \rightarrow \tilde{P}$.*
4. *$P$ waits to perform $P$_ask_$\tilde{P}$_true. Analogously to case 3, this cannot happen.*
5. *$P = \exists.x_l\tilde{P}$ and waits to perform set_$x_l'$_$true/false$ or ask_$true/false_{x_l'}$ for $l \in \{1, \ldots, n\}$. This connector would always be performed, and subsequently $P$ waits to perform eval_$P \rightarrow \tilde{P}$, which is not possible due to case 3.*
6. *$P$ waits to perform set_$x_l$_true_$P \rightarrow \tilde{P}$ respectively set_$x_l$_$false$_$P \rightarrow \tilde{P}$ for $l \in \{1, \ldots, n\}$. If $\tilde{P}$ models a variable then set_$x_l$_true_$P \rightarrow \tilde{P}$ respectively set_$x_l$_$false$_$P \rightarrow \tilde{P}$ is enabled by $\tilde{P}$ and can perform. After this, $f(Seq_H(i+1)) =$ eval_$P' \rightarrow P$ becomes enabled by $P$. If $\tilde{P}$ does not model a variable then analogously (to case 1-5) either $\tilde{P}$ enables set_$x_l$_true_$P \rightarrow \tilde{P}$ respectively set_$x_l$_$false$_$P \rightarrow \tilde{P}$ or $\tilde{P}$ waits to perform set_$x_r$_true_$\tilde{P} \rightarrow \bar{P}$ respectively set_$x_r$_$false$_$\tilde{P} \rightarrow \bar{P}$ ($r \in \{1, \ldots, n\}$) for a successor $\bar{P}$ of $\tilde{P}$. By induction follows, that this connector will perform eventually. Therefore, $f(Seq_H(i+1)) =$ eval_$P' \rightarrow P$ will eventually become enabled as well.*

□

**Observation 14** *Let $Seq_H(i+1) =$ "$eval(P) = true$" such that $P = x_l^r$ for $l \in \{1, \ldots, n\}$ and $r \in \{1, \ldots, k_l\}$, then it is assured that $P$ waits to perform $P'$_ask_$P$_true after $\sigma(i)$ is performed. The same applies for $Seq_H(i+1) =$ "$eval(P) = false$" with $\sigma(i+1) = P'$_ask_$P$_$false$.*

***Proof:***
*Let $Seq_H(i+1) =$ "$eval(P) = true$" (resp. $Seq_H(i+1) =$ "$eval(P) = false$"), then $Seq_H(i) =$ "call_eval(P)" and there is $Q' = \exists x_l.Q$ and $j < i$ such that $Seq_H(j) =$ "call_eval(Q)", i.e. if algorithm eval calls a variable recursively then it is assured that beforehand a subformula was called that quantifies this variable. Let $j$ be maximal for this property. There are two cases for $j-1$:*

*a)* $Seq_H(j-1) =$ "$call\_eval(Q')$", *i.e.* $x_l$ *is set to true in the subsequent call of* $eval(Q)$ *(see algorithm eval). After* $\sigma(j-1) = f(Seq_H(j-1))$ *was performed, either*

*a.1)* $set\_x'_l\_true$ *or*

*a.2)* $ask\_true_{x'_l}$ *becomes enabled.*

*b)* $Seq_H(j-1) =$ "$eval(Q) = false$", *i.e.* $Q$ *was evaluated false and is called by eval again with* $x_l$ *set to false. After* $\sigma(j-1) = f(Seq_H(j-1))$ *was performed, either*

*b.1)* $set\_x'_l\_false$ *or*

*b.2)* $ask\_false_{x'_l}$ *becomes enabled. This is not possible, because then there is no way* $\sigma(j) = call\_Q' \rightarrow Q = f(Seq_H(j))$.

*Consider Case a.1) (resp. b.1)). Let, after* $\sigma(j-1) = f(Seq_H(j-1))$ *was performed,* $set\_x'_l\_true$ *(resp.* $set\_x'_l\_false$*) be enabled and perform. This means that* $Q'$ *waits to perform* $set\_x_l\_true\_Q' \rightarrow Q$ *(resp.* $set\_x_l\_false\_Q' \rightarrow Q$*). Analogously to 13, this connector eventually will become enabled. If* $set\_x_l\_true\_Q' \rightarrow Q$ *(resp.* $set\_x_l\_false\_Q' \rightarrow Q$*) is performed it is clear that* $eval\_Q' \rightarrow Q = \sigma(j) = f(Seq_H(j))$ *becomes enabled. Analogously, for each component* $\tilde{Q}$ *and its predecessor* $\tilde{Q}'$*,* $set\_x_l\_true\_\tilde{Q}' \rightarrow \tilde{Q}$ *(resp.* $set\_x_l\_false\_\tilde{Q}' \rightarrow \tilde{Q}$*) has to be performed before* $eval\_\tilde{Q}' \rightarrow \tilde{Q}$ *becomes enabled. This is until* $\tilde{Q}$ *models a variable. If* $\tilde{Q}$ *models an occurrence of* $x_l$*, then true (resp. false) is assigned to* $\tilde{Q}$*, else, there is no effect on the current state of* $\tilde{Q}$*. Therefore it is assured that* $P$ *waits to perform* $P'\_ask\_P\_true$ *(resp.* $P'\_ask\_P\_false$*) after* $\sigma(i)$ *is performed.*

*Consider Case a.2), i.e.* $ask\_true_{x'_l}$ *is enabled after* $\sigma(j-1)$ *performed. Then the component* $x'_l$ *is in the state labeled* $t$*. This means, the last connector involving* $x'_l$ *can not be* $set\_x'_l\_false$ *or* $ask\_false_{x'_l}$*. There are three cases*

1. $x'_l$ *was never involved since* $\sigma(j-1)$ *is performed. Due to the fact that all components modeling occurrences of variables start in their state labeled* $t$*, it is easy to see that it is not possible that any of these components could reach the state labeled* $f$*. Therefore these components are still in the state labeled* $t$ *when* $P'$ *waits to perform* $P'\_ask\_P\_true$*.*
2. *The last connector involving* $x'_l$ *was* $set\_x'_l\_true$*. With Case a.1) follows that all components that model occurrences of* $x_l$ *were set in their respective state labeled* $t$*. As there was no connector involving* $x'_l$ *since* $set\_x'_l\_true$ *performed, it is assured that this components are still in the state labeled* $t$ *when* $P'$ *waits to perform* $P'\_ask\_P\_true$*.*
3. *The last connector involving* $x'_l$ *was* $ask\_true_{x'_l}$*. This case is easily reducible to the last two cases. Therefore it is assured that all components modeling occurrences of* $x_l$ *are still in the state labeled* $t$ *when* $P'$ *waits to perform* $P'\_ask\_P\_true$*.*

□

**Proof of Lemma 11** In the initial state of $Sys_H$ all components but $H'$ are in their state labeled $t$. The only enabled connector is $eval\_H' \rightarrow H$ with $f($"$call\_eval(H)$"$) = eval\_H' \rightarrow H$. Thus, $\sigma(1) = f(Seq_H(1))$. Lemma 11 will be proved by induction, i.e. we have to show that, if $f(Seq_H(i)) = \sigma(i)$ is performed,

under the assumption $f(Seq_H(j)) = \sigma(j)$ for $1 \leq j \leq i$, the connector $\in C'$ that will be performed next is $f(Seq_H(i+1))$. In fact we show that $f(Seq_H(i+1))$ eventually becomes enabled, such that in between only connectors $\in C \setminus C'$ are performed.

We will now consider the three possible cases for $Seq_H(i)$.

**Induction - Case 1** Consider $Seq_H(i) =$ "$eval(P) = true$", i.e. $\sigma(i) = f(\text{"}eval(P) = true\text{"}) = P'\_ask\_P\_true$ where $P'$ is the subformula $P$ is included in and $P' = H'$ if $P = H$. If existent, let $P''$ be the predecessor of $P'$ ($P'' = H'$ if $P' = H$). It is clear, that $P$ is in its state labeled $t$. There are five cases:

Case 1.a) $P' = P \wedge \tilde{P}$, then $Seq_H(i+1) =$ "$call\_eval(\tilde{P})$". This means, that $P'$ waits to perform $eval\_P' \rightarrow \tilde{P}$. From Observation 13 follows the same for $\tilde{P}$ as well. It follows that the only newly enabled connector in $Sys_H$ is $eval\_P' \rightarrow \tilde{P} = f(Seq_H(i+1))$.

Case 1.b) $P' = \tilde{P} \wedge P$, then $Seq_H(i+1) =$ "$eval(P') = true$". $P'$ waits to perform $P''\_ask\_P'\_true = f(Seq_H(i+1))$ and from Observation 12 follows that this is the only newly enabled connector in $Sys_H$.

Case 1.c) $P' = \neg P$, then $Seq_H(i+1) =$ "$eval(P') = false$". $P'$ waits to perform $P''\_ask\_P'\_false = f(Seq_H(i+1))$ and from Observation 12 follows that this is the only newly enabled connector in $Sys_H$.

Case 1.d) $P' = \exists x_i.P$, then $Seq_H(i+1) =$ "$eval(P') = true$". $P'$ waits to perform $P''\_ask\_P'\_true = f(Seq_H(i+1))$ and from Observation 12 follows that this is the only newly enabled connector in $Sys_H$.

Case 1.e) $P' = H'$, then $i = length(Seq_H)$, i.e. there is no next word on $Seq_H$ and no new connector $\in C'$ is enabled in $Sys_H$.

**Induction - Case 2** Consider $Seq_H(i) =$ "$eval(P) = false$", i.e. $\sigma(i) = f(\text{"}eval(P) = false\text{"}) = P'\_ask\_P\_false$ where $P'$ is the predecessor of $P$ and $P' = H'$ if $P = H$. If existent, let $P''$ be the predecessor of $P'$ ($P'' = H'$ if $P' = H$). It is clear, that $P$ is in its state labeled $t$. There are five cases:

Case 1.a) $P' = P \wedge \tilde{P}$, then $Seq_H(i+1) =$ "$eval(P') = false$". $P$ waits to perform $P''\_ask\_P'\_false = f(Seq_H(i+1))$ which is, according to Observation 12, enabled by $P''$.

Case 1.b) $P' = \tilde{P} \wedge P$, analogously to case 1.

Case 1.c) $P' = \neg P$, analogously to case 1.

Case 1.d) $P' = \exists x_i.P$, then there must be $j < i$ with $Seq_H(j) =$ "$call\_eval(P)$", i.e. if $P$ was evaluated false then $P$ was called by eval previously. Let $j$ be the largest value with this property. By assumption follows that $j < i$ is the biggest index with $\sigma(j) = f(\text{"}call\_eval(P)\text{"}) = eval\_P' \rightarrow P$. In line 9 of the eval algorithm $P$ can be called by eval with $x_i$ set to true and afterwards with $x_i$ set to false. Accordingly, there are two cases for $Seq_H(j-1)$. Either $P'$ was called, i.e. $P$ is called with $x_i$ set to true or $P$ was evaluated false and was called a second time with $x_i$ set to false.

Case 1.d.a $Seq_H(j-1) =$ "$call\_eval(P')$" then $Seq_H(i+1) =$ "$call\_eval(P)$". By assumption follows that $\sigma(j-1) = eval\_P'' \rightarrow P' = f(\text{"}call\_eval(P')\text{"})$, i.e. either $set\_x_i'\_true$ or $ask\_true_{x_i'}$ was enabled after $\sigma(j-1)$ performed.

This assures that the component $x_i'$ is in its state $t$ after $\sigma(j)$ performed. Hence there was no connector involving component $P'$ since $\sigma(j)$, $x_i'$ is still in its state labeled $t$ when $\sigma(i)$ is performed. Therefore, after $\sigma(i)$ performs, the only newly enabled connector is $set\_x_i'\_false$, after that $set\_x_i\_false\_P' \rightarrow P$ and after that $P'$ waits to perform $eval\_P' \rightarrow P = f(\text{“}call\_eval(P)\text{”})$ which is, by Observation 13, assured to become enabled eventually.

Case 1.d.b $Seq_H(j-1) = \text{“}eval(P) = false\text{”}$ then $Seq_H(i+1) = \text{“}eval(P') = false\text{”}$. By assumption follows that $\sigma(j-1) = P'\_ask\_P\_false = f(\text{“}eval(P) = false\text{”})$. By Case 1.d.a follows that $x_i'$ is in its state labeled $f$ when $\sigma(i)$ is performed, i.e. after $\sigma(i)$, the only newly enabled connector is $ask\_false_{x_i'}$. When $ask\_false_{x_i'}$ is performed, it follows from Observation 12 that the only newly enabled connector is $P''\_ask\_P'\_false = f(\text{“}eval(P') = false\text{”})$.

Case 1.e) $P' = H'$, analogously to case 1.

**Induction - Case 3** Consider $Seq_H(i) = \text{“}call\_eval(P)\text{”}$, i.e. $\sigma(i) = f(\text{“}call\_eval(P)\text{”}) = eval\_P' \rightarrow P$ where $P'$ is the predecessor of $P$ and $P' = H'$ if $P = H$. There are four cases

Case 3.a) $P = \neg\tilde{P}$, then $Seq_H(i+1) = \text{“}call\_eval(\tilde{P})\text{”}$. $P$ waits to perform $eval\_P \rightarrow \tilde{P} = f(Seq_H(i+1))$ which is, enabled by $\tilde{P}$ accordingly to Observation 13 and therefore the only newly enabled connector.

Case 3.b) $P = \tilde{P}_1 \wedge \tilde{P}_2$, then $Seq_H(i+1) = \text{“}call\_eval(\tilde{P}_1)\text{”}$. $P$ waits to perform $eval\_P \rightarrow \tilde{P}_1 = f(Seq_H(i+1))$. From Observation 13 follows that this is the only new enabled connector.

Case 3.c) $P = \exists x_i.\tilde{P}$, then $Seq_H(i+1) = \text{“}call\_eval(\tilde{P})\text{”}$. In $Sys_H$ the only new enabled connector is either $set\_x_i'\_true$ or $ask\_true_{x_i'}$. If $set\_x_i'\_true$ is executed the only newly enabled connector is $set\_x_i\_true\_P \rightarrow \tilde{P}$. Anyway, if $set\_x_i\_true\_P \rightarrow \tilde{P}$ or $ask\_true_{x_i'}$ is executed, $P$ waits to perform $eval\_P \rightarrow \tilde{P} = f(\text{“}call\_eval(\tilde{P})\text{”})$ which is enabled by $\tilde{P}$ due to Observation 13.

Case 3.d) $P = x_l^r$, for $l \in \{1,\ldots,n\}$ and $r \in \{1,\ldots,k_i\}$. Then either $Seq_H(i+1) = \text{“}eval(P) = true\text{”}$ or $Seq_H(i+1) = \text{“}eval(P) = false\text{”}$. With Observation 14 follows that $P$ waits to perform either $f(Seq_H(i+1)) = P'\_ask\_P\_true$ or $f(Seq_H(i+1)) = P'\_ask\_P\_false$. Due to the fact that $P'$ waits to perform this connector as well, $f(Seq_H(i+1))$ is the only newly enabled connector $\in C'$.

□

**Proof of Theorem 7**

**Case $H \notin TQBF$** We have shown in Lemma 11 that after every execution of a connector in $C'$ exactly one connector in $C'$ is enabled (barring $H'\_ask\_H\_true/false$). The induction proves that every $\sigma$ corresponds to $Seq_H$, i.e. if $H \notin TQBF$ eventually the connector $H'\_ask\_H\_false$ is performed and there is no way $q^t$ can be reached.

**Case $H \in TQBF$** If $H \in TQBF$, eventually the connector $H'\_ask\_H\_true$ is performed. The only new enabled connector is $set\_H\_true\_H' \rightarrow H$. Let $P' \in K_1 \cup \{H'\}$ be a component and $P \in K_1$ its successor (i.e. $P'$ does not model a variable) such that $set\_P\_true\_P' \rightarrow P$ is enabled. There are four cases for the structure of $P$ and two for $P'$ if $set\_P\_true\_P' \rightarrow P$ is performed.

- $P$ models a variable, then it is assured that $P$ reaches its state labeled $t$ and no new connector is enabled.
- $P = \exists x_i.\tilde{P}$, then either $set\_x_i'\_true$ or $ask\_true_{x_i'}$ becomes enabled. Anyway, it is assured, that $x_i'$ reaches its state labeled $t$ and $set\_\tilde{P}\_true\_P \rightarrow \tilde{P}$ becomes enabled.
- $P = \tilde{P} \wedge \bar{P}$ or $P = \neg\tilde{P}$, then $set\_\tilde{P}\_true\_P \rightarrow \tilde{P}$ becomes enabled.

- $P' = P \wedge \bar{P}$, then $set\_\bar{P}\_true\_P' \rightarrow \bar{P}$ becomes enabled.
- In any other case, $P'$ reaches its state labeled $t$.

By induction follows, that eventually all components reach their state labeled $t$. From this it follows that

$$H \in TQBF \Leftrightarrow (Sys_H, q^t) \in TRIST$$