Efficient Verification of Distributed Real-Time Systems with Broadcasting Behaviors*

Farn Wang[1,2] Li-Wei Yao[1] Ya-Lan Yang[2]

Dept. of Electrical Engineering[1]
Graduate Institute of Electronic Engineering[2]

National Taiwan University

1, Sec. 4, Roosevelt Rd., Taipei, Taiwan 106, ROC;
+886-2-33663602; FAX:+886-2-23671909;
farn@cc.ee.ntu.edu.tw; http://cc.ee.ntu.edu.tw/~farn

RED 8.0 is available at project REDLIB of SourceForge.

Abstract

Binary synchronization has been used extensively in the construction of mathematical models for the verification of embedded systems. Although it allows for the modeling of complex cooperation among many processes in a natural environment, not many tools have been developed to support the modeling capability in this regard. In this article, we first give examples to argue that special algorithms are needed for the efficient verification of systems with complex synchronizations. We then define our models of distributed real-time systems with synchronized cooperation among many processes. We present algorithms for the construction of BDD-like diagrams for the characterization of complex synchronizations among many processes. We present weakest precondition algorithms that take advantage of the just-mentioned BDD-like diagrams for the efficient verification of complex real-time systems. Finally, we report experiments and argue that the techniques could be useful in practice.

Keywords: distributed, real-time, model-checking, verification, synchronization

1 Introduction

In the verification of distributed real-time systems, the appropriate abstraction of the system behaviors is crucial to the balance between the precision of the models and the efficiency of verification. For instance, if we model how a missile is directed to hit a jet-fighter at the granularity of sub-atomic particle interaction, then of course we have an extremely precise model. However, such cumbersome models can also involve too many details irrelevant to the verification of the system and likely incur infeasible and unnecessary computing resource requirements. One commonly used abstraction technique is to model several events as a simultaneous happening. For example, in figure 1, we have a system of an *SAM (surface-to-air missile)* and multiple hostile jet-fighters. There are two events: the '`hit`' of the missile on a jet-fighter and the observations of the '`explosion`' of the hapless jet-fighter by other jet-fighters. On

*The work is partially supported by grant NSC 97-2221-E-002-129-MY3 from NSC, Taiwan, ROC and by a research grant from Research Center for Information Technology Innovation, Academia Sinica, Taiwan, ROC in 2010. A preliminary version of the manuscript is in the Proceedings of the 7th ICFEM (International Conference on Formal Engineering Methods), Nov. 2005, Manchester, UK., LNCS 3785, Springer-Verlag.

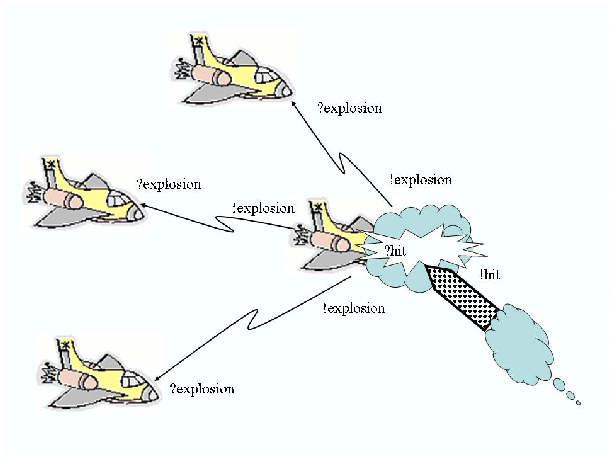


Figure 1: A system with one missile and 4 jet-fighters

one hand, the '`hit`' event is an interaction between the missile and the hapless jet-fighter. On the other hand, right after the '`hit`' event, the '`explosion`' event is broadcast to all the remaining enemy jet-fighters and may affect their actions afterward. In most verification tasks, what happens in the split-second between the '`hit`' event and the '`explosion`' event does not matter. To the missile launcher, what matters is when it can start tracking the next target. To the remaining jet-fighters, what matters is their reaction after the observation. Thus, it is only natural to model the '`hit`' event and the '`explosion`' event as happening simultaneously. Modeling them as two separate events only unnecessarily adds to the verification complexity and does not help engineers in analyzing the behavior of the system.

One language device designed to model simultaneous actions in two processes is the *channel* (i.e., *event* in this work) concept for binary synchronization [13]. Conceptually, the device glues two local transitions[1] from two different processes to model a *global transition*.[2] Such a device can greatly help to improve the modularity of model descriptions. In the above-mentioned SAM example, the action that a missile hits a jet-fighter can be modeled with an event named `hit` between the missile and the hapless jet-fighter. The language device `!hit` represents the sending (or output) *synchronization operation (sync-op)* by the missile through the event while `?hit` represents the receiving (or input) sync-op by the hapless jet-fighter through the same event. Two local transitions labeled respectively with the input sync-op and output sync-op through the same event must happen at the same instant to make a global transition. Modeling such a global transition as two synchronized local transitions can greatly enhance the modularity in model construction.

We use an extension of binary synchronizations for flexibility in model construction. In our extension, we allows for sync-ops like $!\sigma@q$ where q is a place holder for the identifier of the process that responds to this sync-op. The place-holders can then be used for the expressive description of the corresponding local transition.

Example 1 *local transition rules with quantified place-holders* When the SAM hits a jet-fighter, the SAM may want

[1] A *local transition* models the observation of a global state-change from a process in a concurrent system.
[2] A *global transition* models a global state-change and could be the simultaneous interaction of several local transitions.

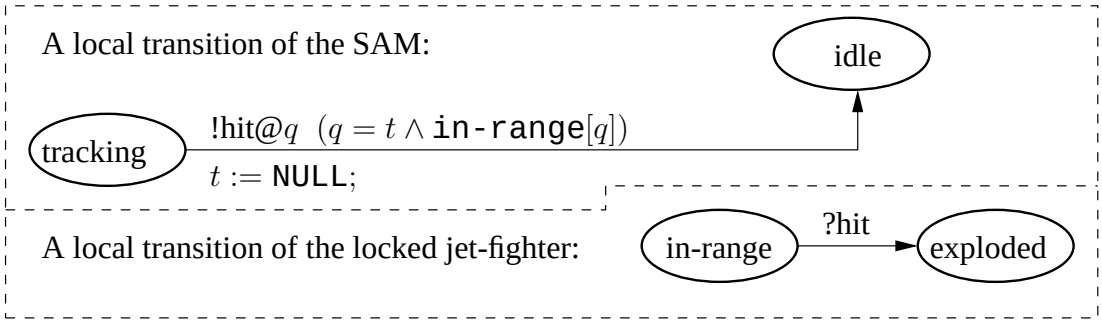
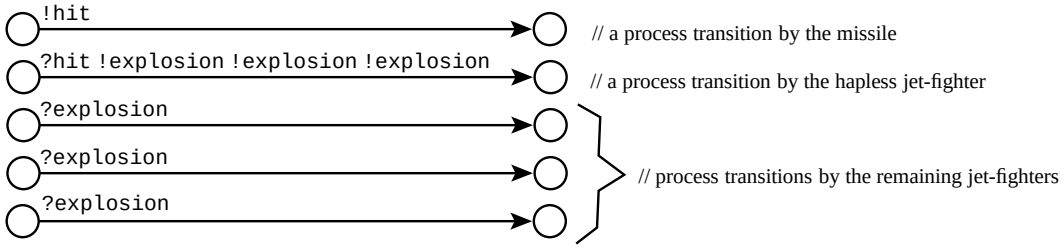


Figure 2: Synchronization of the locking event

to make sure that the fighter is the target and still in the range of the missile. Suppose t is a variable that records the identity of the target. We may have the transition diagrams in figure 2 for this example synchronization. The ovals represent operation modes of the processes. The arcs represent local transition rules. The SAM moves from the `tracking` mode to the `idle` mode while the hapless jet-fighter from the `in-range` mode to the `exploded` mode. On the arcs, we label sync-ops, triggering conditions, and actions. Place-holder q represents the identifier of the process responding to sync-op `!hit`. The hitting local transition rule of the SAM is labeled with sync-op "`!hit`@q" and triggering condition "$q = t \wedge$ `in-range`$[q]$." ■

Although the above-mentioned devices are good for binary synchronizations between two parties, we can use them to construct complex actions, e.g., broadcasting events, out of many simultaneous process transitions. For example, in figure 1, the event that the missile hits one jet-fighter can cause an explosion observed by the other three jet-fighters. By constructing the following 5 local transition rules,



we can glue the five local transition rules to make a global transition that models the simultaneity of the `hit` event and the observations of the `explosion` event. Such a scheme allows for concise and highly abstract models in which the interaction in the split second between the hitting and observation of the explosion is of no concern to the verification engineers. In addition, the scheme also allows for the modular description of each party involved in a complex synchronization. A jet-fighter can be modeled without the knowledge of how many peers are involved in the synchronization. In section 3, we have an example that shows the expressiveness of the scheme.

The above-mentioned idea of using binary synchronizations to construct complex interactions among many processes, although plausible, has not been supported by many verification tools [4, 22]. In this article, we investigate the various issues involved in implementing the idea for the model-checking of dense-time systems [2, 3] with binary and broadcasting synchronizations [13]. There are two challenges involved in the implementation. In the following, we explain the two challenges and our solutions for them.

- **Challenge I:** "*How do we generate the set of global transitions ?*" Intuitively, we can first record, for each event type σ, the local transition rules with $?\sigma$ and those with $!\sigma$. Then we use this recording and some heuristics to enumerate the set of global transitions. However, this enumeration can be expensive because in practice the number of elements in this enumeration can be of exponential complexity to the input size. The reason is that a particular sync-op (say $!\sigma$) can be labeled on many local transition rules of the same process. For example, suppose that when an `explosion` happens, a jet-fighter can observe the event in one of two modes: `cruise` and `evasive maneuver` with different triggering conditions. Thus, we need to construct two local transition rules respectively from these two modes to receive the `explosion` event for each jet-fighter process. In other words, a surviving jet-fighter can execute either of these two local transition rules, one from the `cruise` mode and the other from the `evasive maneuver` mode, to receive an `explosion` event. If there are m such surviving jet-fighters, there are 2^m such synchronization combinations. Thus, even just to enumerate all the synchronization combinations may have become infeasible when large numbers of jet-fighters are involved. One contribution of the article is the presentation of an algorithm that takes advantage of the data-sharing capability of BDD-like diagrams [7, 8, 19, 20] to calculate the symbolic characterizations of global transitions.
- **Challenge II:** "*How do we efficiently evaluate the weakest precondition of global transitions when the number of such global transitions is exponential to the input size ?*" Another contribution of this article is the presentation of an algorithm that uses the data-sharing capability of BDD-like diagrams to effectively avoid the explicit enumeration of all global transitions in the evaluation of weakest preconditions of discrete transitions.

We have endeavored to implement the ideas and carried out experiments to see how our ideas work in practice. Specifically, we report the following three experiments.

- *Experiment for model description complexity.* It is also possible to use traditional techniques to model multi-party complex synchronizations [4, 6, 14, 17, 22]. We show how our techniques can reduce model description complexity for better project management.
- *Experiment for the threshold of global transition enumeration.* We have found that for global transitions with simple synchronizations, weakest precondition calculation with direct enumeration of such global transitions can be more efficient. However, for global transitions with complex synchronizations involving many processes, calculation that takes advantage of data-sharing capability of BDD-like diagrams results in higher efficiency. With the finding, we also propose a hybrid scheme that uses a threshold for the number of processes involved in a global transition to decide which way we should use for the weakest precondition calculation through the global transition.
- *Experiment for performance comparison with UPPAAL.* We want to see how our techniques compare with existing techniques. We choose to compare with the popular UPPAAL [4]. We use their techniques reported in [1, 15] to construct models with multi-party synchronizations. Our techniques show performance advantage over UPPAAL against all the benchmarks in the experiment.

The experiments show that our techniques could really be useful in the verification of real-world projects. The tool, **RED** 8.0, and benchmarks can be downloaded for free at project REDLIB of SourceForge.

2 Related work

Broadcasting semantics is used in modeling language Esterel [5] for synchronous systems. One fundamental semantical feature of Esterel is the *synchrony hypothesis* that may incur back-and-forth communication in the transient state before the synchronous system moves to the next state. Our broadcasting model is defined with dense-time automata and allows flexible abstraction for both synchronous and asynchronous communication modeling. Specifically, the broadcasting of events of Esterel models can either be modeled as global variable read/write operations or as our broadcasting synchronizers. For example, if there are m processes in an Esterel model. Then a broadcasting synchronization signal emitted from a process can be modeled as $m-1$ pairs of binary synchronizations. A receiving process that is not awaiting the signal can be modeled with a dummy transition that receives the signal and does nothing. Moreover, our techniques can in fact be used to support the efficient precondition and postcondition calculation.

It is possible to model complex synchronizations with previous techniques [4,6,14,17,22]. For example, UPPAAL [4] and Kronos [22] also support the broadcasting synchroniztions. However, the transitions of the correspondents in their broadcasting synchronizations are subject to various assumptions and cannot be used with the flexibility proposed in this work. For example, we may use symbols, including P (the indentifier of the executing process of a transition) and $\#PS$ (the number of processes), and congregate notations, including quantified process indentifiers for predicates, to construct concise model templates that can be reused for different processes. With such modeling devices, it is possible to use our tool to construct concise templates that can be shared by different verification projects.

Even if there is no direct support for modeling broadcasting events, there are effort in using traditional language concepts to make such models. For example, there were projects with UPPAAL and Spin [1, 15] in using piecewise binary sychronizations to model a broadcasting synchronizations. The idea was to break each complex broadcasting event into a sequence of transitions by the corresponding processes. Such an approach has the following drawbacks.

- The models of the complex synchronizations then would be unnatural. Especially, the atomicity of the broadcasting events is lost and the users may not be able to model the system in the abstraction level that is useful, natural, and efficient for verification. Then the users are forced to model each complex synchronization as a permutation of events. In [**?**, 1], a pre-selected permutation is used. In general, it can be difficult to argue whether the ignored permutations do not lead to faulty behaviors. As a result, such a selection is not only arguably awkward but may also leave the result of model-checking less than convincing. Moreover, the models constructed with traditional concepts are much more complex to manage. In subsection 6.3, we have detailed comparison of complexity of model descriptions constructed with both traditional devices and with our proposal.
- Since atomicity is lost, we may need some auxiliary mechanism in the models to make sure the sequence is properly ordered, not interfered with or disrupted. Such auxiliary mechanisms have not been formally proven correct and may create unnecessary explosion in reasoning with such mechanisms. This should be clear from our experiment reported in subsection 6.5.

In Petri nets [16], a transition may move many tokens from places to places. Since tokens usually represent processes, such modeling structures may also be used to model complex synchronizations. However, in our modeling language, we allow a process to correspond to a complex synchronization in different modes (control locations) with

different guards. Due to the flexibility of our language, such complex synchronizations can be described with model templates in a very concise way. In contrast, such models may need exponentially many transitions that represent the combinations of all corresponding transitions from all processes with Petri nets. Such a bulky model can also incur unnecessary state explosion.

3 Safety analysis of communicating timed rule systems

We use a variation, called *communicating timed rule system (CTRS)*, of the widely accepted model of *timed automata* [3] to describe the transitions in dense-time state-spaces. A CTRS has a finite set of discrete variables and a finite set of clocks which can hold nonnegative real-values. In its operation, several local transition rules can synchronize and be simultaneously triggered when their corresponding triggering conditions are satisfied. Upon being triggered, the CTRS instantaneously transits by resetting some clocks to zero and changing some discrete variables' values. In between transitions, all clocks increase their readings at a uniform rate.

Let $\mathbb{N}$ be the set of non-negative integers and $\mathbb{R}^{\geq 0}$ be the set of non-negative reals. Given a sequence with elements $a_1, \ldots, a_n$ in sequence, we denote the sequence as $[a_1 a_2 \ldots a_n]$. Given a (partial) function g on $\{a_1, \ldots, a_n\}$, we denote the function as $\{a_1 \leftarrow g(a_1), \ldots, a_n \leftarrow g(a_n)\}$. Let $[\,]$ be a function undefined on everything. Given two partial functions g and g', we let $g \circ g'$ be the function identical to g except that for each x defined in g', $g \circ g'(x) = g'(x)$.

3.1 Syntax of CTRS

Given a system of m processes, we use integers $1, \ldots, m$ to respectively identify the m processes. We also use `NULL` to denote zero. Each process in our system models may have a set of local variables. For $1 \leq p \leq m$, to access a local variable named 'y' of process p, we may simply write '$y[p]$.' While in the scope of execution of process p, we may use the *default shorthand* 'y' for '$y[p]$.' If the process identifier of a peer process is stored in local variable q, then we can also use '$y[q]$' to access the local variable y of the peer process.

Formally speaking, a sync-op of event set Σ and process identifier place-holder set Q can be viewed as a triple $\langle d, \sigma, q \rangle$ with the following restrictions.

- d is the direction of the synchronization. $d =$'!' means it is an output sync-op. $d =$'?' means it is an input sync-op.
- σ is an event name in Σ.
- $q \in Q$ is a place-holder for the identifier of the process that responds to this sync-op. A place-holder is quantified over the execution scope of one local transition rule. Thus the same place-holder name can be used in many local transition rules without ambiguity in the interpretation of the values of the place-holders.

For convenience, we shall write $d\sigma@q$ in place of $\langle d, \sigma, q \rangle$ from now on. When the place-holder of the identifier of the responding process is not used, we may write $d\sigma$ (and $\langle d, \sigma, \rangle$) for simplicity.

Example 2 *Synchronizations with quantified place-holders* Suppose that we have processes 1 and 2 synchronizing with sync-ops $\langle ?, \texttt{hit}, q \rangle$ and $\langle !, \texttt{hit}, q \rangle$ respectively. In the scope of the synchronization, the former q and the latter

q are interpreted respectively as 2 and 1. ■

We let *SOSEQ*(Σ, Q) be the set of all finite sequences of sync-ops of Σ and Q. A variable q is declared in a sequence $[s_1 s_2 \ldots s_n] \in$ *SOSEQ*(Σ, Q) if for some $1 \leq i \leq n$, $s_i = \langle d, \sigma, q \rangle$ for some d and $\sigma \in \Sigma$.

Given a set L of local discrete variable names, a set X of local clock names, and a set Q of process identifier place-holder names, we use *VAR*(L, X, Q) for the following variable reference name set $L \cup X \cup \{y[q] \mid y \in L \cup X, q \in L \cup Q\}$.

We use *LP*(L, X, Q) as the set of all Boolean combinations of atoms of the form $y \sim c$ where $y \in Q \cup$ *VAR*(L, X, Q), '$\sim$' is one of $\leq, <, =, >, \geq$, and $c \in Q \cup \mathbb{N}$. An element in *LP*(L, X, Q) is called a *local predicate* of L, X, and Q. For example, $x \leq 5 \wedge y[q] = 3 \wedge x[t] > 2$ is a local predicate in *LP*$(\{t\}, \{x, y\}, \{q\})$.

A local predicate may contain references to local variables whose identities are interpreted with respect to the executing process. For example, $x \leq 5$ refers to a local variable x. When the inequality is used in the execution of a rule by process 5, then x is to be interpreted as $x[5]$. For convenience, we define *inst*(η, p) as the instantiation of local predicate η with respect to process p. Inductively, it is defined as follows.

- *inst*$(\eta_1 \vee \eta_2, p) \equiv$ *inst*$(\eta_1, p) \vee$ *inst*(η_2, p)
- *inst*$(\neg\eta_1, p) \equiv \neg$*inst*$(\eta_1, p)$
- *inst*$(y \sim c, p) \equiv y[p] \sim c$
- for every $q \in L \cup Q$, *inst*$(y[q] \sim c, p) \equiv y[q[p]] \sim c$
- for every $1 \leq p' \leq m$, *inst*$(y[p'] \sim c, p) \equiv y[p'] \sim c$.

We define $\eta_1 \wedge \eta_2$ and $\eta_1 \rightarrow \eta_2$ respectively as the shorthands for $\neg((\neg\eta_1) \vee (\neg\eta_2))$ and $(\neg\eta_1) \vee \eta_2$. *inst*(η, p) falls in the class of *instantiated local predicates*. For example,

$$\textit{inst}(x \leq 5 \wedge y[q] = 3 \wedge x[t] > 2, 5) \equiv x[5] \leq 5 \wedge y[q[5]] = 3 \wedge x[t[5]] > 2$$

An *assignment* of L, X, and Q is "$y := c$;" with $y \in$ *VAR*(L, X, Q) and $c \in \mathbb{N} \cup Q$. We use *ASEQ*(L, X, Q) for the set of all assignment sequences of L, X, and Q.

Definition 3 **process timed rule system templates (PTRST)** A *PTRST* A is given as a tuple $\langle \Sigma, L, X, Q, E, \epsilon, \tau, \pi \rangle$ with the following restrictions. Σ is a finite set of event names. L is a finite set of local discrete state variable names while X is a finite set of local clocks. Q is the finite set of local place-holder names for process identifiers. For simplicity of presentation, we require that $Q \cap L = \emptyset$. E is a finite set of *local transition rules* (*rules* for short). $\epsilon : E \mapsto$ *SOSEQ*(Σ, Q) defines the sequence of sync-ops of each rule. $\tau : E \mapsto$ *LP*(L, X, Q) defines the triggering condition of each rule. $\pi : E \mapsto$ *ASEQ*(L, X, Q) defines the assignment sequence to local variables of each rule. For convenience, we require that there is a null rule $\perp \in E$ such that $\epsilon(\perp) = [\,]$, $\tau(\perp) =$ *true*, and $\pi(\perp) = [\,]$. ■

To specify the global states of a system, we define *SP*(L, X, m) as the set of all Boolean combinations of atoms of the form $y[p] \sim c$ where $y \in L \cup X$, $1 \leq p \leq m$, '$\sim$' is one of $\leq, <, =, >, \geq$, and $c \in \mathbb{N}$. An element in *SP*(L, X, m) is called a *state predicate* of L and X with respect to concurrency m. For example, $x[5] \leq 5 \wedge y[1] = 3 \wedge x[3] > 2$ is a state-predicate.

Definition 4 **communicating timed rule systems (CTRS)** A *CTRS* M is a tuple $\langle A, m, I, V \rangle$, where A is a PTRST

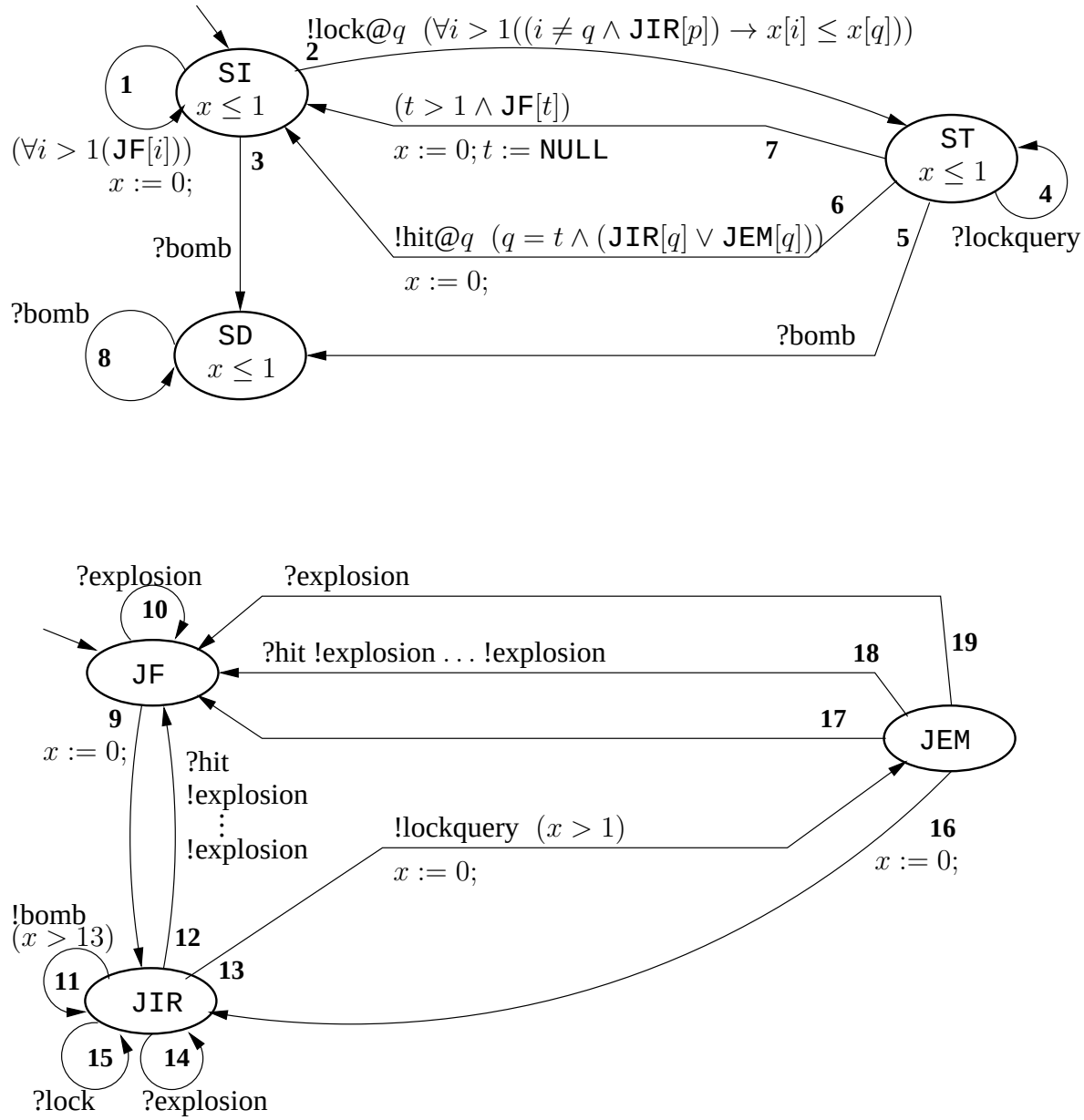


Figure 3: Model templates of fighters and SAM

$\langle \Sigma, L, X, Q, E, \epsilon, \tau, \pi \rangle$, m is number of processes, and $I \in SP(L, X, m)$ is the global *initial condition*. $V \in SP(L, X, m)$ is the global *invariance condition*. The *concurrency* (or number of processes) of the system is m. ■

The general scheme of CTRS allows for the modeling of broadcasting and multicasting of many generic transmission events.

Example 5 *Fighters and SAM system* In figure 3, we show our PTRST for the system of fighters and SAM described in figure 1. We use integer 1 for the identifier of the SAM system while integers $2, \ldots, m$ for those of the $m - 1$ jet-fighters. The connected graph in the upper-half represents the transition-diagram of the SAM while the one in the lower-half represents that of the fighters. The SAM has three operation modes: `SI`(SAM idle), `ST`(SAM tracking),

and SD(SAM destroyed). The SAM starts it execution form mode SI. The fighters have three operation modes: JF(jet-fighter faraway), JIR(jet-fighter in range), and JEM(jet-fighter in evasive maneuver). The fighters start their execution from mode JF. The SAM and the fighters interact through events lock, lockquery, bomb, and hit. The fighters also communicate with one another through event explosion to model the observation of the explosion at the hitting of a missile.

x is a local clock name. t is another local integer variable name that records the identifier of the fighter that has been locked by the SAM. There is also an implicit integer variable name mode that represents the operation modes of each process.

Note that the triggering condition of transition **2** is a quantified predicate. Without loss of generality, we use this as a shorthand for the disjunction of the instantiated quantified subformulas. In fact, our implementation does accept such quantified predicates.

The initial condition of the CTRS is $\texttt{mode}[1] = \texttt{SI} \wedge x[1] = 0 \wedge \bigwedge_{1<p\leq m}(\texttt{mode}[p] = \texttt{JF} \wedge x[p] = 0)$. The invariance condition is $(\texttt{mode}[1] = \texttt{SI} \vee \texttt{mode}[1] = \texttt{ST} \vee \texttt{mode}[1] = \texttt{SD}) \wedge x[1] \leq 1 \wedge \bigwedge_{1<p\leq m}(\texttt{mode}[p] = \texttt{JF} \vee \texttt{mode}[p] = \texttt{JIR} \vee \texttt{mode}[p] = \texttt{JEM})$. The destruction of the SAM is modeled with event bomb which can happen if a jet-fighter stay in mode JIR for more than 13 time units. The SAM can take down a jet-fighter by tracking it in one time units with event hit. The SAM launcher wants to avoid being bombed.

We have also labeled each rule a bold-face number for the convenience of latter discussion. Global transitions can be flexibly constructed out of the binary synchronization events. For example, to model the bombing of the a jet-fighter, the SAM synchronizes with a jet-fighter through event bomb. This may happen between the SAM process using one rule of **3**, **5**, **8** with any jet-fighter process using rule **11**. Thus in total, there could be $3 \times (m-1)$ such global transitions for the modeling of jet-fighter bombing. But in the example, we successfully decompose the many global transitions into the modular descriptions of three SAM rules and one jet-fighter rule. As the number of jet-fighter processes increases, the benefit for conciseness in modular model construction with our scheme will become even more salient. ■

3.2 Global transitions of CTRS

A PTRST cannot execute its rules by its own. According to the semantics of binary synchronization [13], a rule can be executed if and only if all its input sync-ops have been sent out by some processes at the same time and all its output sync-ops have also been received by some processes at the same time. There are several issues in defining semantically correct global transitions. In the following, we first define *transition plans* which may not result in sensible global transitions. Then we use examples to explain what could go wrong in the definition and propose restrictions to refine the definition.

Definition 6 **transition plan (TP)** A *transition plan (TP)* of a CTRS is conceptually a function from $\{1, \ldots, m\}$ to E and prescribes the composition of a global transition. ■

A TP may not describe a consistent global transition in that some sync-ops are not responded in the global transition. We have the following example to explain the issue.

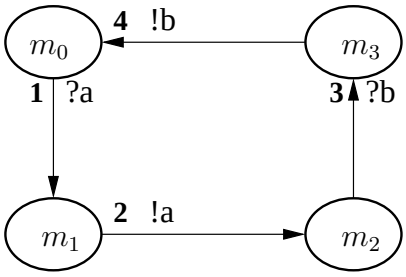


Figure 4: PRTST for explaining minimality

Intuitively, we say a TP is *consistent* if the following constraint is satisfied. For each $\sigma \in \Sigma$, the number of output sync-ops of event type σ must match the number of input sync-ops of event type σ in a TP. Let $\mathit{value}(?) = -1$ and $\mathit{value}(!) = 1$. For each $e \in E$ with $\epsilon(e) = [\langle d_1, \sigma_1, q_1\rangle \ldots \langle d_n, \sigma_n, q_n\rangle]$, we let $\mathit{ECount}_\sigma(e) = \mathit{SUM}_{1\leq i\leq n;\sigma_i=\sigma}\mathit{value}(d_i)$. For convenience, if σ is not used in $\epsilon(e)$, then we let $\mathit{ECount}_\sigma(e) = 0$. The intuition is as follows.

- If $\mathit{ECount}_\sigma(e) > 0$, there are $\mathit{ECount}_\sigma(e)$ output sync-ops of event type σ in $\epsilon(e)$.
- If $\mathit{ECount}_\sigma(e) < 0$, there are $\mathit{ECount}_\sigma(e)$ input sync-ops of event type σ in $\epsilon(e)$.
- If $\mathit{ECount}_\sigma(e) = 0$, there is no sync-ops of event type σ in $\epsilon(e)$.

The consistency constraint of TP T means that $\forall\sigma \in \Sigma(\mathit{SUM}_{1\leq p\leq m}\mathit{ECount}_\sigma(T(p)) = 0)$.

Example 7 *An inconsistent TP* We may have TP $g_1 = \{1 \leftarrow \mathbf{2}, 2 \leftarrow \mathbf{9}, 3 \leftarrow\!\perp\}$ for the CTRS in example 5. g_1 is inconsistent in that the `!lock`@q issued by process 1 with rule **2** is not responded by process 2 with rule **9**. ■

Here we have another example that motivates for the semantic constraints on TPs.

Example 8 *Compatibility with interleaving semantics* Suppose we have the PTRST in figure 4 for four processes. The TP of $T_2 = \{1 \leftarrow \mathbf{1}, 2 \leftarrow \mathbf{2}, 3 \leftarrow \mathbf{3}, 4 \leftarrow \mathbf{4}\}$ can be partitioned into two TPs $T_3 = \{1 \leftarrow \mathbf{1}, 2 \leftarrow \mathbf{2}, 3 \leftarrow\!\perp, 4 \leftarrow\!\perp\}$ and $T_4 = \{1 \leftarrow\!\perp, 2 \leftarrow\!\perp, 3 \leftarrow \mathbf{3}, 4 \leftarrow \mathbf{4}\}$. If we allows for the atomic execution of T_2, it seems somewhat incompatible with the popular interleaving semantics of concurrent systems. In this work, we need constraints on the semantics of global transitions to eliminate such transition plans. ■

The constraints for compatibility with interleaving semantics discussed in example 8 is not at all straightforward. One naive semantic definition is to disallow any global transition whose TP can be broken down to two nontrivial consistent TPs. We use the following example to show why such a naive semantics may be contradictory to an intuitive model that the users want to construct.

Example 9 *More on the compatibility with interleaving semantics* Suppose we have the PTRST in figure 5(a) for four processes. The TP of $T_5 = \{1 \leftarrow \mathbf{4}, 2 \leftarrow \mathbf{5}, 3 \leftarrow \mathbf{6}, 4 \leftarrow \mathbf{7}\}$ can actually be partitioned into two nontrivial consistent TPs $T_6 = \{1 \leftarrow \mathbf{4}, 2 \leftarrow \mathbf{5}, 3 \leftarrow\!\perp, 4 \leftarrow\!\perp\}$ and $T_7 = \{1 \leftarrow\!\perp, 2 \leftarrow\!\perp, 3 \leftarrow \mathbf{6}, 4 \leftarrow \mathbf{7}\}$. It can also be partitioned as $T_8 = \{1 \leftarrow \mathbf{4}, 2 \leftarrow\!\perp, 3 \leftarrow\!\perp, 4 \leftarrow \mathbf{7}\}$ and $T_9 = \{1 \leftarrow\!\perp, 2 \leftarrow \mathbf{5}, 3 \leftarrow \mathbf{6}, 4 \leftarrow\!\perp\}$. If we just examine the binary synchronization relation among the sync-ops, we could run into the conclusion that T_5 violates the interleaving semantics and should not be generated. But it could also happen that neither of the two decompositions in the above actually match the intention of the users. For example, the users may have actually put down constraints

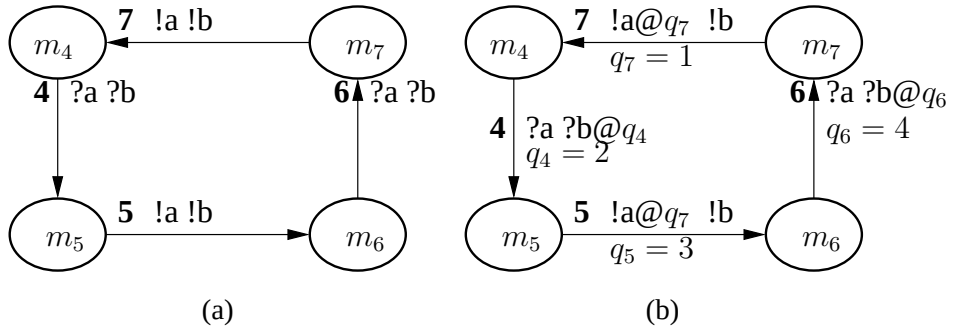


Figure 5: PRTST for explaining minimality

on the quantified process identifier place-holders and described the model as in figure 5(b). As can be seen, the users may actually want to model a circular synchronization with the four processes. We certainly do not want to rule out such synchronizations. ■

According to examples 8 and 9, we propose a *connectivity* requirement on transition functions. For convenience of the definition of the requirement, we need the following definitions. For each place-holder q in a rule executed by process p, we introduce an auxiliary local discrete variable $q[p]$. This is necessary since two rules respectively executed by two processes may contain place-holders with the same name. We assume that given a sequence λ, $|\lambda|$ is its length. Given A, we let ***slen***(A) be $\max_{e\in E}|\epsilon(e)|$. For convenience of discussion, given CTRS $\langle A, m, I, V\rangle$, we assume the following sync-ops.

$\langle d_{1,1}, \sigma_{1,1}, q_1\rangle$	$\langle d_{1,2}, \sigma_{1,2}, q_2\rangle$	$\ldots$	$\langle d_{1,\mathit{slen}(A)}, \sigma_{1,\mathit{slen}(A)}, q_{\mathit{slen}(A)}\rangle$
$\langle d_{2,1}, \sigma_{2,1}, q_1\rangle$	$\langle d_{2,2}, \sigma_{2,2}, q_2\rangle$	$\ldots$	$\langle d_{2,\mathit{slen}(A)}, \sigma_{2,\mathit{slen}(A)}, q_{\mathit{slen}(A)}\rangle$
$\vdots$	$\vdots$		$\vdots$
$\langle d_{m,1}, \sigma_{m,1}, q_1\rangle$	$\langle d_{m,2}, \sigma_{m,2}, q_2\rangle$	$\ldots$	$\langle d_{m,\mathit{slen}(A)}, \sigma_{m,\mathit{slen}(A)}, q_{\mathit{slen}(A)}\rangle$

In this array for synchronization, we use the following $m \times$ ***slen***(A) place-holders of the process identifiers.

$$\begin{array}{cccc} q_1[1] & q_2[1] & \ldots & q_{\mathit{slen}(A)}[1] \\ \vdots & \vdots & \ldots & \vdots \\ q_1[m] & q_2[m] & \ldots & q_{\mathit{slen}(A)}[m] \end{array}$$

In a global transition, according to the traditional semantics of binary synchronization in the literature [13], a binary synchronization must happen exactly between a pair of input-output sync-ops in the array. Moreover, the two sync-ops cannot participate in any other binary synchronization. To explain this precisely, we use the following matrix.

Definition 10 **synchronization plan matrix (SPM)** An SPM compatible with a transition plan T of CTRS $\langle A, m, I, V\rangle$ is an $m \times$ ***slen***(A) 2-dimensional matrix Ψ of integer pairs.

$\langle P_{1,1}, S_{1,1}\rangle$	$\langle P_{1,2}, S_{1,2}\rangle$	$\ldots$	$\langle P_{1,\mathit{slen}(A)}, S_{1,\mathit{slen}(A)}\rangle$
$\langle P_{2,1}, S_{2,1}\rangle$	$\langle P_{2,2}, S_{2,2}\rangle$	$\ldots$	$\langle P_{2,\mathit{slen}(A)}, S_{2,\mathit{slen}(A)}\rangle$
$\vdots$	$\vdots$		$\vdots$
$\langle P_{m,1}, S_{m,1}\rangle$	$\langle P_{m,2}, S_{m,2}\rangle$	$\ldots$	$\langle P_{m,\mathit{slen}(A)}, S_{m,\mathit{slen}(A)}\rangle$

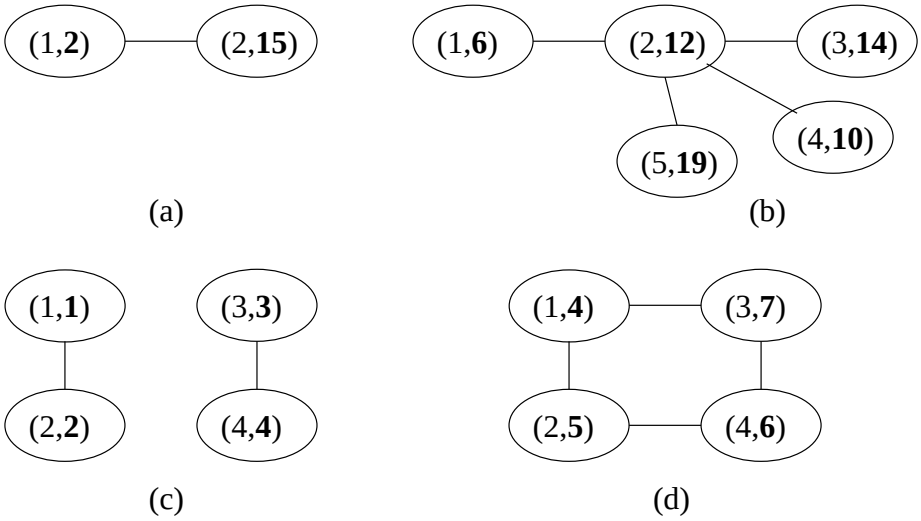


Figure 6: Synchronization trees

For each $1 \leq h \leq m$ and $1 \leq i \leq \mathit{slen}(A)$, $\Psi[h,i] = \langle P_{h,i}, S_{h,i} \rangle$ with the following restrictions. $P_{h,i}$ records the process identifier that responds to the i'th sync-op in rule $T(h)$ issued by process h. $S_{h,i}$ records the index of sync-op in $\epsilon(T(P_{h,i}))$ that responds to the i'th sync-op in rule $T(h)$ issued by process h. A basic requirement of an SPM is that if $\langle P_{h,i}, S_{h,i} \rangle = \langle k, j \rangle$, then $\langle P_{k,j}, S_{k,j} \rangle = \langle h, i \rangle$. An SPM Ψ is compatible with T if the following constraints hold.

- For any $1 \leq h \leq m$, if $T(h) = \perp$, then $\forall 1 \leq i \leq \mathit{slen}(A)(\langle P_{h,i}, S_{h,i} \rangle = \langle \perp, 0 \rangle)$.
- For any $1 \leq h \leq m$, if $T(h) \neq \perp$, then $\forall i > |\epsilon(T(h))|(\langle P_{h,i}, S_{h,i} \rangle = \langle \perp, 0 \rangle)$.
- For any $1 \leq h \leq m$ and $1 \leq i \leq \mathit{slen}(A)$, if $P_{h,i} \neq \perp$ and $S_{h,i} \neq 0$, then $\mathit{value}(d_{h,i}) + \mathit{value}(d_{P_{h,i},S_{h,i}}) = 0$ and $\sigma_{h,i} = \sigma_{P_{h,i},S_{h,i}}$. ■

Definition 11 **synchronization graph** Given an SPM Ψ compatible with a TP T, the *synchronization graph* $G(T, \Psi)$ of T and Ψ is defined as follows. The node set of $G(T, \Psi)$ is $\{(p, e) \mid 1 \leq p \leq m, T(p) = e \neq \perp\}$ while the edge set is $\{((p, e), (p', e')) \mid \exists h \exists h', (\Psi[p, h] = \langle p', h' \rangle)\}$. ■

Example 12 *Synchronization graphs of global transitions* Assume that we have a SAM system and 4 jet-fighters with models in figure 3. A synchronization graph for TP $\{1 \leftarrow \mathbf{2}, 2 \leftarrow \mathbf{15}, 3 \leftarrow \perp, 4 \leftarrow \perp, 5 \leftarrow \perp\}$ is in figure 6(a). A synchronization graph for TP $\{1 \leftarrow \mathbf{6}, 2 \leftarrow \mathbf{12}, 3 \leftarrow \mathbf{14}, 4 \leftarrow \mathbf{10}, 5 \leftarrow \mathbf{19}\}$ is in figure 6(b). The only synchronization graph for T_2 in example 8 is in figure 6(c) while the only triggerible synchronization graph for T_5 in figure 5(b) is in figure 6(d). ■

Definition 13 **global transitions** A global transition is a pair $\langle T, \Psi \rangle$ where T is a consistent TP and Ψ is an SPM such that $G(T, \Psi)$ is connected. ■

Intuitively, if $G(T, \Psi)$ is not connected, then we can break it into two unsynchronized global transitions, just like the case in figure 6(c). As can be seen, it is not possible to construct a connected synchronization graph for T_2. A

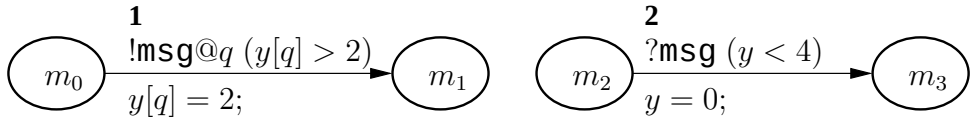


Figure 7: PRTST for explaining ITR-conditions

global transition can be executed if we know both its TP and the compatible SPM that makes the synchronization graph connected.

In general, in a system, some processes may choose not to respond to a synchronization event sent to them in a complex synchronization. With definition 13, we can model such non-responding actions with dummy correspnding transitions that do nothing. It is also easy to extend our language to incorporate such flexibilities, i.e., not all parties have to respond to a set of simultaneous binary synchronizations. However, this is actually equivalent to adding those dummy corresponding transitions in an automatic preprocessing step.

However there could be *intra-transition race-conditions* (*ITR-condition*) in global transitions. An ITR-condition between two rules happens when the order of execution of the assignments in the two rules may affect the result of the synchronization between the two rules. Depending on the intention of the users, an ITR-condition could represent an anomaly in a model that the users are not aware of.

Example 14 *Intra-transition race-conditions (ITR-condition)* Suppose we have a CTRS $\langle A, 2, I, V\rangle$ with PTRST depicted in figure 7. Suppose process 1 is to execute rule **1** while process 2 is to execute rule **2**. In this case, the $y[q]$ in rule **1** and the y in rule **2** are aliases for $y[2]$. If the assignment of rule **1** happens before that of rule **2**, $y[2] = 0$ after the synchronization. Otherwise, $y[2] = 2$ after the synchronization. This is a *write-write* ITR-condition between processes 1 and 2. ■

In a general distributed system model, there could also be read-write ITR-conditions. In our models, because the assignments are executed only after the triggering conditions of all the synchronizing rules have been satisfied and because we only assign constants to variables, only write-write ITR-conditions are possible. In the semantic definition of the CTRS, we assume that all global transitions are ITR-condition free. In subsection 4.3, we present algorithms to check and eliminate ITR-conditions from global transition characterizations.

3.3 Semantics of CTRS

Definition 15 **states** Suppose we are given a CTRS $\langle A, m, I, V\rangle$ where A is a PTRST like $\langle \Sigma, L, X, Q, E, \epsilon, \tau, \pi\rangle$. A state for the CTRS is a valuation $\nu : (\{x[p] \mid x \in X, 1 \leq p \leq m\} \mapsto \mathbb{R}^{\geq 0}) \cup (\{l[p] \mid l \in L, 1 \leq p \leq m\} \mapsto \mathbb{N})$.

An *extended state* ν with a set Q of process identifier place-holders is a state extended with function from $\{q[p] \mid q \in Q, 1 \leq p \leq m\}$ to $\{1, \ldots, m\}$. Given a state ν and a global transition $\langle T, \Psi\rangle$, the extended state $\nu\langle T, \Psi\rangle$ is identical to ν except that for each $1 \leq h \leq m$ and $1 \leq i \leq \mathit{slen}(A)$, if $\Psi[h, i] = \langle P, S\rangle$, $\nu\langle T, \Psi\rangle(q_i[h]) = P$. ■

For convenience, a state ν is also an extended state in which all process identifier place-holders are undefined. Given an extended state ν, we let $\nu^{\exists Q}$ be the state identical to ν with respect to all variables in $\{x[p] \mid x \in X, 1 \leq p \leq m\} \cup \{l[p] \mid l \in L, 1 \leq p \leq m\}$ and undefined for all other variables.

Given an instantiated local predicate or a state predicate η, we say an extended state ν satisfies η, in symbols $\nu \models \eta$, iff η is true when all its variables are interpreted according to ν. For any $\delta \in \mathbb{R}^{\geq 0}$, $\nu + \delta$ is a valuation identical to ν except that for every $x \in X$ and $1 \leq p \leq m$, $\nu(x[p]) + \delta = (\nu + \delta)(x[p])$. We define *assign*$(\nu, [y_1 := c_1; \ldots; y_n := c_n;], p)$ to be $\nu \circ \{y_1[p] \leftarrow c_1, \ldots, y[p]_n \leftarrow c_n\}$.

Definition 16 **runs** Given a CTRS $M = \langle A, m, I, V \rangle$ with $A = \langle \Sigma, X, P, Q, E, \epsilon, \tau, \pi \rangle$, a *run* is an infinite computation of M along which time diverges. Formally speaking, a run is an infinite sequence of state-time pairs $[(\nu_0, t_0)(\nu_1, t_1) \ldots (\nu_k, t_k) \ldots \ldots]$ such that

- $t_0 t_1 \ldots t_k \ldots \ldots$ is a monotonically increasing divergent real-number sequence, i.e., $\forall c \in \mathbb{N}, \exists h > 1, t_h > c$; and
- **Invariance condition:** for all $k \geq 0$ and $t \in [0, t_{k+1} - t_k]$, $\nu_k + t \models \bigwedge_{1 \leq p \leq m} V_p$; and
- **Transitions:** for all $k \geq 0$, either
  - **a time transition:** $\nu_k + (t_{k+1} - t_k) = \nu_{k+1}$; or
  - **a global transition:** There is a global transition $\langle T, \Psi \rangle$ such that $(\nu_k + t_{k+1} - t_k)\langle T, \Psi \rangle \models \bigwedge_{1 \leq p \leq m} \mathit{inst}(\tau(T(p)), p)$ and there are $\hat{\nu}_1, \ldots, \hat{\nu}_m$ such that $\hat{\nu}_1 = \mathit{assign}((\nu_k + t_{k+1} - t_k)\langle T, \Psi \rangle, \pi(T(1)), 1)$, $\hat{\nu}_2 = \mathit{assign}(\hat{\nu}_1, \pi(T(2)), 2)$, $\ldots$, $\hat{\nu}_k = \mathit{assign}(\hat{\nu}_{k-1}, \pi(T(k)), k)$, $\ldots$, $\hat{\nu}_m = \mathit{assign}(\hat{\nu}_{m-1}, \pi(T(m)), m)$, and $\nu_{k+1} = \hat{\nu}_m$. ■

For convenience of discussion, we adopt *safety analysis* as our verification framework. Given a safety predicate $\eta \in \mathit{SP}(L, X, m)$ and a run $[(\nu_1, t_1)(\nu_2, t_2) \ldots (\nu_k, t_k) \ldots \ldots]$ of a CTRS, we say the run satisfies η iff for every $k \geq 1$ and every $t \in [0, t_{k+1} - t_k]$, $\nu_k + t \models \eta$. A CTRS $\langle A, m, I \rangle$ is *safe* with respect to η iff for every run $[(\nu_1, t_1) \ldots \ldots]$ of the CTRS such that $\nu_1 \models I$, the run satisfies η. Our *safety analysis problem* asks whether a given CTRS is safe with respect to a given safety predicate.

4 Symbolic characterization of global transitions

A straightforward way to construct a symbolic characterization of all global transitions is to enumerate all the TPs, then to enumerate all SPMs compatible with the TPs, and then to check if they are both consistent and their synchronization graphs are connected. But, as we have argued in section 1, this could be infeasible in practice.

We say a process identifier place-holder is necessary in a TP if it is used in either the triggering condition or the assignment primitives of a rule in the TP. A TP augmented with information of the process identifier value used in the necessary place-holders of the rules is called an *extended TP*. To implement a model-checker with general and flexible global transitions, we do not really need to know what the global transitions in a CTRS are. Actually, for the precise execution of a global transition, we only need to know its extended TP. In the following, we first present an algorithm that construct symbolic characterizations of TPs in subsection 4.1. The characterizations are already good enough for the evaluation of weakest preconditions of systems without quantified process identifier place-holders. We then extend the characterization to that of extended TPs with values of quantified process identifier place-holders necessary in the rules in subsection 4.2. However, there could be ITR-conditions in the execution rules described in the characterizations. Finally we present the characterization of ITR-conditions, which can be used to check and

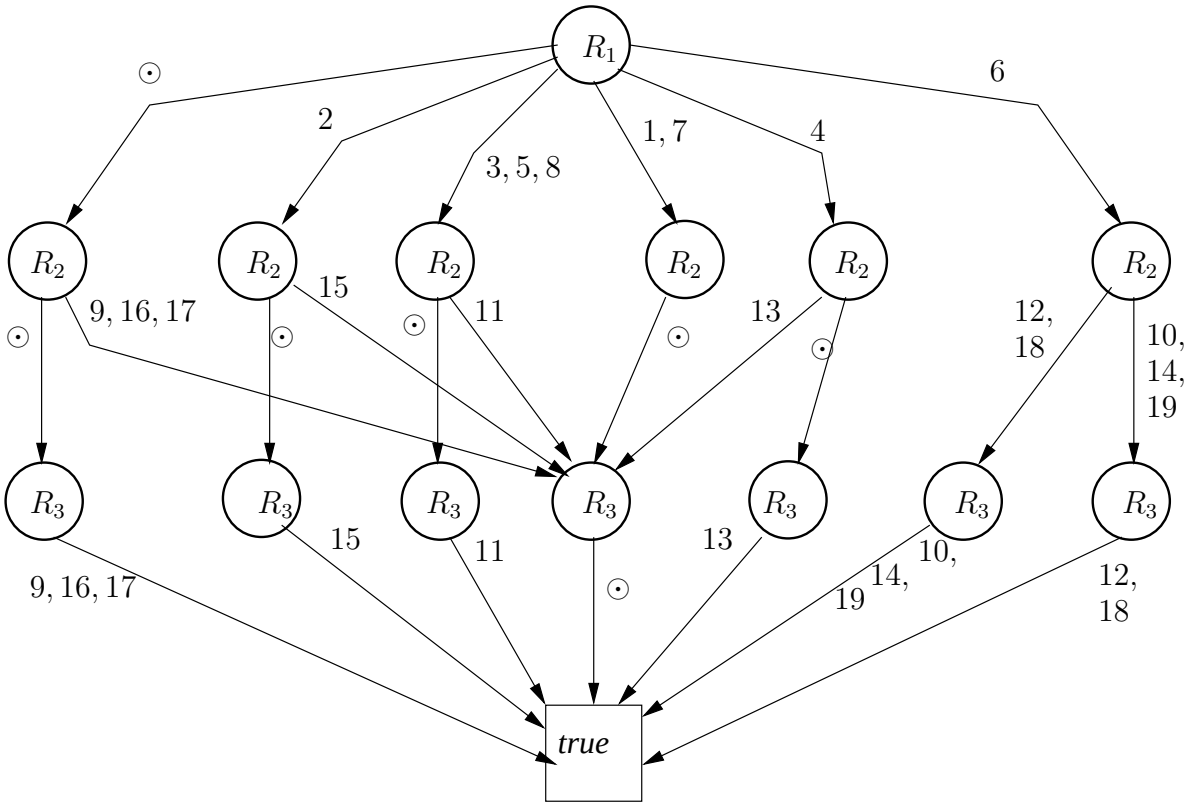


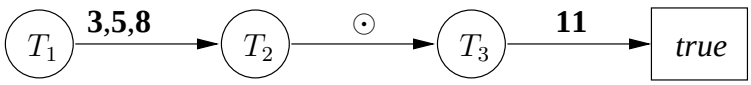
Figure 8: The symbolic characterization of transition plans for the CTRS in figure 3 with 2 fighters.

eliminate ITR-conditions in global transitions, in subsection 4.3.

4.1 Symbolic characterization of TPs

Our algorithm for the symbolic characterization of TPs is based on the consistency constraint of the TPs and the connectivity constraint on the synchronization graphs. Our algorithm utilizes the data-sharing capability of BDD-like diagrams and may very well avoid the exponential blow-up in enumerating the TPs in the average cases. For convenience of presentation, we use positive integers to index the rules. For each process p, we introduce variable T_p, which records the rule to be executed by process p in a TP. In the characterization, the label on a node together with the label on an outgoing arc specifies the participation of a process in a TP. Specifically, $\mathsf{T}_p = \perp$ if process p does not participate in the TP.

Example 17 *TP recordings in BDD-like diagrams* For the CTRS in figure 3 with two fighters, the TP characterization is in figure 8. The arc labeled with 3 from the root means that $\mathsf{T}_1 = 3$, i.e., process 1 executing rule 3. A root-to-sink path specifies TPs. The following path



represents TPs: $\{1 \leftarrow 3, 2 \leftarrow \perp, 3 \leftarrow 11\}$, $\{1 \leftarrow 5, 2 \leftarrow \perp, 3 \leftarrow 11\}$, and $\{1 \leftarrow 8, 2 \leftarrow \perp, 3 \leftarrow 11\}$. The root-to-sink paths in the diagram compose the set of all TPs. ■

Our algorithm constructs the characterizations of TPs by exploring the spanning tree construction [11] in the synchronization graphs of global transitions. The motivation is that nodes in a spanning tree are all connected. Straightforward exploration of the spanning tree with pair-wise connections between input and output sync-ops in the algorithm will be costly and inefficient. In this work, we propose a restriction that could significantly simplify our algorithm design without sacrificing much of the generality of our synchronization scheme. We require that in each local transition rule, there are no input and output sync-ops of the same event type. In other words, we forbid a process to receive an event generated by itself in the syntax. The restriction allows for a counting scheme to symbolically construct spanning trees. To explain the idea, we need the following concepts. A *partial tree* in a synchronization graph is a tree that connects some nodes in the synchronization graph.

Example 18 *Partial trees in the synchronization graphs* In the synchronization graph in figure 6(b), nodes $(2, \mathbf{12}), (3, \mathbf{14}), (4, \mathbf{10}), (5, \mathbf{19})$ constitute a partial tree in the synchronization graph. In figure 6(d), nodes $(1, \mathbf{4}), (2, \mathbf{5}), (3, \mathbf{7})$ constitute a partial tree. ■

According to definition 13, given a global transition (T, Ψ), $G(T, \Psi)$ must be connected. This implies that there is a partial tree in $G(T, \Psi)$ which contains all the nodes in $G(T, \Psi)$. Note that a partial tree may not be good enough to construct a TP or SPM since some sync-ops issued according to its nodes may not be responded by sync-ops issued according to its any other nodes. However, a partial tree can potentially be expanded to span the synchronization graphs that contain it. Now we want to propose a condition sufficient for the nodes in a partial tree to constitute a global transition. First, we want to define the following concepts. Please be reminded that the value of $\mathit{ECount}_\sigma(e)$ faithfully records the sync-op numbers of type σ only when the input and output sync-ops of type σ do not both occur in $\epsilon(e)$. We let $\mathit{ECount}(e)$ be the function of $\{\sigma \leftarrow \mathit{ECount}_\sigma(e) \mid \sigma \in \Sigma\}$. Given a partial tree P with node set $\{(p_1, e_1), \ldots, (p_k, e_k)\}$, we let $\mathit{ECount}(P) = \{\sigma \leftarrow \sum_{1 \le i \le k} \mathit{ECount}_\sigma(e_k) \mid \sigma \in \Sigma\}$. We now present the following lemma that tells us how to span synchronization graphs with iterative expansion of partial trees.

Lemma 19 *Let D be the set of nodes in synchronization graph $G(T, \Psi)$. Let P be a partial tree of $G(T, \Psi)$ and D' be the set of nodes in P. Given $(p, e) \in D - D'$, if* $\mathrm{ECount}_\sigma(e) \cdot \mathrm{ECount}(D')(\sigma) < 0$ *for some $\sigma \in \Sigma$, then there is a global transition (T', Ψ') with a partial tree P' in $G(T', \Psi')$ such that*

- *the node set of P' is $D' \cup \{(p, e)\}$ and*
- *the edge set is that of D' augmented with an edge between a node in D' and (p, e).*

Proof : Without loss of generality, we assume that the nodes in D' are $(1, e_1), \ldots, (p-1, e_{p-1})$. We assume that $\mathit{ECount}_\sigma(e) > 0$ and $\mathit{ECount}(D')(\sigma) < 0$. There are the following two cases to analyze.

- **Case I:** There are $1 \le \dot{p} < p$, $\ddot{h} > 0$, and $\dot{h} > 0$ such that $\Psi[p, \ddot{h}] = \langle \dot{p}, \dot{h} \rangle$. By adding $((p, e), (\dot{p}, e_{\dot{p}}))$ to P, we get a solution for P'. Since (p, e) is not in P, adding (p, e) to P does not create any cycles and P' is still a partial tree in $G(T, \Psi)$. The lemma is proven in this case since we can let $T' = T$ and $\Psi' = \Psi$.
- **Case II:** For all $1 \le \dot{p} < p$, $\ddot{h} > 0$, and $\dot{h} > 0$, we have $\Psi[p, \ddot{h}] \neq \langle \dot{p}, \dot{h} \rangle$. The configuration of $G(T, \Psi)$ can be visualized in a generic way as in figure9(a). The nodes in the partial tree P are all marked black and enclosed in the upper-left circle in figure 9(a). The edges in P are thick while those not in P are thin. The dashed lines represent the connectivity in the graph that are not strongly related to the presentation of the proof.

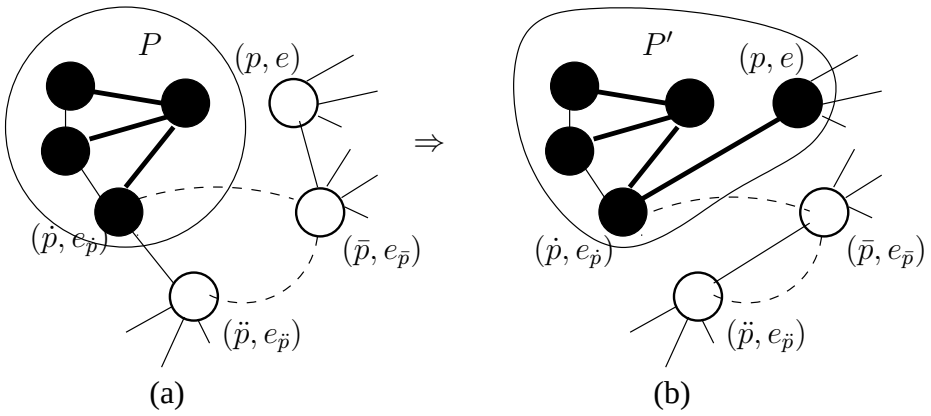


Figure 9: Construction of Ψ' from Ψ

Since $\mathit{ECount}_\sigma(e) > 0$, we may also assume that there is an $h > 0$ such that the h'th sync-op in $\epsilon(e)$ is of type $!\sigma$ and $\Psi[p, h] = \langle \bar{p}, \bar{h} \rangle$ with $\bar{p} > p$ and $\bar{h} > 0$. Furthermore, $\mathit{ECount}(D')(\sigma) < 0$ implies that there is a $1 \leq \dot{p} < p$ and $\dot{h} > 0$ such that the $\dot{h}$'th sync-op in $\epsilon(e_{\dot{p}})$ is of type $?\sigma$ and not responded by process with identifier in $[1, p-1]$. We assume that $\Psi[\dot{p}, \dot{h}] = \langle \ddot{p}, \ddot{h} \rangle$ with $\ddot{p} > p$ and $\ddot{h} > 0$.

In the following, we first construct an intermediate SPM $\bar{\Psi}$, then construct Ψ' out of $\bar{\Psi}$, and then construct T' out of Ψ' and T. We can construct $\bar{\Psi}$ identical to Ψ except that $\bar{\Psi}[p, h] = \langle \dot{p}, \dot{h} \rangle$, $\bar{\Psi}[\dot{p}, \dot{h}] = \langle p, h \rangle$, $\bar{\Psi}[\bar{p}, \bar{h}] = \langle \ddot{p}, \ddot{h} \rangle$, and $\bar{\Psi}[\ddot{p}, \ddot{h}] = \langle \bar{p}, \bar{h} \rangle$. The synchronization graph for $G(T, \bar{\Psi})$ can be viewed in figure 9(b). The nodes in partial tree P' is enclosed in the upper-left corner of figure 9(b). As can be seen from the construction, P' in $G(T, \bar{\Psi})$ can be disconnected. If it is disconnected, then one part consists of those connected to the nodes in P' while the other part consists of those connected to node $(\ddot{p}, e_{\ddot{p}})$. We define Ψ' as an SPM identical to $\bar{\Psi}$ except that for each (p', e') not connected to (p, e) in P' and $h' > 0$, let $\Psi'[p', h'] = 0$. We also define T' as a TP identical to T except that for each (p', e') not connected to (p, e) in $G(T, \bar{\Psi})$, $T'(p') = \perp$.

Now we need to check whether (T', Ψ') fits the definition of global transitions. There are two requirements on the definition, the consistency of T' and the connectivity of $G(T', \Psi')$. The latter is straightforwardly proven since we only include those nodes connected to P' in figure 9(b). The consistency requirement is proven as follows. Note that $\bar{\Psi}$ does not change the function from process identifiers to E and is still compatible with T. Since the nodes set in T could be partitioned to two mutually disconnected parts, the transition rules involved in each of this two parts satisfy the consistency requirement. Since T' is constructed out of T by removing the connected part that is not connected to P', we know that the rules in T' still satisfies the consistency requirement. Thus the lemma is also proven in this case. ■

Lemma 19 formalizes the the intuition that if a partial tree is contained by a synchronization graph, then we can find the graph by iteratively adding local transitions to match unmatched sync-ops. Thus the lemma inspires us of an algorithmic step to iteratively expand partial trees to construct global transitions. But we also need a condition to check when such an expansion ends with a global transition. We say that a partial tree P *matches* (T, Ψ) if the nodes of P is $\{(p_1, e_1), \ldots, (p_k, e_k)\}$ and $T = \{p_i \leftarrow e_i \mid 1 \leq i \leq k\} \cup \{p \leftarrow \perp \mid \forall 1 \leq i \leq k (p_i \neq p)\}$. Specifically, a partial tree, that matches a global transition (T, Ψ), is a full specification of TP T. The following lemma tells us when

we can tell a match is reached.

Lemma 20 *Let P be a partial tree in a synchronization graph. If* ECount(P) *maps every event type to zero, then P matches some global transitions.*

Proof : We use a constructive proof. Suppose the nodes in P are $(1, e_1), \ldots, (p, e_p)$. Suppose the global transition we construct is (T, Ψ). T is $\{i \leftarrow e_i \mid 1 \leq i \leq p\} \cup \{i \leftarrow \perp \mid i > p\}$. Since $\mathit{ECount}(P)(\sigma) = 0$ for each $\sigma \in \Sigma$, we have the following deduction.

$$\begin{array}{ll} & \mathit{SUM}_{1 \leq i \leq p} \mathit{ECount}_\sigma(e_i) = 0 \\ \Rightarrow & \mathit{SUM}_{1 \leq i \leq p} \mathit{ECount}_\sigma(e_i) + \mathit{SUM}_{p < i \leq m} \mathit{ECount}_\sigma(\perp) = 0 \\ \Rightarrow & \mathit{SUM}_{1 \leq i \leq m} \mathit{ECount}_\sigma(T(i)) = 0 \end{array}$$

The second line is true since $\epsilon(\perp)$ is a null sequence with zero sync-op count. The third line is true since T is constructed to match the pair relation specified in the nodes of P and to map every process identifier not specified in the nodes to zero. The third line proves the consistency constraint for T.

The construction of SPM Ψ is as follows. Since $\mathit{SUM}_{1 \leq p \leq m} \mathit{ECount}_\sigma(\epsilon(T(p)) = 0$ is true, for event type σ, the number of input sync-ops matches that of output ones issued from $e_1, \ldots, e_p$. Thus we can do pairwise matching between the input and output sync-ops and each sync-op will be exactly responded by one with the opposite direction. The pairwise matching is then sufficient for the construction of Ψ. The only constraint is that, for each edge $((p_1, e_{p_1}), (p_2, e_{p_2}))$ in the partial tree P, we must have this edge correspondence in Ψ. This construction finishes the proof. ■

Lemmas 19 and 20 together help us developing an algorithm to construct characterizations of consistent TPs with connected synchronization graphs. The characterizations are like the one in figure 8. We need a function $S : \Sigma \mapsto \mathcal{Z}$ to record the numbers of standing receiving and sending requests for the signals in Σ. Our algorithm is in table 1. The algorithm basically traverses through all partial trees. What makes it efficient is that we use the set variable Δ to record the argument pairs that have already been processed. Suppose we have recursively invoked `rec-TPs`() by picking a path $(p_1, e_1), \ldots, (p_k, e_k)$ in the synchronization graph. When we invoke a particular instance of `rec-TPs`() to execute statement (b), the record for the current invocation is a triple like (S, T, ϕ). Here S is the accumulation of the standing transmission and receiving events along path $(p_1, e_1), \ldots, (p_k, e_k)$, i.e., for each $\sigma \in \Sigma$, $S(\sigma) = \sum_{1 \leq i \leq k} \mathit{ECount}_\sigma(e_i)$. T records the information of the TPs, i.e., $T = \{p_i \leftarrow e_i \mid 1 \leq i \leq k\}$. ϕ is the result BDD-like diagram for argument pair S and T. In statement (b), every time when we find out that the argument pair has been processed before, we simply return the recorded result and save the computation resources. If the argument pair has not been processed before, we calculate and record the result in either statement (c) or (d). Statement (c) takes care of the case that a consistent TP with a connected synchronization graph has been generated. Statement (d) is executed when further traversing is still needed by invoking `rec-TPs`() to explore further along the synchronization graphs. The set of TPs for CTRS M can be enumerated as the set of root-to-sink paths in `TPs`(M).

4.2 Symbolic characterizations of extended TPs

We need a procedure, `FM_elim`(η, W), that implements the Fourier-Motzkin elimination [10] of variables in W from predicate η. There could be clock variables and discrete variables in W. Geometrically, `FM_elim`(η, W) is the

$\mathtt{TP}(M)$ /* M is a CTRS $M = \langle A, m, I, V\rangle$, $A = \langle \Sigma, X, L, E, \epsilon, \tau, \pi\rangle$. */ {
  $\Delta := \emptyset$; $\phi :=$ *false*;
  for each $1 \le p \le m$ and each $e \in E$, do
    $\phi := \phi \vee (\mathsf{T}_p = e \wedge$ `rec-TPs`$(\{\sigma \leftarrow$ *ECount*$_\sigma(e) \mid \sigma \in \Sigma\}, \{p \leftarrow e\}))$;
  return ϕ;
}

`rec-TPs`(S, T) {
  if $\exists\phi((S, T, \phi) \in \Delta)$, return ϕ; ..... (b)
  if $\forall\sigma \in \Sigma(S(\sigma) = 0)$,
    { $\phi := \bigwedge_{1\le p\le m; p \text{ undefined in } T} \mathsf{T}_p = \perp$; $\Delta := \Delta \cup \{(S, T, \phi)\}$; return ϕ; } ..... (c)
  $\phi :=$ *false*; get one $\sigma \in \Sigma$ such that $S(\sigma) \neq 0$;
  for each $1 \le p \le m$ such that p is undefined in T, do {
    for each $e \in E$ such that *ECount*$_\sigma(e) \cdot S(\sigma) < 0$, do
      $\phi := \phi \vee (\mathsf{T}_p = e \wedge$ `rec-TPs`$(\{\sigma' \leftarrow S(\sigma') +$ *ECount*$_{\sigma'}(e) \mid \sigma' \in \Sigma\}, T \circ \{p \leftarrow e\}))$;
  }
  $\Delta := \Delta \cup \{(S, T, \phi)\}$; return ϕ; ..... (d)
}

Table 1: Algorithm for symbolic characterization of TPs

projection of η on the space without dimensions W. Algebraically, given $W = \{w_1, \ldots, w_h\}$, `FM_elim`$(\eta, W) = \exists w_1 \exists w_2 \ldots \exists w_h(\eta)$. The implementation of `FM_elim`() with BDD-like diagrams for dense-time spaces has been discussed in [19].

The algorithm in table 1 does not generate information of place-holder values for synchronization parties. In this subsection, we present an algorithm that is based on $\mathtt{TP}(M)$ to construct a BDD-like diagram for the extended TPs. The idea is to construct a BDD-like diagram for SPMs, then to add in information of values of place-holders, and then to eliminate information for the SPMs. Embedding all possible SPMs in a BDD-like diagram for TPs can result in bulky and less space-efficient representations. A better strategy is to only add in necessary SPM information directly in $\mathtt{TP}(M)$. With the strategy, we propose the algorithm in table 2. For the system in figure 3, the extended TP characterization is in figure 10.

4.3 Detecting and eliminating ITR-conditions

We can construct a characterization for those rules in a TP that cause ITR-conditions. We say a variable reference y (or $y[q]$) *is written in* rule e (in symbols $y \in \pi(e)$ (or $y[q] \in \pi(e)$)), if for some $c \in \mathbb{N}$, assignment $y := c$; (or $y[q] := c$;) occurs in e. A predicate that characterizes the (write-write) ITR-conditions is as follows.

$$\mathtt{race}(m, E) \equiv \bigvee_{1\le p<p'\le m} \bigvee_{e,e'\in E} \mathsf{T}_p = e \wedge \mathsf{T}_{p'} = e' \wedge \begin{pmatrix} & \bigvee_{y[q]\in\pi(e); y\in\pi(e')} q[p] = p' \\ \vee & \bigvee_{y\in\pi(e); y[q']\in\pi(e')} q'[p'] = p \\ \vee & \bigvee_{y[q]\in\pi(e); y[q']\in\pi(e')} q[p] = q'[p'] \end{pmatrix}$$

We can use the predicate to check for ITR-conditions in $\mathtt{TP}(M)$. We can also use it to eliminate ITR-conditions from $\mathtt{TP}(M)$. In the following, we assume that all such BDD-like diagrams given to us are free of ITR-conditions.

`ETPs`(M) /* M is a CTRS $M = \langle A, m, I, V\rangle$ with $A = \langle \Sigma, X, L, E, \epsilon, \tau, \pi\rangle$. */ {
  $\phi :=$ `rec-ETPs`(`TP`(M));
  for $1 \le p \le m$ and $1 \le i \le$ *slen*(A_p), $\phi := \exists P_{p,i} \exists S_{p,i} \phi$; .....(e)
  return ϕ;
}

`rec-ETPs`(D) /* $D = \bigvee_{1\le i\le n} T_p = e_i \wedge D_i$. */ {
  $\phi :=$ *false*;
  for each $1 \le i \le n$, {
    $\psi :=\ T_p = e_i \wedge$ `rec-ETPs`(D_i);
    if for some $1 \le j \le |\epsilon(e_i)|$, $\epsilon(e_i)[j] = \langle d, \sigma, q\rangle$ and q is necessary in e_i,
      $\psi := \psi \wedge$ `place-holder-restriction`(p, d, σ, j);
    $\phi := \phi \vee \psi$;
  }
  return ϕ;
}

`place-holder-restriction`(p, d, σ, j) {
  return $\bigvee_{1\le p'\le m; p'\ne p; e'\in E; 1\le k\le|\epsilon(e')|; \epsilon(e')[k]=\langle -d,\sigma,q'\rangle} \begin{pmatrix} P_{p,j} = p' \wedge S_{p,j} = k \\ \wedge \quad P_{p',k} = p \wedge S_{p',k} = j \wedge q[p] = p' \end{pmatrix}$
}

Table 2: Algorithm for symbolic characterizations of extended TPs

5 Weakest precondition calculation for backward reachability analysis

To calculate a representation of the backwardly reachable state-space, we need two basic procedures, one for the computation of weakest preconditions of backward time-progressions and the other for those of all global transitions. We call the former `time_bck`() and the latter `xplans_bck`(). `time_bck`(η) returns the weakest precondition of state description η through backward time-progress. Details about `time_bck`() can be found in [12, 19].

`xplans_bck`(η) returns the weakest precondition of states in description η through all global transitions. Its construction is based on procedure `assign_bck`$_{\langle y:=c;\rangle}(\eta, p)$ for the weakest precondition of η before assignment "$y := c$;" with $y \in$ *VAR*(L, X, Q) and $c \in Q \cup \mathbb{N}$, executed by process p. The implementation for similar procedures has been discussed in the literature [8, 12, 19]. However, in this work, y could be a local variable with a quantifier place-holder for the identifier of a peer process. We use the algorithm in table 3 to calculate the weakest precondition of assignments used in CTRS. Given a sequence $\langle y_1 := c_1; \ldots y_{n-1} := c_{n-1}; y_n := c_n;\rangle$ of assignments with $n > 1$, we let `assign_bck`$_{\langle y_1:=c_1;\ldots y_{n-1}:=c_{n-1};y_n:=c_n;\rangle}(\eta, p)$ equal to

$$\texttt{assign\_bck}_{\langle y_1:=c_1;\ldots y_{n-1}:=c_{n-1};\rangle}(\texttt{assign\_bck}_{\langle y_n:=c_n;\rangle}(\eta, p), p).$$

In subsections 5.1 and 5.2, we discuss two algorithms of `xplans_bck`(). The two algorithms both involve double-loops. The former is traditional in that it enumerates all paths (i.e. global transitions) in `ETP`(M) to calculate a weakest precondition. The latter is our innovation and takes advantage of the data-sharing capability of BDD-like diagrams to calculate the weakest precondition of all global transitions as a whole.

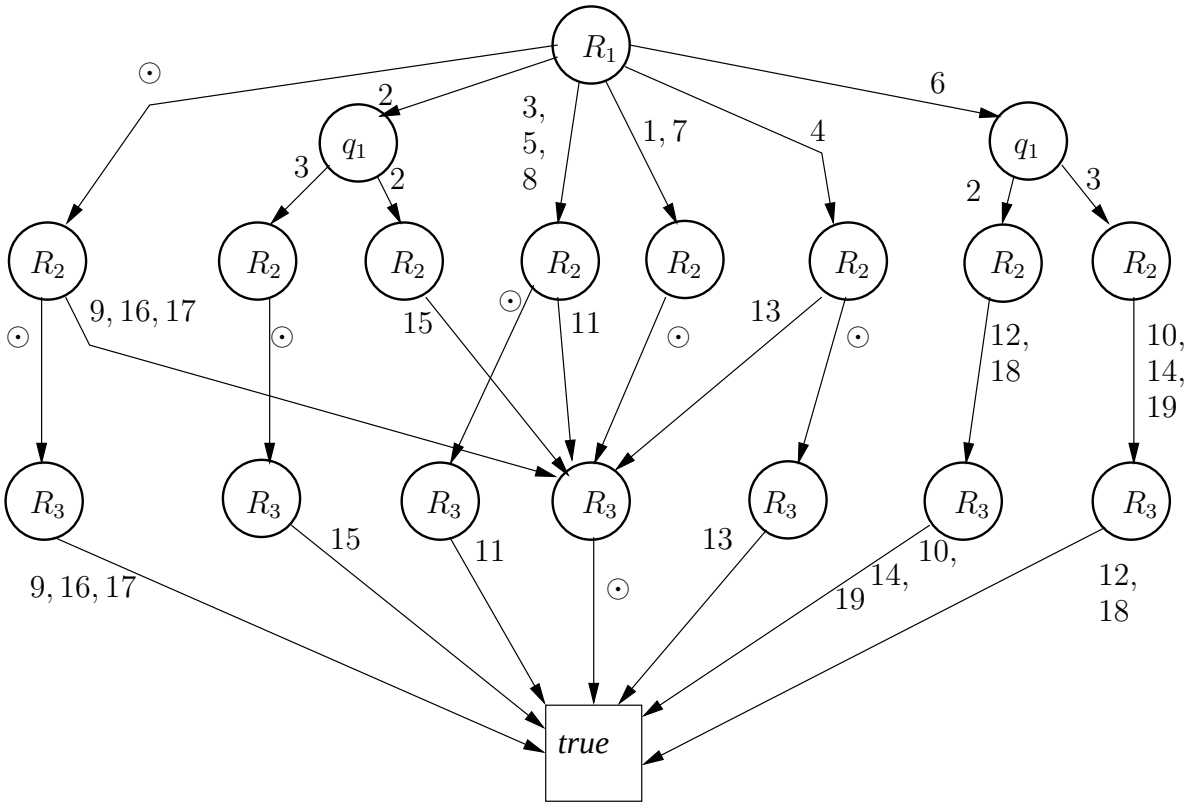


Figure 10: The symbolic characterization of extended TPs for the CTRS in figure 3

With these two basic procedures, the backward reachability procedure, denoted $\texttt{reachable\_bck}(\eta_1, \eta_2)$, which characterizes the state-space for $\exists \eta_1 \mathcal{U} \eta_2$ [12, 19], can be implemented as the least fixpoint of equation:

$$Y = \eta_2 \vee (\eta_1 \wedge \texttt{time\_bck}(\eta_1 \wedge \texttt{xplans\_bck}(Y)))$$

i.e., $\texttt{reachable\_bck}(\eta_1, \eta_2) \equiv \texttt{lfp}Y.(\eta_2 \vee (\eta_1 \wedge \texttt{time\_bck}(\eta_1 \wedge \texttt{xplans\_bck}(Y))))$. The monotonicity of F in fixpoint equation $Y = F(Y)$ and finite structures of CTRS state-space [2] together ensure the computability of the least fixpoint.

We can answer the safety of a CTRS with respect to a safety predicate η by calculating $I \wedge \texttt{reachable\_bck}(V, \neg\eta)$. If the calculation results in *false*, the system is safe. Otherwise it is unsafe.

Note that all the symbolic state-space characterizations can be represented as sets of pairs of BDDs for discrete variables and DBMs for clocks [9]. Or they can also be represented as unified BDD-like diagrams [19]. Our algorithms are independent of the representation scheme of the characterizations.

5.1 Algorithm I for `xplans_bck`()

The first algorithm for $\texttt{xplans\_bck}(\eta)$ is in table 4 with a double loop. It first enumerates all paths (i.e. extended TPs) in $\texttt{ETP}(M)$ in the outer-loop (at statement (f)) and then enumerates the rules (at statement (g)) recorded in the

`assign_bck`$_{\langle y:=c;\rangle}(\eta, p)$ /* $1 \leq p \leq m$ */ {
  switch {
  case $y \in L \cup X$:
    if $c \in \mathbb{N}$, return `FM_elim`$(\eta \wedge y[p] = c, \{y[p]\})$;
    else return `FM_elim`$(\eta \wedge y[p] = c[p], \{y[p]\})$;
  case y is $w[q]$ where $w \in L \cup X$ and $q \in Q \cup L$:
    if $c \in \mathbb{N}$, return $\bigvee_{1 \leq i \leq m}$ `FM_elim`$(\eta \wedge q[p] = i \wedge w[i] = c, \{w[i]\})$;
    else return $\bigvee_{1 \leq i \leq m}$ `FM_elim`$(\eta \wedge q[p] = i \wedge w[i] = c[p], \{w[i]\})$;
  case y is $w[p']$ where $w \in L \cup X$ and $1 \leq p' \leq m$:
    if $c \in \mathbb{N}$, return `FM_elim`$(\eta \wedge w[p'] = c, \{w[p']\})$;
    else return `FM_elim`$(\eta \wedge w[p'] = c[p], \{w[p']\})$;
  }
}

Table 3: Symbolic algorithm for the weakest precondition of assignments

`xplans_bck`(η) /* with context CTRS M */ {
  Let $\phi :=$ *false*;
  for each ETP T recorded in `ETP`(M), do { ..... (f)
    $\psi := \eta \wedge \bigwedge_{1 \leq p \leq m; q \in Q; T(q[p]) \neq \bot} q[p] = T(q[p])$; ..... (g)
    for each $1 \leq p \leq m$, if $T(p) \neq \bot$, $\psi :=$ `assign_bck`$_{\pi(T(p))}(\psi, p)$; ..... (h)
    $\phi := \phi \vee$ `FM_elim` $\Big(\psi \wedge \bigwedge_{1 \leq p \leq m}$ *inst*$(\tau(T(p)), p), \{q[p] \mid 1 \leq p \leq m; q \in Q\}\Big)$; ..... (i)
  }
  return $\phi \wedge V$; ..... (j)
}

Table 4: Traditional algorithm for precondition calculation

current extended TP in the inner-loop to calculate a weakest precondition. Statement (g) conjuncts the postcondition with the constraints on the quantified place-holders for the process identifiers used in each rule. Statement (h) calculates the precondition right before the assignments of the global transition. Statement (i) calculates the preconditions right at the satisfaction of the triggering conditions of all the participating rules and eliminates all the place-holders of the process identifiers. Finally, statement (j) makes a conjunction of the weakest preconditions of all global transitions with the invariance condition.

Algorithm I may perform badly when there are too many extended TPs. This could be the case when there are extended TPs involving many many processes and each process can participate in such plans with many rules. Suppose g is the number of global transitions. In the worst case, g can be exponential to the input sizes in bits. But when there is no such huge extended TPs, this algorithm can be efficient in that it breaks η into g small chunks to calculate weakest preconditions.

`xplans_bck`(η) /* with context CTRS M */ {
  Let $\eta := \eta \wedge \mathtt{ETP}(M)$;
  for each $1 \leq p \leq m$, do ..... (k)
    $\eta := (\eta \wedge \mathsf{T}_p = \perp) \vee \bigvee_{e \in E} \mathtt{assign\_bck}_{\pi(e)}(\eta \wedge \mathsf{T}_p = e, p)$; ..... (l)
  $\eta = \eta \wedge \bigwedge_{1 \leq p \leq m}(\mathsf{T}_p = \perp \vee \bigvee_{e \in E}(\mathsf{T}_p = e \wedge \mathit{inst}(\tau(e), p))$; ..... (m)
  return $\mathtt{FM\_elim}(\eta, \{\mathsf{T}_1, \ldots, \mathsf{T}_m\} \cup \{q[p] \mid q \in Q, 1 \leq p \leq m\})$; ..... (n)
}

Table 5: New algorithm for precondition calculation with complex synchronizations

5.2 Algorithm II for `xplans_bck`()

Algorithm II works on $\eta \wedge \mathtt{ETP}(M)$ in table 5. Algorithm II is also a double-loop (at statements (k) and (l)). Suppose after the $(p-1)$'st iteration of the outer loop, the value of ψ is ψ_{p-1}. In the p'th iteration of the outer loop, it partitions ψ_{p-1} into $|E|$ chunks. For the chunk for $\mathsf{T}_p = e$, statement (l) calculates the precondition through the assignments of rule e. After the preconditions of the assignments of all rules have been considered, statement (m) then takes into consideration the triggering conditions of all rules. Variables $\mathsf{T}_1, \ldots, \mathsf{T}_m$ and all the place-holders can then be removed with the Fourier-Motzkin elimination with statement (n).

The number of iterations of the inner loop of algorithm II is always $m \cdot |E|$. Thus, it does not suffer from the possible combinatorial explosion of global transition count as algorithm I does. But in each iteration of the outer loop, it partitions η into $|E|$ chunks to calculate weakest preconditions. Since $|E|$ is independent of the concurrency size, when the concurrency is high, such chunks can be large BDD-like diagrams with complexities exponential to the input sizes in bits. Usually the performance of BDD operations may degrade as the sizes of the BDDs increase.

5.3 A hybrid scheme to combine algorithms I and II

As mentioned in the last two subsections, algorithm I may show its drawbacks when the number of global transitions is large. And algorithm II could show its drawbacks when the number is small. We have designed a hybrid of algorithms I and II so that global transitions of few parties can be handled with algorithm I while those of many parties can be with algorithm II. Specifically, we have a run-time threshold parameter β so that global transitions of party count $\leq \beta$ will be evaluated with algorithm I and those of party count $> \beta$ will be evaluated with algorithm II. In our experiment, we have observed that usually smaller values of β, like 2, 3, or 4, engender good verification performance.

6 Implementation and experiments

6.1 Implementation

We have implemented the ideas in our model-checker/simulator [20, 21], **RED** version 8.0, for timed automata. It has been implemented with BDD-like diagram, *CRD* (Clock-Restriction Diagram) [19]. **RED** can be used with a graphical editor. It supports full TCTL model-checking and simulation-checking, both with multiple fairness assumptions. For more details, please check [18].

6.2 Design of the experiments

We report experiments with the following five benchmarks that use global transitions involving all processes.

- *CSMA/CD* [22]: This is the Ethernet bus arbitration protocol with the idea of collision-and-retry. There can be one bus process and m sender processes. Global transitions involving all processes happen when the bus process sends out collision signals to all senders at the same time. We want to verify that at any moment, at most one process is in the transmission mode for ≥ 52 time units.
- *CSMA/CA* [1, 15]: This is the variation of CSMA/CD for wireless connections. The major concern is the hidden-terminal problem. So a sender uses a signal to test if the medium is really available. If it is, then the sender starts the real transmission. The model that we use here is an adaptation of the ones in [1, 15] to take advantage of the modeling flexibility proposed in this work. There are a network process, a sender process, and m other processes that may interfere and correspond to the sender process. The timing constants used are 3 and 9. Due to the choice of the constants, it is not possible for a process to interfere the transmission once the sender is in the transmission mode. We want to check whether this property is true.
- *SAM, Missile against hostile jet-fighters*: This is the system illustrated in figure 1 and example 5. There can be a missile process and m jet-fighter processes. Global transitions involving all processes happen when a hapless jet-fighter is hit and the explosion is observed by all the other jet-fighters. The properties we want to verify is that the SAM launcher will never be bombed.
- *FIFO channel*: There are a user process, a queue manager process, and m queue slot processes. We use a local variable '`number`' of each slot process to record the position of the slot in the queue. Global transitions involving all processes happen when the queue head slot is dequeued and the other occupied slot processes must decrement their `number`s. We want to verify that the values of variables '`number`' correctly record the queue formation.
- *ARP, Ethernet address resolution protocol*: This protocol is used by a sender to get the Ethernet address (48 bits long) of the receiver in Ethernet. Initially, the sender only knows the IP address of the receiver. There are one sender process and m receiver processes. Global transitions involving all processes happen when the sender broadcasts the IP address of the receiver along the Ethernet. We want to verify that when the sender starts sending the messages, it knows the correct Ethernet address of the receiver.

The five benchmarks are all parameterized, i.e., the concurrency sizes are adjustable through the parameter m. We use parameterized benchmarks so that we can observe how our techniques scale with respect to concurrency sizes, which are a major factor for combinatorial explosion in verification complexity.

As explained in section 2, our modeling language accepts flexibility in constructing complex synchronizations with model templates. In fact, the five benchmarks used in our experiments cannot be modeled directly with the broadcasting devices, if any, supported in IF tool set [6], UPPAAL [4], Spin [14], and Kronos [22]. Since we also want to investigate whether our proposal for modeling complex synchronizations really provides any advantage over the traditional approach, we decide to compare with UPPAAL, which is the most popular model-checker for timed automatas, with the approaches in [1,15] that break a complex synchronization into a sequence of binary synchronizations. Especially, we will report the comparison in model description complexity in subsection 6.3 and verification

performance in subsection 6.5.

6.3 Model description complexities

In table 6, for each benchmark, we report the model description complexities for UPPAAL and our **RED**. Since our techniques support flexible model constructions for model template construction, all our model files are of sizes irrelevant to the sizes of concurrency. In contrast, for UPPAAL, users have to specifically construct a model for each concurrency size. This is not only inconvenient and may incur modeling errors.

Moreover, for the CSMA/CD benchmark, the UPPAAL team has implemented a program to generate the model with the input of concurrency size. We have then adapted their program code to generate models of the other four benchmarks. In table 6, we also report the sizes of such programs for the benchmarks. We found that such programming effort is also a factor to be discussed since we usually spent several days to get such a program correct. Especially, we have also to test the programs by trying different concurrency sizes and simulating the generated models. In contrast, the models with **RED** can usually be constructed and checked in several hours and are reusable for all concurrency sizes.

6.4 Experiment with synchronization enumeration depths

In table 7, please find the performance data that compares the weakest precondition algorithms presented in subsections 5.1, 5.2, and 5.3. All data are collected on an Intel Duo CPU P7450 at 2.13GHz with 2 GB RAM running Linux. In each data cell, we list a pair of the form '`time`/`memory`' where '`time`' is the running time in seconds and '`memory`' is the memory in kilobytes. Data are collected with tool `memtime` from the UPPAAL project.

For each benchmark, we collected data with $\beta = 0$ (corresponding to running Algorithm II alone), a medium value for $\beta = 3$ (corresponding to running the hybrid scheme), and the maximum value for β, i.e., the number of processes (corresponding to running Algorithm I alone). As can be seen from the data, the techniques proposed in this work really help in enhancing the verification performance of our model-checker. The medium values for β were chosen through trials and errors. We found that the medium values equal to the sizes of small global transitions usually engender either good or the best verification performance. Invariably, when the number of global transitions is big, running Algorithm I alone always ends up with the worst performance.

6.5 Comparison with UPPAAL

In table 8, please find the performance comparison data with UPPAAL version 4.0.12. All data are collected on an Intel Duo CPU P7450 at 2.13GHz with 2 GB RAM running Linux. In each data cell, we list a pair of the form '`time`/`memory`' where '`time`' is the running time in seconds and '`memory`' is the memory in kilobytes. Data are collected with tool `memtime` from the UPPAAL project.

For each benchmark, we collected data with the medium value for β (corresponding to running the hybrid scheme). As can be seen from the data, the techniques proposed in this work really exhibit performance advantage in comparison with UPPAAL against all the benchmarks. Especially, our implementation exhibits better performance advantage

Table 6: Model description complexities of UPPAAL and our tool

benchmarks	m	**RED**	UPPAAL					
			generators	#loc.	#var.	#clocks	#channels	#transitions
CSMA/CD 1 bus + m senders	4	9 mode templates, 3 variable templates, 3 clock templates, 4 channel templates, 23 transition templates	120 lines	18	3	5	7	40
	5			22	3	6	8	49
	6			26	3	7	9	58
	7			30	3	8	10	67
	8			34	3	9	11	76
	9			38	3	10	12	85
	10			42	3	11	13	94
	11			46	3	12	14	103
	12			50	3	13	15	112
	13			54	3	14	16	121
	14			58	3	15	17	130
CSMA/CA 1 medium + m senders	4	8 mode templates, 3 variable templates, 3 clock templates, 6 channel templates, 27 transition templates	157 lines	29	6	6	14	101
	5			35	6	7	17	124
	6			41	6	8	20	147
	7			47	6	9	23	170
	8			53	6	10	26	193
	9			59	6	11	29	216
	10			65	6	12	32	239
	11			71	6	13	35	262
	12			77	6	14	38	285
	13			83	6	15	41	308
	14			89	6	16	44	331
	15			95	6	17	47	354
	16			101	6	18	50	377
SAM 1 missles + m fighters	2	6 mode templates, 1 variable templates, 2 clock templates, 5 channel templates, 19 transition templates	232 lines	13	9	3	8	36
	3			24	12	4	11	57
	4			39	15	5	14	82
	5			58	18	6	17	111
FIFOM 1 user + 1 manager + m slots	4	5 mode templates, 9 variable templates, 3 clock templates, 7 channel templates, 14 transition templates	176 lines	24	17	6	9	51
	5			29	19	7	10	62
	6			34	21	8	11	73
	7			39	23	9	12	84
	8			44	25	10	13	95
	9			49	27	11	14	106
	10			54	29	12	15	117
ARP 1 network + m senders	2	8 mode templates, 2 variable templates, 2 clock templates, 4 channel templates, 17 transition templates	133 lines	14	2	3	8	33
	3			19	2	4	12	49
	4			24	2	5	16	65
	5			29	2	6	20	81
	6			34	2	7	24	97
	7			39	2	8	28	113
	8			44	2	9	32	129
	9			49	2	10	36	145
	10			54	2	11	40	161
	11			59	2	12	44	177

Table 7: Performance data of scalability w.r.t. various strategies

benchmarks	m	Enumeration depths of synchronizations		
		0	3	all
CSMA/CD 1 bus + m senders	4	0.16/163868	0.14/164264	0.16/164528
	5	0.18/165616	0.14/165748	0.26/167332
	6	0.21/168416	0.21/168020	0.62/173168
	7	0.31/173748	0.24/170448	2.17/194512
	8	0.5/181416	0.34/179040	8.67/255220
	9	0.75/195208	0.62/194020	39.25/447380
	10	1.58/230480	1.04/226784	181.36/1072968
	11	2.87/300632	2.14/295748	N/A
	12	5.27/451940	5.51/446528	
	13	14.98/780352	28.94/774016	
	14	N/A	56.76/987213	
CSMA/CA 1 medium + m senders	4	1.07/294568	0.97/290476	0.93/291400
	5	1.3/301196	1.2/294860	1.28/300140
	6	1.67/310196	1.54/301616	1.99/321944
	7	1.96/321984	1.78/311292	4.22/380856
	8	2.44/336808	2.04/325060	10.92/552152
	9	3.33/355332	2.67/344112	N/A
	10	3.86/378652	3.06/369944	
	11	4.91/408588	4.02/405420	
	12	6.3/448664	5.84/454872	
	13	8.88/506640	7.04/527108	
	14	12.36/600176	10.81/638728	
	15	27.93/760864	16.7/824100	
	16	N/A	22.93/1020936	
SAM 1 missles + m fighters	2	0.99/287952	1.54/286632	1.58/286632
	3	1.8/301032	1.72/291000	2.6/290468
	4	4.28/376248	2.26/310252	2.9/309856
	5	24.28/851848	4.32/426688	8.48/429452
FIFOM 1 user + 1 manager + m slots	4	0.6/287700	0.02/144584	0.82/287964
	5	0.61/289176	0.66/289044	0.97/291420
	6	0.7/291592	0.69/291460	1.35/301800
	7	0.77/296120	0.77/295988	2.83/334608
	8	0.85/306988	0.89/306852	8.83/443948
	9	1.36/341744	1.3/341476	31.71/810144
	10	3.51/479708	3.3/479440	N/A
ARP 1 network + m senders	2	0.1/163876	0.11/163480	0.1/163480
	3	0.17/166140	0.15/164556	0.15/164160
	4	0.28/172448	0.16/166772	0.22/168224
	5	0.61/183284	0.32/172212	0.52/180924
	6	1.37/216428	0.6/186748	1.47/215768
	7	3.13/269500	1.29/210940	5.24/335584
	8	6.61/364324	2.66/256676	21.62/763016
	9	17.49/540724	6.66/345020	N/A
	10	49.02/881572	11.35/525232	
	11	N/A	36.94/905548	

N/A means “not available.”

Table 8: Performance data of scalability w.r.t. various strategies

benchmarks	m	UPPAAL 4.0.12		**RED**	
		no speed-up	speed-up	no speed-up	speed-up
CSMA/CD 1 bus + m senders	4	0.01/3944	0/3944	0.14/164264	0.13/163868
	5	0.02/3948	0.01/3948	0.14 /165748	0.16/164032
	6	0.04/3948	0.01/21252	0.21/168020	0.22/168020
	7	0.14/21424	0.14/21420	0.24/170448	0.27/168864
	8	0.41/22296	0.37/22112	0.34/179040	0.44/179304
	9	1.23/23796	1.06/23796	0.62/194020	0.82/194020
	10	3.62/28236	3.11/28240	1.04/226784	1.53/226784
	11	10.12/39448	6.73/39444	2.14/295748	3.69/294032
	12	28.07/99060	21.53/99064	5.51/446528	8.29/446528
	13	78.73/218224	66/218228	28.94/774016	17.88/774016
	14	191.07/533028	215.96/533028	56.76/987213	51.65/970244
CSMA/CA 1 medium + m senders	4	0.02/3948	0.01/3944	0.97/290476	1.04/290476
	5	0.06/21464	0.02/21456	1.2/294860	1.27/294860
	6	0.13/22600	0.21/22464	1.54/301616	1.43/301616
	7	1.57/26736	1.64/26732	1.78/311292	1.69/311292
	8	5.88/43480	3.94/43480	2.04/325060	2.04/325060
	9	28.61/174728	36.08/174724	2.67/344112	2.54/344112
	10	164.57/648072	167.24/648068	3.06/369944	3.2/369944
	11	N/A		4.02/405420	4.1/405420
	12			5.84/454872	6.06/454872
	13			7.04/527108	7.64/527108
	14			10.81/638728	11.68/638728
	15			16.7/824100	19.55/824100
	16			22.93/1020936	N/A
SAM 1 missles + m fighters	2	0.02/21068	0.02/3224	1.54/286632	1.62/286632
	3	3.39/21484	3.39/21484	1.72/291000	1.88/291000
	4	N/A		2.26/310252	2.72/310252
	5			4.32/426688	6.64/42668
FIFOM 1 user + 1 manager + m slots	4	0.02/3948	0/3948	0.02/144584	0.56/287568
	5	0.04/21228	0/3948	0.66/289044	0.56/289044
	6	0.1/21564	0.03/21300	0.69/291460	0.66/291460
	7	0.19/24200	0.18/24336	0.77/295988	0.7/295988
	8	2.92/44916	1.88/44920	0.89/306852	0.82/306852
	9	37.38/256252	40.18/256252	1.3/341476	1.3/341476
	10	N/A		3.3/479440	2.88/479440
ARP 1 network + m senders	2	0.01/3940	0.01/3940	0.11/163480	0.12/163480
	3	0.01/3944	0.01/20980	0.15/164556	0.15/164820
	4	0.04/3944	0.02/21108	0.16/166772	0.18/167696
	5	0.04/21144	0.05/21144	0.32/172212	0.34/173400
	6	0.2/21324	0.19/21324	0.6/186748	0.65/187276
	7	0.85/21940	0.86/21932	1.29/210940	1.22/209092
	8	3.91/23160	3.56/23160	2.66/256676	2.65/256280
	9	18.52/26532	19.81/26536	6.66/345020	6.48/345020
	10	96.93/36464	102.39/36468	11.35/525232	16.24/525232
	11	533.68/65144	646.49/65136	36.94/905548	49.98/905548

N/A means "not available."

over UPPAAL against the CSMA/CA benchmark than against the other benchmarks. After checking the state representations generated with our tool, we found that one reason for this advantage was that the CSMA/CA benchmark

uses two broadcasting operations, one for checking that all other senders are hidden while another for checking that some senders may not be hidden. In contrast, the other benchmarks use only one broadcasting operation. The performance advantage may imply that our techniques scale better to the number of broadcasting operations in a model.

7 Conclusion

Successful verification of complex systems demands both an appropriate abstraction level and efficient verification techniques for such systems. Toward this end, we have made the following contributions: (1) A flexible synchronization scheme to model complex high-level behaviors in distributed real-time systems. (2) A BDD-based algorithm to calculate the characterizations of complex synchronizations among rules. (3) A BDD-based algorithm for the weakest preconditions of all global transitions calculated in a bulk. (4) A hybrid scheme to tune the performance of our techniques.

References

[1] L. Aceto, A. Bergueno, and K. G. Larsen. Model checking via reachability testing for timed automata. In B. Steffen, editor, *4th TACAS*, volume LNCS 1384, pages 263–280, 1998.

[2] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, May 1993.

[3] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[4] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Uppaal - a tool suite for automatic verification of real-time systems. In *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, volume LNCS 1055, pages 431–434. Springer-Verlag, 27-29 March 1996.

[5] G. Berry and G. Gonthier. The esterel synchronous programming language: Design, semantics, implementation. science of computer programming. 19:87–152, 1992.

[6] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. The if toolset. In *Formal Methods fo the Design of Real-Time Systems*, volume LNCS 3185. Springer-Verlag, 2004.

[7] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computer*, C-35(8), 1986.

[8] J. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. In *IEEE LICS*, 1990.

[9] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *CAV*, volume LNCS 407. Springer-Verlag, 1989.

[10] J. Fourier. In *(reported in:) Analyse des travaux de l'Académie Royale des Sciences pendant l'année 1824*. Partie Mathématique, 1827.

[11] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6(2):109–122, 1986.

[12] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:193–244, 1994. A preliminary version appeared in the Proceedings of the Seventh Annual Symposium on Logic in Computer Science (LICS), IEEE Computer Society Press, 1992, pp. 394-406.

[13] C. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[14] G. J. Holzmann. *The Spin Model Checker - Primer and Reference Manual*. Addison-Wesley, Reading Massachusetts, 2004.

[15] H. E. Jensen, K. G. Larsen, , and A. Skou. Modelling and analysis of a collision avoidance protocol using spin and uppaal. In *2nd SPIN Workshop*, August 1996.

[16] W. Reisig. *A Primer in Petri Net Design*. Springer-Verlag, 1992.

[17] V. Subramonian, C. Gill, C. Sánchez, and H. B. Simpa. Reusable models for timing and liveness analysis of middleware for distributed real-time and embedded systems. In *EMSOFT*, 2006.

[18] F. Wang. Model-checking distributed real-time systems with states, events, and multiple fairness assumptions. A preliminary version of the work appears in the proceedings of AMAST 2004, LNCS 3116, Springer-Verlag.

[19] F. Wang. Efficient verification of timed automata with bdd-like data-structures. *STTT (Software Tools for Technology Transfer)*, 6(1), 2004. special issue for the 4th VMCAI, Jan. 2003, LNCS 2575, Springer-Verlag.

[20] F. Wang. Symbolic parametric safety analysis of linear hybrid systems with bdd-like data-structures. *IEEE Transactions on Software Engineering*, 31(1):38–51, 2005. A preliminary version is in proceedings of 16th CAV, 2004, LNCS 3114, Springer-Verlag.

[21] F. Wang, G.-D. Huang, and F. Yu. Tctl inevitability analysis of dense-time systems: From theory to engineering. *IEEE Transactions on Software Engineering*, 32(7), 2006. A preliminary version of the work appears in the proceedings of 8th CIAA (Conference on Implementation and Application of Automata), July 2003, Santa Barbara, CA, USA; LNCS 2759, Springer-Verlag.

[22] S. Yovine. Kronos: A verification tool for real-time systems. *International Journal of Software Tools for Technology Transfer*, 1(1/2), October 1997.