Transformations for Compositional Verification of Assumption-Commitment Properties*

Ahmed Mahdi[1], Bernd Westphal[2], and Martin Fränzle[1]

[1] Carl von Ossietzky Universität,
Ammerländer Heerstraße 114-118, 26111 Oldenburg, Germany
[2] Albert-Ludwigs-Universität Freiburg,
Georges-Köhler-Allee 52, 79110 Freiburg, Germany

Abstract. This paper presents a transformation-based compositional verification approach for verifying assumption-commitment properties. Our approach improves the verification process by pruning the state space of the model where the assumption is violated. This exclusion is performed by transformation functions which are defined based on a new notion of edges supporting a property. Our approach applies to all computational models where an automaton syntax with locations and edges induces a transition system semantics in a consistent way which is the case for hybrid, timed, Büchi, and finite automata. We have successfully applied our approach to Fischer's protocol.

1 Introduction

Many systems in real life are hybrid or real-time systems. Designing, developing, and verifying properties of these systems are becoming more and more complex. Hybrid and real-time systems are modelled by computational models such as *hybrid* and *timed automata*, respectively. This enables us to verify desired properties in these system models. The verification process is challenging because system models are increasingly complex and verification tools face the well-known space explosion problem.

We propose a new technique to improve the memory usage and time consumption of the verification process of *assumption-commitment* specifications. An assumption-commitment specification consists of an assumption and a commitment, which is required to hold if the assumption holds. For example, in the industrial field, *contracts* [7, 8, 19] consisting of assumptions and guarantees are used for component specifications. If the assumptions of a contract are fulfilled, then the guarantees have to hold. Consider, e.g., the avionics brake system [18]: if either the first or the second command units fail (no double failures), then the system has to guarantee that the brake system is safe.

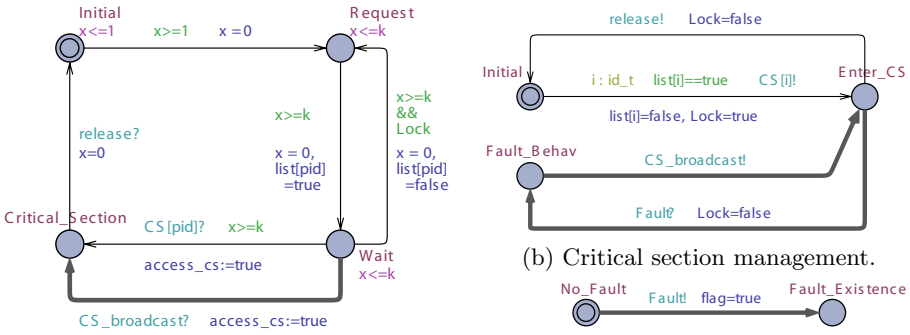
Our approach is based on the new notion of edges *supporting* a specification. Intuitively, a specification is supported by an edge if there is a computation

* Partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center SFB/TR 14 AVACS (`http://www.avacs.org`).

path in the model's semantics which satisfies the specification and uses that edge. That is, if the edge is *reachable* by a computation path which satisfies the specification. Instead of verifying an assumption-commitment property on the model, we apply a source-to-source transformation to the model, where those edges which *do not support* the assumption are effectively disabled. This transformation excludes computation paths from the verification process which are irrelevant for the overall property because they violate the assumption. Thereby, our approach decreases complexity already before running a model checking tool. Furthermore, our approach is independent from particular model checking tools as we transform the model and leave the model checking procedure untouched. We develop our approach for a generalized notion of *automata* consisting of directed, action-labelled edges between locations in order to uniformly treat computational models such as finite and Büchi automata, timed and hybrid automata, and even programs. A necessary assumption of our approach is that the operational semantics by which an automaton induces a transition system is *consistent* for the syntactical transformations. This consistency assumption is typically satisfied by the standard semantics. Furthermore, our approach is particularly well-suited for systems which provide many functions and operation modes, e.g. a plane's brake system may offer landing and taxiing modes. For validation purposes, it is useful to have only a single system model including all features but verification may practically be infeasible on such a model. Given an assumption-commitment specification, where the assumption limits the focus to only some features, our approach allows to mechanically create a smaller verification model which is guaranteed to reflect the relevant behaviour of the original model. Thereby, there is no more need to create specially tailored verification models manually.

Related Work. There are many works [4, 10, 12, 21] on excluding irrelevant computation paths from the verification process by abstracting the original model. Our work, in contrast, is a source-to-source transformation hence abstractions can still be applied. The exclusion of model behaviour by a source-to-source transformation proposed in [14] only considers networks of timed automata with disjoint activities. Slicing of timed automata [11] removes locations and clock and data variables on which a given property does not depend on, thus it also keeps variables on which an assumption depends while our approach may remove the corresponding behaviour. With partial model checking [2], verification problems are modularized by computing weakest context specifications through quotienting, which allows to successively remove components completely from the verification problem. We are instead trying to pragmatically reduce the size of components before composition by exploiting the specification. Both approaches could well go together. Static contract checking for functional programs [22] is dealing with a very different class of computational objects and relies heavily on assumptions local to the individual functions, while our approach is meant to also "massage" the global specification into the components.

The paper is structured as follows. Firstly, we motivate the idea of our approach using Fischer's protocol. Section 3 introduces generalized automata and



(a) Process. (b) Critical section management. (c) Fault detection.

Fig. 1: Uppaal model of the Fischer's protocol with direct fault detection.

Section 4 defines the transformations 'edge removal' and 'edge redirection'. Section 5 introduces the concept of supporting edges and Section 6 shows how our approach is used in practice. Finally, we summarize the benefits of using our approach.

2 Motivating Example: Fischer's Protocol with Faults

Embedded systems are nowadays expected to provide increasingly many functions and different modes. One example are modes for fault detection and handling. In the real-world, faults cannot be avoided in general: wires may break, radio frequencies may continuously be blocked, and physical sensors and actors may fail. One way to deal with this situation is to detect and display faults in order to, e.g., inform users to take countermeasures against the fault. A system is then correct if and only if it delivers regular functionality *unless* a fault is displayed. One requirement is then that under the assumption of the absence of faults, the system functions properly.

As a simple example, consider a variant of the well-known Fischer's protocol [5]. Figure 1 shows the Uppaal [3] demo model of the Fischer's protocol which we extended with fault detection. If no fault occurs, processes (cf. Figure 1a) pose a request to enter their critical section by the shared variable list. The critical section manager (cf. Figure 1b) grants access to the critical section by a synchronisation on CS and expects notification of leaving the critical section on channel release. Our extensions are indicated by thick edges: if a fault occurs, processes may enter the critical section bypassing the manager and thereby violate the mutual exclusion property. For simplicity, we merged the environment model which triggers faults and fault detection in Figure 1c. Location Fault_Existence models the display of a fault occurrence. Table 1 shows results from attempts to verify mutual exclusion given no faults occur, formally, $(\Box p) \rightarrow (\Box q)$ with $p = \neg\mathsf{Fault\_Existence}$ and $q = (\forall\, i \neq j : \mathit{Process} \bullet \neg(i.\mathsf{Crit\_Sec} \wedge j.\mathsf{Crit\_Sec}))$. Note that it is sufficient to check the Uppaal query $\mathsf{A}\Box(p \rightarrow q)$ for the considered model due to the immediate detection. In the original model as shown in

#	original model query: A□($p \to q$) seconds	MB	kStates	non-supporting removed query: A□q seconds	MB	kStates	non-supporting redirected query: A□($p \to q$) seconds	MB	kStates
4	0.08	5.1	9.9	0.02	4.5	3.0	0.02	4.5	3.0
5	1.02	12.2	82.9	0.22	6.3	20.7	0.23	6.3	20.7
6	11.46	67.9	683.9	1.65	17.7	140.0	2.18	17.7	140.0
7	127.33	516.0	5,610.7	13.18	62.8	933.1	19.40	88.2	933.1
8	1,274.64	4,193.8	47,630.3	107.18	365.3	6,158.6	168.37	562.7	6,158.6
9	>2,000.00	–	–	894.83	2,297.0	40,310.8	1359.34	3,659.0	40,310.8

Table 1: Figures for verifying mutual exclusion.[3]

(a) Delayed fault detection. (b) Delayed fault detection redirected.

Fig. 2: Treating delayed detection.

Figure 1, Uppaal does not succeed to verify a system with 9 processes in 2,000 seconds, a system with 8 processes takes about 20 min. to be successfully verified.

A clever verification engineer may observe that the verification is expensive because also all fault scenarios are explored – and found to violate the assumption that there are no faults. Faults are triggered by the edge in Figure 1c, thus if this edge is removed, all fault scenarios are excluded from the model. Column "non-supporting removed" of Table 1 shows results from the verification of the mutual exclusion property in the modified model. We observe savings in time and memory consumption of an order of magnitude for the larger instances and verification even scales better in the number of processes.

Note that Figure 1 is a special case, because fault detection is immediate (no delay between occurrence and detection) and persistent. For real-world systems, fault detection often needs time so there may be small durations of time, where the system cannot guarantee proper operation but where the fault is not yet displayed. An Uppaal model of delayed fault detection for Fischer's protocol is shown in Figure 2a. Here, the query stated above does not hold. Instead, we need to check the commitment "globally mutual exclusion" under the assumption "globally no fault". Still, we can effectively exclude fault scenarios from the verification procedure by *redirecting* the right edge to a fresh sink location deadend (cf. Figure 2b). Checking the overall property then reduces to checking that violations of mutual exclusion are always finally followed by fault display (results are provided in Section 6).

In the following, we develop a theory of redirecting and removing edges in order to prune the state space for assumption-commitment specifications. Our approach provides a formal justification of the removal applied *ad-hoc* above: if we prove that the edge in Figure 1c does *not* support the assumption, our results guarantee that we can conclude from a positive verification result for the

[3] All results: Linux x64, 16 Quad-Core Opteron 8378, 132 GB, Uppaal 4.1.18

transformed model to the original model. That is, we guarantee that no relevant scenarios are missed – a guarantee which cannot be given for manual ad-hoc transformations. We show that redirecting edges reflects all computation paths which satisfy the assumption. The redirection transformation is semantically optimal, if all edges which do not support the assumption, are redirected.

3 Compositional Verification of Assumption-Commitment Specifications for Generalized Automata

In the following, we consider a generalized notion of automata which allows us to treat, among others, timed and hybrid automata uniformly.

Definition 1 (Automaton). *An* automaton $\mathcal{A} = (Loc, Act, E, L_{ini})$ *consists of a finite set of* locations Loc*, a finite set of* actions Act*, a set* $E \subseteq Loc \times Act \times Loc$ *of directed* edges*, and a set of* initial locations $L_{ini} \subseteq Loc$*. Each edge* $(\ell, \alpha, \ell') \in E$ *has a* source location ℓ*, an action* α*, and a* destination location ℓ'*.*

Finite, Büchi, timed [1], and hybrid automata [9] can be represented as automata in the sense of Definition 1. For example, for timed automata, we can consider pairs of locations and invariants as locations, and triples consisting of synchronization, guard, and update vector as action. Thereby, the alphabet of a timed automaton is represented in the set of actions . Moreover, *programs* [15] are automata as follows: the nodes in the control flow graph (CFG) become automaton locations and the edges in the CFG become automaton edges labelled with statements.

In the following definition, we introduce an edge-centric notion of operational semantics for generalized automata, i.e. there is one transition relation per edge and one dedicated additional transition relation. This allows for a simple definition of *support* in Section 5. Later we will characterise those operational semantics for which our approach applies as *consistent* (cf. Section 4).

Definition 2 (Operational Semantics). *Let* V *be a set of* states *and* Aut *a set of automata. An* operational semantics *of* Aut *(over* V*) is a function* $\mathcal{T}$ *which assigns to each automaton* $\mathcal{A} = (Loc, Act, E, L_{ini}) \in Aut$ *a labelled transition system* $\mathcal{T}(\mathcal{A}) = (Conf, \Lambda, \{\xrightarrow{\lambda} \mid \lambda \in \Lambda\}, C_{ini})$ *where* $Conf \subseteq Loc \times V$ *is the set of* configurations*,* $\Lambda = E \cup \{\bot\}$*, where* $\bot \notin E$*, is the set of* labels*,* $\xrightarrow{\lambda} \subseteq Conf \times Conf$ *are* transition relations*, and* $C_{ini} \subseteq (L_{ini} \times V) \cap Conf$ *is the set of* initial configurations*.*

The standard operational semantics of hybrid and timed automata induce an operational semantics of the aforementioned generalized automata .

An operational semantics induces computation paths as usual. In addition, we distinguish computation paths based on the occurring labels.

Definition 3 (Computation Path). *A* computation path *of automaton* $\mathcal{A} \in Aut$ *under operational semantics* $\mathcal{T}(\mathcal{A}) = (Conf, \Lambda, \{\xrightarrow{\lambda} \mid \lambda \in \Lambda\}, C_{ini})$ *is an initial and consecutive, infinite or maximally finite sequence* $c_0 \xrightarrow{\lambda_1} c_1 \xrightarrow{\lambda_2} \cdots$ *where* $c_0 \in C_{ini}$ *(initiation) and for each* $i \in \mathbb{N}_0$*,* $(c_i, c_{i+1}) \in \xrightarrow{\lambda_{i+1}}$ *(consecution).*

Let E be the set of edges of $\mathcal{A}$. We use $\Xi_{\mathcal{T}}(\mathcal{A}, F)$ to denote the set of computation paths of $\mathcal{A}$ where only label $\perp$ and labels from $F \subseteq E$ occur. $\Xi_{\mathcal{T}}(\mathcal{A}) := \Xi_{\mathcal{T}}(\mathcal{A}, E)$ denotes the set of all computation paths of $\mathcal{A}$ (under $\mathcal{T}$).

Our approach applies to so-called assumption-commitment specifications as defined in the following. Note that we use a *semantical* characterisation of specifications for simplicity. A specification is a set of sequences, i.e. we consider path specifications. Specifications can *syntactically* be described by, e.g., LTL [17].

Definition 4 (Assumption-Commitment Specification). *A* specification *over* alphabet Σ *is a set $S \subseteq \Sigma^* \cup \Sigma^\omega$ of finite or infinite sequences over Σ.*

A specification is called assumption-commitment *specification if there are specifications P and Q such that $S = \overline{P} \cup Q$, where $\overline{P}$ denotes the complement of P in $\Sigma^* \cup \Sigma^\omega$, i.e. the set $(\Sigma^* \cup \Sigma^\omega) \setminus P$. We write $P \to Q$ to denote the assumption-commitment specification $\overline{P} \cup Q$. A set $p \subseteq \Sigma$ is called* atomic proposition*, and the specification $\Box p := p^* \cup p^\omega$ is called an* invariant*.*

We establish the satisfaction relation between automata and specifications based on the observable behaviour of the automaton, i.e., the sequence of configurations obtained by disregarding the labelled transitions.

Definition 5 (Satisfying a Specification). *Let $\xi = c_0 \xrightarrow{\lambda_1} c_1 \xrightarrow{\lambda_2} \cdots \in \Xi_{\mathcal{T}}(\mathcal{A})$ be a computation path of automaton $\mathcal{A}$ under operational semantics $\mathcal{T}$. The* observable behaviour *of ξ is the sequence $\downarrow\xi = c_0, c_1, \ldots$ We use $\mathcal{O}_{\mathcal{T}}(\mathcal{A})$ to denote the set of observable behaviours of the computation paths of $\mathcal{A}$ under $\mathcal{T}$, i.e. $\mathcal{O}_{\mathcal{T}}(\mathcal{A}) = \{\downarrow\xi \mid \xi \in \Xi_{\mathcal{T}}(\mathcal{A})\}$. Automaton $\mathcal{A}$ is said to* satisfy *the specification S (under $\mathcal{T}$), denoted by $\mathcal{A} \models_{\mathcal{T}} S$, if and only if the set of observable behaviours of $\mathcal{A}$ (under $\mathcal{T}$) is a subset of S, i.e. if $\mathcal{O}_{\mathcal{T}}(\mathcal{A}) \subseteq S$.*

The following theorem states two observations for assumption-commitment specifications S of the form $P \to Q$. Firstly, whether an automaton satisfies S depends exactly on the observable behaviours satisfying P. That is, in order to check an automaton $\mathcal{A}_1$ against S, we may as well check $\mathcal{A}_2$ (even under a different operational semantics) as long as $\mathcal{A}_1$ and $\mathcal{A}_2$ (under the considered semantics) agree on the observable behaviours satisfying P. Secondly, it is possible to verify satisfaction of S by an automaton through checking only Q in an overapproximation of the automaton's observable behaviour.

Theorem 1 (Compositional Verification). *Let $\mathcal{A}_1 \in Aut_1$ and $\mathcal{A}_2 \in Aut_2$ be automata and $\mathcal{T}_1$ and $\mathcal{T}_2$ operational semantics for Aut_1 and Aut_2, respectively. Let $P \to Q$ be an assumption-commitment specification.*

1. *The* common-P*-rule: whenever the set of observable behaviours of $\mathcal{A}_1$ that satisfy P is equal to the set of observable behaviours of $\mathcal{A}_2$ that satisfy P, then $\mathcal{A}_1$ satisfies $P \to Q$ if and only if $\mathcal{A}_2$ satisfies $P \to Q$, i.e.,*

$$\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap P = \mathcal{O}_{\mathcal{T}_2}(\mathcal{A}_2) \cap P \implies (\mathcal{A}_1 \models_{\mathcal{T}_1} P \to Q \iff \mathcal{A}_2 \models_{\mathcal{T}_2} P \to Q).$$

2. *The* over-approximating-P*-rule: whenever the set of observable behaviours of* $\mathcal{A}_1$ *that satisfy* P *is a subset of the set of observable behaviours of* $\mathcal{A}_2$*, then* $\mathcal{A}_1$ *satisfies* $P \to Q$ *if* $\mathcal{A}_2$ *satisfies* P*, i.e.,*

$$\mathcal{O}_{\mathcal{T}_1}(\mathcal{A}_1) \cap P \subseteq \mathcal{O}_{\mathcal{T}_2}(\mathcal{A}_2) \implies (\mathcal{A}_2 \models_{\mathcal{T}_2} Q \implies \mathcal{A}_1 \models_{\mathcal{T}_1} P \to Q).$$

In general, the second implication does not hold in the other direction.

(Semi-)admissible transformation functions as introduced in the next section entail the premises of the over-approximating-P- and the common-P-rule.

4 Automata Transformations

In this section, we define a general concept of transformations for automata. We call transformations which preserve a specification *admissible* and those which over-approximate a specification *semi-admissible*. After that, we introduce the two transformations *redirecting edges* and *removing edges*.

Definition 6 (Transformation). *Let* Aut *be a set of automata. A* transformation *is a function* $\mathcal{F} : Aut \to Aut$ *which assigns to each* original automaton $\mathcal{A} \in Aut$ *a* transformed automaton $\mathcal{F}(\mathcal{A}) \in Aut$*.*

Definition 7 (Admissible Transformation). *Let* $\mathcal{T}$ *be an operational semantics of* Aut *and* S *a specification. Transformation* $\mathcal{F}$ *on* Aut *is called*

1. admissible *for* S *(under* $\mathcal{T}$*) if and only if for each automaton* $\mathcal{A} \in Aut$*, the observable behaviours of* $\mathcal{A}$ *and* $\mathcal{F}(\mathcal{A})$ *under* $\mathcal{T}$ *coincide on* S*, i.e. if*

$$\forall\, \mathcal{A} \in Aut \bullet \mathcal{O}_{\mathcal{T}}(\mathcal{F}(\mathcal{A})) \cap S = \mathcal{O}_{\mathcal{T}}(\mathcal{A}) \cap S.$$

2. semi-admissible *for* S *(under* $\mathcal{T}$*) if and only if for each automaton* $\mathcal{A} \in Aut$*, the observable behaviour of* $\mathcal{F}(\mathcal{A})$ *under* $\mathcal{T}$ *over-approximates the observable behaviour of* $\mathcal{A}$ *in* S*, i.e. if* $\forall\, \mathcal{A} \in Aut \bullet \mathcal{O}_{\mathcal{T}}(\mathcal{A}) \cap S \subseteq \mathcal{O}_{\mathcal{T}}(\mathcal{F}(\mathcal{A}))$*.*

The following lemma states the benefit of transformations which are (semi-) admissible for the assumption of assumption-commitment specifications: the original and the transformed automaton obtained by an admissible transformation satisfy the premise of the common-P-rule, for a semi-admissible transformation, the premise of the over-approximating-P-rule holds.

Lemma 1. *Let* $\mathcal{T}$ *be an operational semantics of* Aut*,* $S = P \to Q$ *an assumption-commitment specification, and* $\mathcal{F}$ *a transformation on* Aut*.*

1. *If* $\mathcal{F}$ *is admissible for* P*, then for* $\mathcal{A} \in Aut$*,* $\mathcal{F}(\mathcal{A}) \models S$ *if and only if* $\mathcal{A} \models S$*.*
2. *If* $\mathcal{F}$ *is semi-admissible for* P*, then for* $\mathcal{A} \in Aut$*,* $\mathcal{F}(\mathcal{A}) \models Q$ *implies* $\mathcal{A} \models S$*.*

Proof. Definition 7 and Theorem 1. □

The first proposed transformation function *redirects* a set of edges in a given automaton to a *new location*. It is defined as follows.

Definition 8 (Redirecting Edges). *Let $\mathcal{A} = (Loc, Act, E, L_{ini})$ be an automaton, $F \subseteq E$ a set of edges, and $\dashv \notin Loc$ a fresh location. We use $\mathcal{A}[F/\dashv]$ to denote the automaton $(Loc \cup \{\dashv\}, Act, E', L_{ini})$ where*

$$E' = (E \setminus F) \cup \{(\ell, \alpha, \dashv) \mid (\ell, \alpha, \ell') \in F\}.$$

We say $\mathcal{A}[F/\dashv]$ is obtained from $\mathcal{A}$ by redirecting *the edges in F (to $\dashv$).*

A transformation is a *syntactical* operation, thus the observable behaviour of a transformed automaton may in general, given a sufficiently pathological operational semantics, not resemble the behaviour of the original automaton at all. The following definition of consistency states minimal sanity requirements on operational semantics which we need in order to effectively use the redirection transformation. These requirements are directly satisfied by the standard semantics of, e.g., timed automata.

Definition 9 (Consistent for Redirection). *An operational semantics $\mathcal{T}$ for Aut over states V is called* consistent *(for redirection) if and only if for each automaton $\mathcal{A} = (Loc, Act, E, L_{ini}) \in Aut$, there is a location $\dashv$ such that $\mathcal{A}[F/\dashv] \in Aut$ and $\mathcal{T}(\mathcal{A}[F/\dashv]) = (Conf', \Lambda', \{\xrightarrow{\lambda}{}' \mid \lambda \in \Lambda'\}, C'_{ini})$ where*

1. *the set of configurations over the old locations, and the transition relations for $\perp$ and the unchanged edges do not change, i.e.*

$$Conf' \cap (Loc \times V) = Conf, \quad \forall\, e \in E \setminus F \bullet \xrightarrow{e}{}' = \xrightarrow{e}, \\ \text{and} \quad \xrightarrow{\perp}{}' \cap (Conf \times Conf) = \xrightarrow{\perp},$$

2. *$\mathcal{T}(\mathcal{A}[F/\dashv])$ simulates transitions induced by edges from F and vice versa, and the $\perp$-transition relation does not leave $\dashv$, i.e.*

$$\xrightarrow{(\ell,\alpha,\dashv)}{}' = \{(c, \langle \dashv, v' \rangle) \mid \exists\, e = (\ell, \alpha, \ell') \in F \bullet (c, \langle \ell', v' \rangle) \in \xrightarrow{e}\}, \\ \text{and} \quad \forall\, v, v' \in V, \ell' \in Loc' \bullet ((\dashv, v), (\ell', v')) \in \xrightarrow{\perp}{}' \implies \ell' = \dashv$$

3. *the fresh location $\dashv$ is not initial, i.e. $C'_{ini} = C_{ini}$.*

The following lemma states that for consistent semantics, the redirection transformation affects only behaviours where redirected edges are used.

Lemma 2. *Let $\mathcal{T}$ be an operational semantics of Aut which is consistent for redirection. Let $\mathcal{A} \in Aut$ be an automaton with edges E and $F \subseteq E$. Then there is a location $\dashv$ such that $\Xi_{\mathcal{T}}(\mathcal{A}[F/\dashv], E \setminus F) = \Xi_{\mathcal{T}}(\mathcal{A}, E \setminus F)$.*

The following second transformation *removes* edges from an automaton.

Definition 10 (Removing edges). *Let $\mathcal{A} = (Loc, Act, E, L_{ini})$ be an automaton and $F \subseteq E$ a set of edges. We use $\mathcal{A} \setminus F$ to denote the automaton $(Loc, Act, E \setminus F, L_{ini})$. We say $\mathcal{A} \setminus F$ is obtained from $\mathcal{A}$ by* removing *F.*

As for redirection, we want that all computation paths of the original automaton that take only non-removed edges are preserved in the new automaton. A sufficient criterion is the following notion of consistency *for removal*.

Definition 11 (Consistent Operational Semantics for Removal). *An operational semantics $\mathcal{T}$ for Aut is called consistent (for removal) if and only if for each automaton $\mathcal{A} \in Aut$, $\mathcal{A} \setminus F \in Aut$ and*

$$\mathcal{T}(\mathcal{A} \setminus F) = (Conf, \Lambda \setminus F, \{\xrightarrow{\lambda} \mid \lambda \in \Lambda \setminus F\}, C_{ini}),$$

given $\mathcal{T}(\mathcal{A}) = (Conf, \Lambda, \{\xrightarrow{\lambda} \mid \lambda \in \Lambda\}, C_{ini})$. That is, if the operational semantics of $\mathcal{A} \setminus F$ is obtained from the operational semantics of $\mathcal{A}$ by removing some transition relations and leaving everything else unchanged.

Lemma 3. *Let $\mathcal{T}$ be an operational semantics of Aut which is consistent for removal. Let $\mathcal{A} = (Loc, Act, E, L_{ini}) \in Aut$ be an automaton and $F \subseteq E$ a set of edges. Then $\Xi_{\mathcal{T}}(\mathcal{A} \setminus F, E \setminus F) = \Xi_{\mathcal{T}}(\mathcal{A}, E \setminus F)$.*

5 Supporting Edges

In this section, we introduce the novel concept of supporting edges, based on *edge reachability*. This concept identifies a relation between a specification and edges. Informally, an edge supports a specification if and only if there is a computation path which satisfies the specification and where that edge is taken.

Definition 12 (Supporting Edges). *Let $\mathcal{T}$ be an operational semantics of Aut and $\mathcal{A} \in Aut$ an automaton with edges E. An edge $e \in E$*

1. supports specification S *(under $\mathcal{T}$) if and only if there is a computation path where label e occurs and whose observable behaviour is in S, i.e. if*

$$\exists\, \xi = c_0 \xrightarrow{\lambda_1} c_1 \xrightarrow{\lambda_2} \cdots \in \Xi_{\mathcal{T}}(\mathcal{A})\ \exists\, i \in \mathbb{N} \bullet \lambda_i = e \wedge {\downarrow}\xi \in S.$$

2. supports atomic proposition p *(under $\mathcal{T}$) if and only if there is a computation path where label e occurs between two configurations that are in p, i.e. if*

$$\exists\, \xi = c_0 \xrightarrow{\lambda_1} c_1 \xrightarrow{\lambda_2} \cdots \in \Xi_{\mathcal{T}}(\mathcal{A})\ \exists\, i \in \mathbb{N} \bullet \lambda_i = e \wedge \{c_{i-1}, c_i\} \subseteq p.$$

3. potentially supports atomic proposition p *(under $\mathcal{T}$) if and only if there are two configurations of $\mathcal{T}(\mathcal{A}) = (Conf, \Lambda, \{\xrightarrow{\lambda} \mid \lambda \in \Lambda\}, C_{ini})$ which are in p and in $\xrightarrow{e}$-relation, i.e. if $\exists\, c, c' \in Conf \cap p \bullet (c, c') \in \xrightarrow{e}$.*

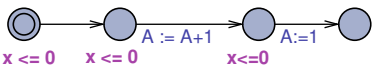


Fig. 3: Notions of support.

Note that Definition 12.1 is the strongest and 12.3 the weakest notion of support as stated in the following lemma. For example, consider the timed automaton in Figure 3 where A is initially 0. If delays with duration 0 are not allowed, the leftmost edge supports the proposition $A = 0$, but not the specification $\Box A = 0$. The middle edge potentially supports the proposition, but it does not support it. The rightmost edge does not even potentially support the proposition $A = 0$.

Lemma 4. *Let $\mathcal{A} \in Aut$ be an automaton and p an atomic proposition.*

1. *If an edge e of $\mathcal{A}$ supports the invariant $\Box p$ (under $\mathcal{T}$), then e supports the proposition p (under $\mathcal{T}$), but in general not vice versa.*
2. *If e supports p, then e potentially supports p, but in general not vice versa.*

Our main result shows that redirecting and removing edges which *do not support* a specification S are admissible and semi-admissible for S, respectively.

Theorem 2 (Admissibility). *Let $\mathcal{T}$ be an operational semantics of Aut with states V and let S be a specification over Σ. Let F be a set of edges of automaton $\mathcal{A} \in Aut$ which do not support S under $\mathcal{T}$.*

1. *$\mathcal{F}^F_{\mathsf{rd}} : \mathcal{A} \mapsto \mathcal{A}[F/\dashv]$ is admissible for S if $\mathcal{T}$ is consistent for redirection and if S* does not refer to *the fresh location $\dashv$, i.e. if $(\{\dashv\} \times V) \cap \Sigma = \emptyset$.*
2. *$\mathcal{F}^F_{\mathsf{rm}} : \mathcal{A} \mapsto \mathcal{A} \setminus F$ is semi-admissible for S if $\mathcal{T}$ is consistent for removal.*

To apply Theorem 2 we need a set of edges which do not support the given specification. In general, detecting edges which do not support a specification is as expensive as *reachability checking*. Though if the specification is an invariant, the contrapositions of the implications in Lemma 4 are particularly useful: if an edge does not potentially support a proposition, then it does not support the proposition, and if an edge does not support a proposition p, then it does not support the invariant $\Box p$. A sufficient criterion for an edge not (potentially) supporting a proposition p is to be a *cause* or a *witness*. An edge e is a *cause* of a violation of p if the p is always violated after taking this edge, e.g., if the action of e causes p not to hold. Similarly, an edge is a *witness* of a violation of p if p is necessarily violated when e is taken, e.g., if the guard of e implies $\neg p$. Removal of witnesses is even admissible. There are sufficient syntactical criteria to detect causes and witnesses. Furthermore, detection of potential support can be reduced to an SMT problem for the formula given by Definition 12.3 and attacked by SMT solvers like SMTInterpol [6]. For the special case of timed automata and bounded-integer propositions, a procedure based on the well-known reaching definitions static analysis detects all edges which support an atomic proposition [13]. Considering *all* edges which do not support a given specification is optimal in the sense that removing or redirecting any more edges breaks (semi-)admissibility. But it is not necessary to determine *all* non-supporting edges in order to obtain an optimal reduction of behavior. It is sufficient to determine all *points of no return* (PNR) for a given specification, i.e., edges which are the first on a computation path which do not support the specification. Causes and witnesses are often PNRs.

Networks of Automata. All previous discussions consider a single automaton. However, most practical models are networks of automata. In the following, we discuss briefly how our approach is applied to networks of automata and automata templates. For timed automata, each network has an equivalent timed automaton, the *parallel composition*. Edges in the parallel composition are constructed from internal transitions of automata in the network, or (with broadcast) from synchronisation edges of one or more automata in the network over a

channel. An edge e in the network *supports a specification* if and only if there is an edge in the parallel composition which supports the specification and which is constructed from e. Edges not supporting a specification in this sense can safely be disabled by applying redirection or removal to the automata in the network. The same approach applies to hybrid automata and as neither redirection nor removal changes the set of actions, the sets of labels are preserved and thus no new computation paths emerge. In Uppaal, networks of automata are composed of automaton template instances. An edge in a template supports a specification if and only if there is an edge instance which supports the specification. That is, an edge can only be safely redirected or removed in the template, if all instances of this edge in the network do not support the specification.

Computation Paths vs. Runs The standard semantics of timed and hybrid automata distinguish between computation paths and *runs*, where the latter are computation paths with the *progress* property [16]. Interestingly, for timed automata, removing and redirecting edges have the same semantical effect if only *runs* are considered. However, in practice that will not give an obvious benefit, because verification tools typically check computation paths, not only runs.

6 Compositional Verification

In the previous sections, we have introduced a concept of transformation by either redirecting or removing edges and we introduced different notions of edges supporting a specification or an atomic proposition. This section proposes an approach to use the previous theory in practice with the Uppaal tool.

To use redirection and removal, we propose to apply the procedure shown in Figure 4 to all assumption-commitment properties $P \rightarrow Q$. The first step is to remove edges which don't support the assumption and check whether the resulting model satisfies the commitment Q. If Q is satisfied, we deduce that the original model satisfies the property by the over-approximating-P-rule from Theorem 1. Otherwise, we need to redirect the edges that do not support P. Then checking whether the resulting model satisfies $P \rightarrow Q$ yields the final verification result by the common-P-rule from Theorem 1. The reason for using removal before redirection is that removing edges leads to a smaller state space than redirecting, and consequently less time and memory consumption. For an example, see Table 1, right-most and middle column.

Input: automaton $\mathcal{A}$ with edges E, specification $S = P \rightarrow Q$, $F \subseteq E$ not supporting P

if $\mathcal{A} \setminus F \models Q$ **return** *true*;
else return $\mathcal{A}[F/\dashv] \models S$;

Fig. 4: Verification procedure.

Note that not all cases can be handled by removing, because we may remove edges which make a violation of the assumption observable. This is, e.g., the case for Fischer's protocol with delayed detection (cf. Section 2). In such cases, one has to use redirection to obtain a definite positive or negative answer. In order to alleviate *state-space explosion*, removing and redirecting is supposed to be performed locally at component level first. A *second sweep* can be done after the composition, but experiments show that the first, local one is the decisive.

#	original model query: $\neg q \rightsquigarrow \neg p$ seconds	MB	kStates	non-supporting redirected query: $\neg q \rightsquigarrow \neg p$ seconds	MB	kStates
3	0.10	3.9	6.0	0.07	3.9	2.7
4	5.90	29.7	109.6	0.87	5.1	37.0
5	788.20	130.1	2216.9	92.25	104.1	1250.0
6	>2,000.00	–	–	1037.24	228.7	3853.4
7	>2,000.00	–	–	>2,000.00	–	–

Table 2: Fischer's protocol with delayed fault detection.

Recall Fischer's protocol with immediate detection as introduced in Figure 1. According to our approach proposed above, we proceed as follows: the edges synchronizing on channel Fault in Figures 1c and 1b do not support the atomic proposition ¬Fault_Existence. As the automaton resulting from removing these edges satisfies the mutual exclusion property, we can conclude that the original model satisfies the assumption-commitment property by using Theorem 1.2. The verification results are stated in Table 1. If the fault detection is delayed as described in Section 2, then the automata model obtained from edge removal does not satisfy the mutual exclusion property (the commitment). So we can not conclude any beneficial results. Therefore we check whether the resulting automata model after redirecting satisfies the assumption-commitment property. Note that Tables 1 and 2 only report verification time. For the case study, identifying non-supporting edges using an SMT-solver takes less than one second and the time needed for the subsequent simple source-to-source transformation is negligible. Interestingly, using non-supporting edges enables us to reduce the LTL property $(\Box p) \rightarrow (\Box q)$, which is not directly supported by the TCTL fragment of Uppaal, to the leads-to query $\neg q \rightsquigarrow f$ where f is a fresh observer for non-supporting edges.

7 Conclusion

We presented a new technique of verifying assumption-commitment specifications in a large class of computational models, e.g., hybrid, timed, finite, and Büchi automata, and programs. The technique depends on transformations of automata by either redirecting or removing edges which do not support the assumption of the considered property. To this end, we introduced the new concept of "an edge supports a specification" which identifies a relation between specifications and edges in the automaton based on edge reachability. We showed for a model of Fischer's protocol that removing and redirecting edges significantly speeds up the verification process and improves the memory usage in comparison to verifying the same property in the original model without transformation.

Further work consists of an investigation of further uses of the notion of supporting edges, for example to indicate cut-points in a model where automata which over-approximate certain features of a multi-feature model can be inserted. Furthermore, there are methods to detect the supporting edges such as the *reaching definitions*-based approach in [13], but more powerful and efficient methods are needed. To this end, we will investigate syntactic criteria and significant extensions of the existing semantical methods.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. TCS 126(2), 183–235 (1994)
2. Andersen, H.R.: Partial model checking (extended abstract). In: LICS. pp. 398–407. IEEE Computer Society (1995)
3. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) SFM. LNCS, vol. 3185, pp. 200–236. Springer (2004)
4. Benedetto, M.D.D., Gennaro, S.D., D'Innocenzo, A.: Verification of hybrid automata diagnosability by abstraction. IEEE TAC 56(9), 2050–2061 (2011)
5. Budkowski, S., Cavalli, A.R., Najm, E. (eds.): Formal Description Techniques and Protocol Specification, Testing and Verification, FORTE XI / PSTV XVIII'98, IFIP Conference Proceedings, vol. 135. Kluwer (1998)
6. Christ, J., Hoenicke, J., Nutz, A.: Smtinterpol: An interpolating smt solver. In: Donaldson, A.F., Parker, D. (eds.) SPIN. Lecture Notes in Computer Science, vol. 7385, pp. 248–254. Springer (2012)
7. Damm, W.: Contract-based analysis of automotive and avionics applications: The SPEEDS approach. In: Cofer, D.D., et al. (eds.) FMICS. LNCS, vol. 5596, p. 3. Springer (2008)
8. Damm, W., et al.: Using contract-based component specifications for virtual integration testing and architecture design. In: DATE. pp. 1023–1028. IEEE (2011)
9. Henzinger, T.A.: The theory of hybrid automata. In: LICS. pp. 278–292. IEEE (1996)
10. Herbreteau, F., et al.: Lazy abstractions for timed automata. In: Sharygina et al. [20], pp. 990–1005
11. Janowska, A., Janowski, P.: Slicing timed systems. FI 60(1-4), 187–210 (2004)
12. Laarman, A., Olesen, M.C., et al.: Multi-core emptiness checking of timed büchi automata using inclusion abstraction. In: Sharygina et al. [20], pp. 968–983
13. Mahdi, A.: Compositional verification of computation path dependent real-time systems properties. Master's thesis, University of Freiburg (Apr 2012)
14. Muñiz, M., Westphal, B., et al.: Timed automata with disjoint activity. In: Jurdzinski, M., et al. (eds.) FORMATS. LNCS, vol. 7595, pp. 188–203. Springer (2012)
15. Nielson, F., et al.: Principles of program analysis (2. corr. print). Springer (2005)
16. Olderog, E.R., Dierks, H.: Real-time systems. Cambridge University Press (2008)
17. Pnueli, A.: The temporal logic of programs. In: FOCS. pp. 46–57. IEEE (1977)
18. SAE Int.: ARP-4761. Tech. rep., Aerospace Recommended Practice (1996)
19. Sangiovanni-Vincentelli, A.L., Damm, W., et al.: Taming Dr. Frankenstein: Contract-based design for cyber-physical systems. EJC 18(3), 217–238 (2012)
20. Sharygina, N., et al. (eds.): CAV 2013, LNCS, vol. 8044. Springer (2013)
21. Sher, F., Katoen, J.P.: Compositional abstraction techniques for probabilistic automata. In: IFIP TCS. LNCS, vol. 7604, pp. 325–341. Springer (2012)
22. Xu, D.N., Jones, S.L.P., Claessen, K.: Static contract checking for haskell. In: Shao, Z., Pierce, B.C. (eds.) POPL. pp. 41–52. ACM (2009)