Uniform Analysis for Communicating Timed Systems

(Extended Technical Report)

Hossein Hojjat[1], Philipp Rümmer[2], Pavle Subotic[2], and Wang Yi[2]

[1] Swiss Federal Institute of Technology Lausanne (EPFL)
[2] Uppsala University, Sweden

Abstract. Languages based on the theory of timed automata are a well established approach for modelling and analysing real-time systems, with many applications both in industrial and academic context. Model checking for timed automata has been studied extensively during the last two decades; however, even now industrial-grade model checkers are available only for few timed automata dialects (in particular Uppaal timed automata), exhibit limited scalability for systems with large discrete state space, or cannot handle parametrised systems. Leveraging recent advances of general-purpose fixed-point engines, we present a flexible method for translating networks of timed automata to Horn constraints, which can then be solved via of-the-shelf solvers. The resulting analysis method is fully symbolic and applicable to systems with large or infinite discrete state space, can be extended to include various language features, for instance Uppaal-style communication/broadcast channels and BIP-style interactions, and can analyse systems with infinite parallelism. Experiments demonstrate the feasibility of the method.

1 Introduction

With increasing complexity and ubiquity of embedded systems, verification of functional and non-functional properties is becoming ever more vital. We consider the problem of analysing properties of systems with real-time aspects, which is commonly addressed with the help of *timed automata* models. By modelling systems as timed automata, a variety of relevant properties can be analysed, including schedulability, worst-case execution time of concurrent systems, interference, as well as functional properties. Tools and model checking techniques for timed automata have been studied extensively during the last two decades, one prime example being the Uppaal tool [16], which uses difference-bound matrices (DBMs) as efficient representation of time, and explicit representation of data (discrete state). Despite many advances, scalability of tools for analysing timed automata remains a concern, in particular for models of industrial size.

We investigate the use of fully-symbolic model checking for the analysis of timed systems, leveraging counterexample-guided abstraction refinement (CEGAR) [9, 12] to represent state space, with Craig interpolation [6] for the refinement step, as well as the recently proposed framework of Horn clauses [18, 11] as intermediate system representation. Symbolic methods enable us to handle timed systems that are beyond the capabilities of DBM-based model checkers, due to the size of the discrete state space (which in realistic models can be large, or even infinite). The flexibility of Horn constraints makes it possible to elegantly encode language features of timed systems that are commonly considered difficult; in particular, systems with an unbounded (or infinite) number of processes can be handled in much the same way as bounded systems.

Contributions of the paper are: 1. a uniform analysis methodology for modelling languages providing (finite or infinite) concurrency, real-time constraints, as well as inter-process communication using shared memory, synchronous message passing, and synchronisation using barriers; instances of this framework include Uppaal timed automata [16] and BIP [4]; 2. an experimental evaluation using a set of (well-known) parametric timed automata models.

1.1 Related Work

We focus on work most closely related to ours; for a general overview of timed automata analysis, the reader is referred to surveys like [22].

Our work is inspired by recent results on the use of **Horn clauses** for concurrent system analysis, in particular Owicki-Gries and Rely-Guarantee approaches in [10]. We use Owicki-Gries-style invariants in our work, but generalise the way how invariants can relate different processes a system (using *invariant schemata*), and include systems with infinitely many processes and time. For parametric systems, we generate invariants quantifying over all processes in a system; the way such invariants are derived has similarities to [5], where quantified invariants are inferred in the context of datatypes like arrays. Encoding of timed automata as Horn clauses has also been proposed in [13, 7], but only restricted to derivation of monolithic system invariants (non-compositional reasoning).

***k*-Indexed invariants** were introduced in [21], as an instance of the general concept of **thread-modular model checking** [8], using self-reflection to automatically build environments of threads. We carry over the approach to Horn clauses, and investigate its use for extensions of timed systems.

SMT-based full model checking (***k*-induction** and **IC3**) for timed automata has recently been investigated in [15], using the region abstraction for discretisation. In comparison, our work relies on CEGAR to handle time and data alike, and achieves compositional and parametric analysis via Horn clauses.

The approach of **backward reachability** has been used for verification of various classes of parametric systems, including timed systems [3, 2, 1], establishing decision procedures with the help of suitable syntactic restrictions. A detailed comparison between backward reachability for timed systems and our approach is beyond the scope of this paper, and is planned as future work. Since our approach naturally includes abstraction through CEGAR, we expect better scalability for systems with complex process-local behaviour (e.g., if individual processes are implemented as software programs). On the other hand, backward reachability gives rise to decision procedures for important classes of parametric systems; it is unclear whether such results can be carried over to our setting.

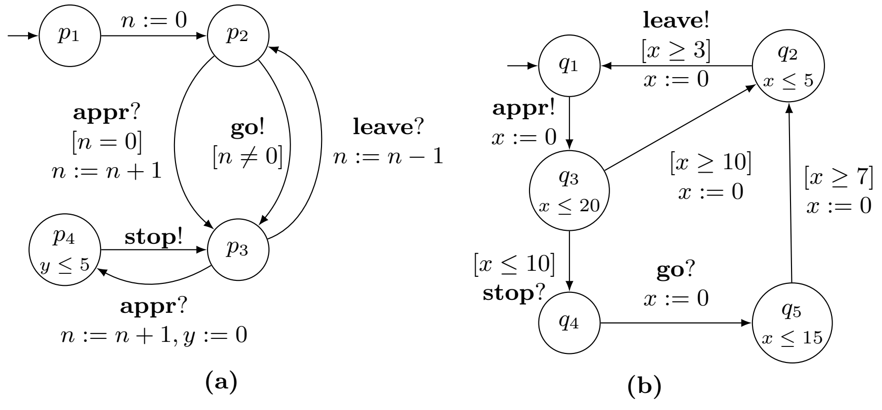


Fig. 1. Railway Control System [23]. a) Controller, b) Train.

2 Motivating Examples

We start be illustrating the scope of our method using two example models.

2.1 Railway Control System

Fig. 1 depicts a train controller system taken from [23], consisting of a number of trains travelling towards a critical point that can be passed by only one train at a time, and a controller responsible for preventing collisions. Compared to [23], the model was simplified by removing the queue to store incoming requests from the trains; since we only focus on safety, not fairness, this queue becomes irrelevant. The controller randomly releases trains without considering any specific order.

The trains communicate with the controller using binary communication channels. When a train approaches the critical point it informs the controller via the channel **appr**. It then waits for 20 time units; if the controller does not stop the train using **stop**, it enters its crossing state q_2. As a safety property of the system we require that at any time only one train can be in q_2; using Uppaal, the authors of [23] could successfully prove safety for up to 6 trains. In our setting, we consider the model with *infinitely* many instances of the train automaton; this subsumes the parametric problem of showing safety for an arbitrary (finite) number N of trains. We show safety of the infinite model (automatically) by computing a quantified inductive invariant of the form $\forall id_1, id_2, id_3.\ I(ctrl, train(id_1), train(id_2), train(id_3))$, where *ctrl* is the state of the controller, and *train*(*id*) the state of a specific train *id*; in other words, the invariant expresses a property that holds for any triplet of trains at any time. Note that invariants of this kind can express that at most two trains are in q_3.

2.2 RT-BIP Example

Figure 2 shows a temperature control system modeled in the component coordination language BIP [4]. The Controller component is responsible for keeping the

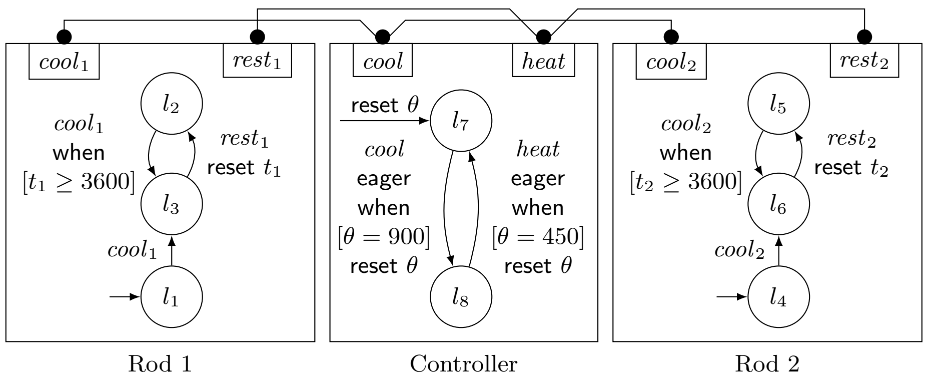


Fig. 2. Temperature Control System [4]

value of θ between the values 450 and 900. Whenever the value of θ reaches the upper bound of 900 the Controller sends a cooling command to the Rod 1 and Rod 2 components using its *cool* port. In the lower bound of 450 the Controller resets the Rods. The Rod components can accept a *cool* command again only if 3600 time units have elapsed. Compared to [4], we use the RT-BIP dialect and model physical time using clocks θ, t_1 and t_2. Note that in this model all the communications are in the form of Rendezvous so the priority layer of the BIP glue is essentially empty. The required safety property of the system is to ensure deadlock freedom. Deadlock happens when the value of θ reaches 900 but there was not sufficient time (3600 time units) for the rods to be engaged again.

BIP semantics requires that all process run to an interaction point, before interaction takes place. We model this using a basic form of synchronization barrier (Section 7.3), together with a global variable *iact* to choose between interactions. Among a set of parallel processes who share a barrier, whenever a process reaches the barrier it stops until all the rest reach the barrier. Verification of the model shows in fact this model has deadlock; the heating period of the controller is faster than the required delay time of the rods.

3 Preliminaries

Constraint languages. Throughout this paper, we assume that a first-order vocabulary of *interpreted symbols* has been fixed, consisting of a set $\mathcal{F}$ of fixed-arity function symbols, and a set $\mathcal{P}$ of fixed-arity predicate symbols. Interpretation of $\mathcal{F}$ and $\mathcal{P}$ is determined by a fixed structure (U, I), consisting of a non-empty universe U, and a mapping I that assigns to each function in $\mathcal{F}$ a set-theoretic function over U, and to each predicate in $\mathcal{P}$ a set-theoretic relation over U. As a convention, we assume the presence of an equation symbol "=" in $\mathcal{P}$, with the usual interpretation. Given a set X of variables, a *constraint language* is a set *Constr* of first-order formulae over $\mathcal{F}, \mathcal{P}, X$. For example, the language of quantifier-free Presburger arithmetic (mainly used in this paper) has $\mathcal{F} = \{+, -, 0, 1, 2, \ldots\}$ and $\mathcal{P} = \{=, \leq, |\}$), with the usual semantics.

We write $\mathit{dist}(x_1, \ldots, x_n)$ to state that the values $x_1, \ldots, x_n$ are pairwise distinct, i.e., $\mathit{dist}(x_1, \ldots, x_n) \equiv \big(\forall i, j \in \{1, \ldots, n\}.\ (i = j \vee x_i \neq x_j)\big)$.

Horn Clauses. We consider a set $\mathcal{R}$ of uninterpreted fixed-arity relation symbols. A *Horn clause* is a formula $H \leftarrow C \wedge B_1 \wedge \cdots \wedge B_n$ where

- C is a constraint over $\mathcal{F}, \mathcal{P}, X$;
- each B_i is an application $p(t_1, \ldots, t_k)$ of a relation symbol $p \in \mathcal{R}$ to first-order terms over V, C;
- H is similarly either an application $p(t_1, \ldots, t_k)$ of $p \in \mathcal{R}$ to first-order terms, or is the constraint *false*.

H is called the *head* of the clause, $C \wedge B_1 \wedge \cdots \wedge B_n$ the *body.* In case $C = \mathit{true}$, we usually leave out C and just write $H \leftarrow B_1 \wedge \cdots \wedge B_n$. First-order variables in a clause are implicitly universally quantified; relation symbols represent set-theoretic relations over the universe U of a structure $(U, I) \in \mathcal{S}$. Notions like (un)satisfiability and entailment generalise to formulae with relation symbols.

Definition 1 (Solvability). *Let $\mathcal{HC}$ be a set of Horn clauses over relation symbols $\mathcal{R}$. $\mathcal{HC}$ is called* (semantically) solvable *(in the structure (U, I)) if there is an interpretation of the relation symbols $\mathcal{R}$ as set-theoretic relations such the universal closure $Cl_\forall(h)$ of every clause $h \in \mathcal{HC}$ holds in (U, I).*

We can practically check solvability of sets of Horn clauses by means of *predicate abstraction* [10, 20], using model checkers like Z3-Horn [13] or Eldarica [14].

4 Basic Encoding of Concurrent Systems

4.1 Semantics of Concurrent Systems

We work in the context of a simple, but expressive system model with finitely or infinitely many processes executing concurrently in interleaving fashion; in subsequent sections, further features like communication will be added. Each process has its own local state (taken from a possibly infinite state space), and in addition the system as a whole also has a (possibly infinite) global state that can be accessed by all processes. We use the following notation:

- G is a non-empty set representing the global state space.
- P is a non-empty index set representing processes in the system.
- The non-empty set L_p represents the local state space of a process $p \in P$.
- $\mathit{Init}_p \subseteq G \times L_p$ is the set of initial states of a process $p \in P$.
- $(g, l) \xrightarrow{p} (g', l')$ is the transition relation of a process $p \in P$, with global states $g, g' \in G$ and local states $l, l' \in L_p$.

Given a set of processes defined in this manner, we can derive a system by means of parallel composition:

– $S = G \times \prod_{p \in P} L_p$ is the system state space. Given a system state $s = (g, \bar{l}) \in S$, we write $\bar{l}[p] \in L_p$ for the local state belonging to process $p \in P$.
– $S_0 = \{(g, \bar{l}) \mid \forall p \in P.\ (g, \bar{l}[p]) \in \mathit{Init}_p\} \subseteq S$ is the set of initial system states.
– The transition relation of the system as a whole is defined by:

$$\frac{p \in P \qquad (g, \bar{l}[p]) \xrightarrow{p} (g', l')}{(g, \bar{l}) \to (g', \bar{l}[p/l'])}$$

We write $\bar{l}[p/l'] \in \prod_{p \in P} L_p$ for the state vector obtained by updating the component belonging to process $p \in P$ to $l' \in L_p$.

Safety. We are interested in checking *safety properties* of systems as defined above. We define (un)safety in the style of coverability, by specifying a vector $(\langle p_1, E_1\rangle, \ldots, \langle p_m, E_m\rangle)$ of process-state-pairs, where the $p_i \in P$ are pairwise distinct, and $E_i \subseteq G \times L_{p_i}$ for each $i \in \{1, \ldots, m\}$. System error states are:

$$\mathit{Err} = \left\{(g, \bar{l}) \in S \mid \forall i.\ (g, \bar{l}[p_i]) \in E_i\right\}$$

Intuitively, a system state is erroneous if it contains n (pairwise distinct) processes whose state is in $E_1, \ldots, E_n$, respectively. Many common error properties, for instance occurrence of local runtime exceptions or violation of mutual exclusion, can be expressed using this concept of error. A system is *safe* if there is no sequence $s_0 \to s_1 \to \ldots \to s_n$ of transitions such that $s_0 \in S_0$ and $s_n \in \mathit{Err}$.

4.2 Encoding Safety of Finite Systems

To check that a system is safe it is sufficient to find an *inductive invariant,* which is a set $\mathit{Inv} \subseteq S$ of states with the properties

– Initiation: $S_0 \subseteq \mathit{Inv}$;
– Consecution: for all $s \to t$ with $s \in \mathit{Inv}$, also $t \in \mathit{Inv}$;
– Safety: $\mathit{Inv} \cap \mathit{Err} = \emptyset$.

The following sections define methods to derive inductive invariants with the help of Horn constraints. We first concentrate on the case of a finite set $P = \{1, 2, \ldots, n\}$ of processes, and show how an encoding in the spirit of Owicki-Gries [19] can be done with the help of Horn constraints. In comparison to earlier work [10], inductive invariants can be defined to cover individual processes, as well as sets of processes, in order to handle required relational information (inspired by the concept of *k-indexed invariants* [21]). We define this concept formally with the help of *invariants schemata.*

Recall that the component-wise order $<$ on the set $\mathbb{N}^n$ is a well-founded partial order. An *antichain* is a set $A \subseteq \mathbb{N}^n$ whose elements are pairwise $<$-incomparable; as a consequence of Dickson's lemma, antichains over $\mathbb{N}^n$ are finite. An *invariant schema* for processes $P = \{1, 2, \ldots, n\}$ is an antichain $A \subseteq \{0, 1\}^n \subseteq \mathbb{N}^n$. Intuitively, every vector in A represents an invariant to be inferred; entries with value 1 in the vector indicate processes included in the invariant, while processes with entry 0 are not visible (entries > 1 are relevant in Sect. 5.2).

$$\left\{ R_{\bar{a}}(\mathsf{g}, \mathsf{l}_1, \ldots, \mathsf{l}_k) \;\leftarrow\; \mathit{Init}_{i_1}(\mathsf{g}, \mathsf{l}_1) \wedge \cdots \wedge \mathit{Init}_{i_k}(\mathsf{g}, \mathsf{l}_k) \right\}_{\bar{a} \in A} \tag{2}$$

$$\left\{ \begin{array}{l} R_{\bar{a}}(\mathsf{g}', \bar{\mathsf{l}}[p/\mathsf{l}'][\bar{a}]) \\ \qquad \leftarrow\; \big((\mathsf{g}, \bar{\mathsf{l}}[p]) \xrightarrow{p} (\mathsf{g}', \mathsf{l}')\big) \wedge R_{\bar{a}}(\mathsf{g}, \bar{\mathsf{l}}[\bar{a}]) \wedge \mathit{Ctxt}(\{p\}, \mathsf{g}, \bar{\mathsf{l}}) \end{array} \right\}_{\substack{p=1,\ldots,n \\ \bar{a} \in A}} \tag{3}$$

$$\mathit{false} \;\leftarrow\; \Big(\bigwedge_{j=1,\ldots,m} (\mathsf{g}, \bar{\mathsf{l}}[p_j]) \in E_j \Big) \wedge \mathit{Ctxt}(\{p_1, \ldots, p_m\}, \mathsf{g}, \bar{\mathsf{l}}) \tag{4}$$

Fig. 3. Horn constraints encoding a finite system. In (2), the numbers $i_1, \ldots, i_k$ are the indexes of non-zero entries in $\bar{a}$. Symbols in $\mathsf{sans\ serif}$ are implicitly universally quantified variables.

Example 2. Consider the RT-BIP model in Sect. 2.2, with processes $P = \{1, 2, 3\}$ ($1 \cong$ Rod 1, $2 \cong$ Controller, $3 \cong$ Rod 2). Schema $A_1 = \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$ leads to fully modular safety analysis with three local invariants, each of which refers to exactly one process. $A_2 = \{(1, 1, 0), (0, 1, 1)\}$ introduces two invariants, each relating one cooling rod with the controller. The strongest invariant schema, $A_3 = \{(1, 1, 1)\}$ corresponds to analysis with a single monolithic invariant.

To define the system invariant specified by a schema A, we assume that $\{R_{\bar{a}} \mid \bar{a} \in A\}$ is a set of relation variables, later to be used as vocabulary for Horn constraints. Further, given a local state vector $\bar{l} \in \prod_{p \in R} L_p$ and $\bar{a} \in A$, we write $\bar{l}[\bar{a}]$ for the vector $(\bar{l}[i_1], \bar{l}[i_2], \ldots, \bar{l}[i_k])$, where $i_1 < i_2 < \cdots < i_k$ are the indexes of non-zero entries in $\bar{a} = (a_1, \ldots, a_n)$ (i.e., $\{i_1, \ldots, i_k\} = \{i \in \{1, 2, \ldots, n\} \mid a_i > 0\}$). The system invariant is then defined as the conjunction of the individual relation symbols $R_{\bar{a}}$, applied to global and selected local states:

$$\mathit{Inv}(g, \bar{l}) \;=\; \bigwedge_{\bar{a} \in A} R_{\bar{a}}(g, \bar{l}[\bar{a}]) \tag{1}$$

Concrete solutions for the variables $\{R_{\bar{a}} \mid \bar{a} \in A\}$, subject to the conditions *Initiation, Consecution,* and *Safety* given in the beginning of this section, can be computed by means of an encoding as Horn clauses. For this purpose, we assume that a system can be represented within some constraint language, for instance within Presburger arithmetic: the sets Init_p, the transition relation $s \xrightarrow{p} t$, as well as the error specification $(\langle p_1, E_1 \rangle, \ldots, \langle p_m, E_m \rangle)$ are encoded as constraints in this language. Horn clauses can then be formulated as shown in Fig. 3. The clauses (2) represent initiation, for each of the variables $R_{\bar{a}}$; (3) is consecution, and (4) encodes unreachability of error states.

In (3), (4), we refer to a *context invariant* $\mathit{Ctxt}(\{p_1, \ldots, p_k\}, g, \bar{l})$, which includes those literals from $\mathit{Inv}(g, \bar{l})$ relevant for the processes $p_1, \ldots, p_k$:

$$\mathit{Ctxt}(Q, g, \bar{l}) \;=\; \bigwedge \big\{ R_{\bar{c}}(g, \bar{l}[\bar{c}]) \mid \bar{c} \in A \text{ and } \exists q \in Q.\ \bar{c}[q] > 0 \big\}$$

However, note that different choices can be made concerning invariants $R_{\bar{a}}$ to be mentioned in the body of (3), (4); it is in principle possible to add arbitrary literals from $Inv(g, \bar{l})$. Adding more literals results in constraints that are weaker, and potentially easier to satisfy, but can also introduce irrelevant information.

Lemma 3 (Soundness). *If the constraints in Fig. 3 are (semantically) solvable for any invariant schema A, then the analysed system is safe.*

Proof. By checking that $Inv(g, \bar{l})$ in (1) is an inductive invariant of the system.

Lemma 4 (Completeness). *If a system is safe, then there exists an invariant schema A such that the constraints in Fig. 3 are (semantically) solvable.*

Proof. Choose $A = \{(1, 1, \ldots, 1)\}$, and interpret $R_{(1,1,\ldots,1)}$ with the set of reachable system states.

It is important to note that Lem. 4 talks about *semantic solvability.* Despite existence of such a model-theoretic solution, in general there is no guarantee that a *symbolic solution* exists that can be expressed as a formula of the chosen constraint language. However, such guarantees can be derived for individual classes of systems, for instance for the case that the considered system is a network of timed automata (and a suitable constraint language like linear arithmetic).

4.3 Counterexample-Guided Refinement of Invariant Schemata

The question remains how it is practically possible to find invariant schemata that are sufficient to find inductive invariants. This aspect can be addressed via a counterexample-guided refinement algorithm as shown in the pseudo-code of Fig. 4. Initially, verification is attempted using the weakest invariant schema, $A_0 = \{(1, 0, 0, \ldots), (0, 1, 0, \ldots), \ldots\}$. If verification is impossible, a Horn solver will produce a concrete counterexample to solvability of the generated Horn constraints. It can then be checked whether this counterexample points to genuine unsafety of the system, or just witnesses insufficiency of the invariant schema. In the latter case, a stronger invariant schema can be chosen, and verification is reattempted.

A simple criterion to identify genuine counterexamples (function *Genuine*) is to check whether the counterexample *cex* uses any clause (3) for parameters $p, \bar{a}$, such that $\bar{a}[p] = 0$. If such clauses do *not* occur in *cex*, a direct translation to a system execution leading into an error is possible.

At the moment, we use only a straightforward implementation of the *Refine* operation (shown in Fig. 4), conjoining relevant vectors in the current invariant schema according to the processes occurring in a counterexample. More sophisticated refinement algorithms are possible.

5 Safety for Unbounded Systems

5.1 Encoding of Unbounded Homogeneous Systems

We now relax the restriction that the process index set P of a system is finite, and also consider an infinite number of processes. Since our definition of safety

```
def SchemataCEGAR(S) {
  $A = \emptyset$
  for ($i \leftarrow 1$ to $|P|$) {
    $\bar{a} = (0, \dots, 0)$ ; $|\bar{a}| = |P|$ ; $\bar{a}[i] = 1$
    $A = A \cup \{\bar{a}\}$
  }
  while ($true$) {
    if ($!Solvable(S, A)$) {
      $cex = HornSolver(S, A)$
      if( $!Genuine(cex)$)
        report "ERROR"
      else A = $Refine(A, cex)$
    }
    else
      report "SAFE"
  }
}

def Refine($A$, $cex$) {
  pick $i, j \in P$ from $cex$, and $\bar{a}_1, \bar{a}_2 \in A$ with $\bar{a}_1 \neq \bar{a}_2$, $\bar{a}_1[i] = 1, \bar{a}_2[j] = 1$
  return $(A \setminus \{\bar{a}_1, \bar{a}_2\}) \cup \{\bar{a}_1 + \bar{a}_2\}$
}
```

Fig. 4. The CEGAR algorithm of Invariant Schemata

only considers finite paths into potential error states, this represents the case of systems with an unbounded number of (active) threads. Showing safety for a system with infinitely many processes raises the challenge of reasoning about a state vector with infinitely many entries. This can be addressed by exploiting the symmetry of the system, by deriving a single parametric invariant that is inductive for each process; the corresponding system invariant universally quantifies over all processes. Necessary relational information pertaining to multiple processes can be captured with the help of *k-indexed invariants* [21]. The correctness of parametric invariants can be encoded as a finite set of Horn constraints, which again yields an effective method to derive such invariants automatically.

Initially we restrict attention to *homogeneous* systems, in which all processes share the same initial states and transition relation; however, each process has access to its process id (as a natural number), and can adapt its behaviour with respect to the id.[3] We assume that $P = \mathbb{N}$, $Init_p = Init$, and $L_p = L$ for all processes $p \in P$. Then, for any number $k \in \mathbb{N}_{>0}$, and given a fresh relation variable R, a k-indexed invariant has the shape:

$$Inv(g, \bar{l}) = \forall p_1, \dots, p_k \in \mathbb{N}. \left(dist(p_1, \dots, p_k) \to R(g, p_1, \bar{l}[p_1], \dots, p_k, \bar{l}[p_k]) \right) \quad (5)$$

[3] By exploiting the fact that the id can be accessed, in fact the model in this section is as expressive as the (syntactically richer) one in Sect. 5.2.

$$\left\{ \begin{array}{l} R(\mathsf{g}, \mathsf{p}_{\sigma(1)}, \mathsf{l}_{\sigma(1)}, \ldots, \mathsf{p}_{\sigma(k)}, \mathsf{l}_{\sigma(k)}) \\ \quad \leftarrow\ \mathit{dist}(\mathsf{p}_1, \ldots, \mathsf{p}_k) \wedge R(\mathsf{g}, \mathsf{p}_1, \mathsf{l}_1, \ldots, \mathsf{p}_k, \mathsf{l}_k) \end{array} \right\}_{\sigma \in S_k} \tag{6}$$

$$R(\mathsf{g}, \mathsf{p}_1, \mathsf{l}_1, \ldots, \mathsf{p}_k, \mathsf{l}_k)\ \leftarrow\ \mathit{dist}(\mathsf{p}_1, \ldots, \mathsf{p}_k) \wedge \mathit{Init}(\mathsf{g}, \mathsf{l}_1) \wedge \cdots \wedge \mathit{Init}(\mathsf{g}, \mathsf{l}_k) \tag{7}$$

$$\begin{array}{l} R(\mathsf{g}', \mathsf{p}_1, \mathsf{l}'_1, \ldots, \mathsf{p}_k, \mathsf{l}_k) \\ \quad \leftarrow\ \mathit{dist}(\mathsf{p}_1, \ldots, \mathsf{p}_k) \wedge \big((\mathsf{g}, \mathsf{l}_1) \xrightarrow{\mathsf{p}_1} (\mathsf{g}', \mathsf{l}'_1)\big) \wedge R(\mathsf{g}, \mathsf{p}_1, \mathsf{l}_1, \ldots, \mathsf{p}_k, \mathsf{l}_k) \end{array} \tag{8}$$

$$\begin{array}{l} R(\mathsf{g}', \mathsf{p}_1, \mathsf{l}_1, \ldots, \mathsf{p}_k, \mathsf{l}_k) \\ \quad \leftarrow\ \mathit{dist}(\mathsf{p}_0, \mathsf{p}_1, \ldots, \mathsf{p}_k) \wedge \big((\mathsf{g}, \mathsf{l}_0) \xrightarrow{\mathsf{p}_0} (\mathsf{g}', \mathsf{l}'_0)\big) \wedge \mathit{RConj}(0, \ldots, k) \end{array} \tag{9}$$

$$\mathit{false}\ \leftarrow\ \mathit{dist}(\mathsf{p}_1, \ldots, \mathsf{p}_r) \wedge \Big(\bigwedge_{j=1,\ldots,m} (\mathsf{p}_j = p_j \wedge (\mathsf{g}, \mathsf{l}_j) \in E_j) \Big) \wedge \mathit{RConj}(1, \ldots, r) \tag{10}$$

Fig. 5. Horn constraints encoding a homogeneous infinite system with the help of a k-indexed invariant. S_k is the symmetric group on $\{1, \ldots, k\}$, i.e., the group of all permutations of k numbers; as an optimisation, any generating subset of S_k, for instance transpositions, can be used instead of S_k. In (10), we define $r = \max\{m, k\}$.

R represents a formula that can talk about the global state g, as well as about k pairs $(p_i, \bar{l}[p_i])$ of (pairwise distinct) process identifiers and local process states. R can therefore express which combinations of states of multiple processes can occur simultaneously, and encode properties like mutual exclusion (at most one process can be in some state at a time). For $k = 1$, the invariants reduce to Owicki-Gries-style invariants (for infinitely many processes).

Fig. 5 gives the Horn clauses encoding the assumed properties of $\mathit{Inv}(g, \bar{l})$ for a given k. Since k-indexed invariants quantify over all permutations of k processes, it can be assumed that R is symmetric, which is captured by (6). Initiation is encoded in (7). Consecution is split into two cases: (8) covers the situation that $\mathsf{p} \in \{\mathsf{p}_1, \ldots, \mathsf{p}_k\}$ makes a transition, and (9) for transitions due to some process $\mathsf{p}_0 \notin \{\mathsf{p}_1, \ldots, \mathsf{p}_k\}$. In (8), due to symmetry of R, it can be assumed that $\mathsf{p} = \mathsf{p}_1$. Unreachability of errors $(\langle p_1, E_1 \rangle, \ldots, \langle p_m, E_m \rangle)$ is specified by (10).

As shorthand notation in (9), (10), for numbers $a, b \in \mathbb{N}$ with $a \leq b$ the expression $\mathit{RConj}(a, \ldots, b)$ represents the conjunction of all R-instances for process ids in the range $a, \ldots, b$ (as in Sect. 4.2, it is possible to include further literals from $\mathit{Inv}(g, \bar{l})$ in the body of (8)–(10), resulting in weaker constraints):

$$\mathit{RConj}(a, \ldots, b) \quad = \quad \bigwedge_{\substack{i_1, \ldots, i_k \in \{a, \ldots, b\} \\ i_1 < i_2 < \cdots < i_k}} R(\mathsf{g}, \mathsf{p}_{i_1}, \mathsf{l}_{i_1}, \ldots, \mathsf{p}_{i_k}, \mathsf{l}_{i_k})$$

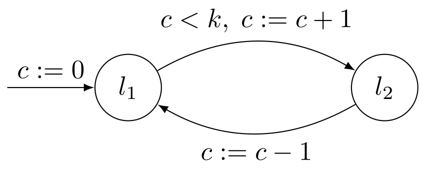
Theorem 5 (Expressiveness).

1. *If the constraints in Fig. 5 are satisfiable for a given k, then they are also satisfiable for any $k' > k$ (for the same system).*
2. *If $k' > k > 0$, then there are systems that can be verified with k'-indexed invariants, but not with k-indexed invariants.*

Proof. 1. Given a solution R_k of the constraints, a k'-solution $R_{k'}$ is:

$$R_{k'}(g, p_1, l_1, \ldots, p_{k'}, l_{k'}) \;=\; \bigwedge_{\substack{i_1,\ldots,i_k \in \{1,\ldots,k'\} \\ dist(i_1,i_2,\ldots,i_k)}} R_k(g, p_{i_1}, l_{i_1}, \ldots, p_{i_k}, l_{i_k})$$

2. Consider a system defined by infinitely many copies of the process:



where c is a global integer variable; the property to check is that no $k + 1$ processes can simultaneously reside in l_2. Absence of this error can be proven with $k + 1$-indexed invariants, but not with k-indexed invariants.

5.2 Encoding of Unbounded Heterogeneous Systems

The encodings of Sect. 4.2 and 5.1 can be combined, to analyse systems that contain n different types of processes, each of which can either be a *singleton* process, or a process that is *infinitely replicated.* Compared to Sect. 5.1, process types enable more fine-grained use of k-indexed invariants, since it is now possible to specify which processes are considered with which arity in an invariant.

More formally, we now use the process index set $P = \bigcup_{i=1,\ldots n}(\{i\} \times P_i)$, where P_i is either $\{0\}$ (singleton case) or $\mathbb{N}$ (replicated case). *Invariant schemata* from Sect. 4.2 generalise to unbounded heterogeneous systems, and are now antichains A of the partially ordered set $\prod_{i=1,\ldots n}(\{0,1\} \cup P_i) \subseteq \mathbb{N}^n$. This means that an invariant can refer to at most one instance of a singleton process, but to multiple instances of replicated processes. As in Sect. 4.2, we use a set $\{R_{\bar{a}} \mid \bar{a} \in A\}$ of relation variables, and define the system invariant as a conjunction of individual invariants, each of which now quantifies over ids of processes. Namely, for a vector $\bar{a} = (a_1, \ldots, a_n)$ and process type $i \in \{1, \ldots, n\}$, a_i distinct processes $p^i_1, \ldots, p^i_{a_i} \in P_i$ are considered:[4]

$$Inv(g, \bar{l}) \;=\; \bigwedge_{\substack{\bar{a} \in A \\ \bar{a}=(a_1,\ldots,a_n)}} \begin{array}{l} \forall p^1_1, \ldots, p^1_{a_1} \in P_1. \ldots \forall p^n_1, \ldots, p^n_{a_n} \in P_n. \\ \quad \big(dist(p^1_1, \ldots, p^1_{a_1}) \wedge \cdots \wedge dist(p^n_1, \ldots, p^n_{a_n}) \\ \qquad \to R_{\bar{a}}(g, p^1_1, \bar{l}[(1, p^1_1)], p^1_2, \bar{l}[(1, p^1_2)], \ldots, p^n_{a_n}, \bar{l}[(n, p^n_{a_n})])\big) \end{array}$$

It is then possible to formulate Horn constraints about the required properties of the invariants. The Horn clauses combine features of those in Fig. 3 and 5, but are left out from this paper due to the notational complexity.

[4] Note that if i is a singleton process, there is only a single process id ($P_i = \{0\}$), so that the corresponding argument of $R_{\bar{a}}$ could be left out.

Example 6. Consider the railway control system in Fig. 1, which consists of a singleton process $P_1 = \{0\}$, the controller, and an infinitely replicated process $P_2 = \mathbb{N}$, the trains. The system can be verified with the schema $A = \{(1, 3)\}$; this means, an inductive invariant is derived that relates the controller with a triplet of (arbitrary, but distinct) trains.

6 Encoding of Physical Time

We now describe how our model of execution, and the encoding as Horn constraints, can be extended to take physical time into account. In this and the following sections we focus on the Horn encoding of unbounded homogeneous systems (Sect. 5.1), but stress that the same extensions are possible for heterogeneous systems (Sect. 5.2) and finite systems (Sect. 4.2).

In our system model, time is represented as a component of the global state $g \in G$. As a convention, we write $g[C]$ to access the current time, and $g' = g[C/C']$ to update time to a new value $C' \in \mathit{Time}$, where $\mathit{Time} = \mathbb{Q}$ for a dense model of time, and $\mathit{Time} = \mathbb{Z}$ for discrete time. Time elapse is represented by an additional rule, augmenting the transition relation as defined in Sect. 4.1:

$$\frac{C' \in \mathit{Time} \quad C' \geq g[C] \quad \forall p \in P.\ (g[C/C'], \bar{l}[p]) \in \mathit{TInv}_p}{(g, \bar{l}) \to (g[C/C'], \bar{l})}\ \textsf{time-elapse}$$

The premises state that time can only develop monotonically, and only as long as the *time invariant* $\mathit{TInv}_p \subseteq G \times L_p$ of all processes $p \in P$ is satisfied. We make the assumption that TInv_p is convex with respect to time, i.e., $(g[C/C_1], l) \in \mathit{TInv}_p$ and $(g[C/C_2], l) \in \mathit{TInv}_p$ imply $(g[C/C_3], l) \in \mathit{TInv}_p$ for all $C_1 \leq C_3 \leq C_2$.

Concepts like clocks or stopwatches can easily be represented by defining local process transitions. For instance, a clock is realised by means of a *Time*-valued variable x; resetting the clock is translated to the assignment $x := g[C]$, so that the value of the clock at any point is $g[C] - x$.

When the model of time is dense it is also possible to retain the global variable C in the integer domain. Considering the value of the time to be a fractional number the variable C can store the numerator of the time value. We add a new global variable to the system as the dominator U which is an arbitrary positive value. The value of U stays constant throughout the system execution. The numerator C is then incremented by the time elapse transitions. Positiveness is expressed by the constraint $U > 0$ that is part of all clauses. The encoding of rationals using numerator and denominator makes it possible to keep all variables integer-valued. In practice this is helpful when no rational solver is available.

Horn constraints. On the level of inductive invariants, time just requires to add one further clause to the constraints in Fig. 5; as before, this necessitates sets TInv_p that can be represented in the constraint language of the clauses.

$$\begin{aligned} &R(\mathsf{g}[C/\mathsf{C}'], \mathsf{p}_1, \mathsf{l}_1, \ldots, \mathsf{p}_k, \mathsf{l}_k) \\ &\quad \leftarrow \; \mathit{dist}(\mathsf{p}_1, \ldots, \mathsf{p}_k) \wedge (\mathsf{C}' \geq \mathsf{g}[C]) \wedge R(\mathsf{g}, \mathsf{p}_1, \mathsf{l}_1, \ldots, \mathsf{p}_k, \mathsf{l}_k) \wedge \\ &\qquad (\mathsf{g}[C/\mathsf{C}'], \mathsf{l}_1) \in \mathit{TInv}_{\mathsf{p}_1} \wedge \cdots \wedge (\mathsf{g}[C/\mathsf{C}'], \mathsf{l}_k) \in \mathit{TInv}_{\mathsf{p}_k} \end{aligned} \tag{11}$$

7 Communication and Synchronisation

At this point, our model of execution supports communication between processes via the global state of a system (shared variables). To naturally represent timed automata models and message passing communication, it is appropriate to introduce further communication primitives, together with their encoding as Horn constraints, which is done in the next sections.

7.1 Uppaal-style Binary Communication Channels

Binary communication channels in Uppaal implement a simple form of synchronisation between pairs of processes (rendezvous). We assume that Ch is a finite set of channel identifiers. In addition to local transitions $(g, l) \xrightarrow{p} (g', l')$ of a process $p \in P$ (as in Sect. 4.1), we then also consider *send* transitions $(g, l) \xrightarrow{p,\, a!} (g', l')$ and *receive* transitions $(g, l) \xrightarrow{p,\, a?} (g', l')$ for any communication channel $a \in Ch$. Send and receive transitions are paired up in system transitions:

$$\frac{(g, \bar{l}[p_1]) \xrightarrow{p_1,\, a!} (g', l_1') \quad (g', \bar{l}[p_2]) \xrightarrow{p_2,\, a?} (g'', l_2') \quad p_1 \neq p_2 \quad a \in Ch}{(g, \bar{l}) \to (g'', \bar{l}[p_1/l_1'][p_2/l_2'])} \;\textsf{binary-comm}$$

Note that the effect of the send transition (on global state) occurs prior to the execution of the receive transition; this means that transfer of data can easily be realised with the help of additional global variables.

Horn constraints. Recall that Fig. 5 contains two clauses, (8) and (9), that model local process transitions. Since communication through a channel implies that two process transitions take place simultaneously (say, for processes $p_s, p_r \in P$), it is now necessary to distinguish four cases (and add four clauses) to characterise how a k-indexed invariant about processes $Q_k = \{p_1, \ldots, p_k\} \subseteq P$ is affected: clause (12) for the case $\{p_s, p_r\} \subseteq Q_k$ (this case disappears for $k = 1$); clause (13) for $p_s \in Q_k$, but $p_r \notin Q_k$; clause (14) for $p_r \in Q_k$, but $p_s \notin Q_k$; and clause (15) for $p_s, p_r \notin Q_k$. The clauses are instantiated for every channel $a \in \mathit{Ch}$:

$$\begin{aligned}
&R(\mathsf{g}'', \mathsf{p}_1, \mathsf{l}'_1, \mathsf{p}_2, \mathsf{l}'_2, \ldots, \mathsf{p}_k, \mathsf{l}_k) \\
&\quad \leftarrow \; \mathit{dist}(\mathsf{p}_1, \ldots, \mathsf{p}_k) \wedge \big((\mathsf{g}, \mathsf{l}_1) \xrightarrow{\mathsf{p}_1,\, a!} (\mathsf{g}', \mathsf{l}'_1)\big) \wedge \big((\mathsf{g}', \mathsf{l}_2) \xrightarrow{\mathsf{p}_2,\, a?} (\mathsf{g}'', \mathsf{l}'_2)\big) \wedge \\
&\qquad\; R(\mathsf{g}, \mathsf{p}_1, \mathsf{l}_1, \mathsf{p}_2, \mathsf{l}_2, \ldots, \mathsf{p}_k, \mathsf{l}_k)
\end{aligned} \tag{12}$$

$$\begin{aligned}
&R(\mathsf{g}'', \mathsf{p}_1, \mathsf{l}'_1, \mathsf{p}_2, \mathsf{l}_2, \ldots, \mathsf{p}_k, \mathsf{l}_k) \\
&\quad \leftarrow \; \mathit{dist}(\mathsf{p}_0, \ldots, \mathsf{p}_k) \wedge \big((\mathsf{g}, \mathsf{l}_1) \xrightarrow{\mathsf{p}_1,\, a!} (\mathsf{g}', \mathsf{l}'_1)\big) \wedge \big((\mathsf{g}', \mathsf{l}_0) \xrightarrow{\mathsf{p}_0,\, a?} (\mathsf{g}'', \mathsf{l}'_0)\big) \wedge \\
&\qquad\; \mathit{RConj}(0, \ldots, k)
\end{aligned} \tag{13}$$

$$\begin{aligned}
&R(\mathsf{g}'', \mathsf{p}_1, \mathsf{l}'_1, \mathsf{p}_2, \mathsf{l}_2, \ldots, \mathsf{p}_k, \mathsf{l}_k) \\
&\quad \leftarrow \; \mathit{dist}(\mathsf{p}_0, \ldots, \mathsf{p}_k) \wedge \big((\mathsf{g}, \mathsf{l}_0) \xrightarrow{\mathsf{p}_0,\, a!} (\mathsf{g}', \mathsf{l}'_0)\big) \wedge \big((\mathsf{g}', \mathsf{l}_1) \xrightarrow{\mathsf{p}_1,\, a?} (\mathsf{g}'', \mathsf{l}'_1)\big) \wedge \\
&\qquad\; \mathit{RConj}(0, \ldots, k)
\end{aligned} \tag{14}$$

$$\begin{aligned}
&R(\mathsf{g}'', \mathsf{p}_3, \mathsf{l}_3, \mathsf{p}_4, \mathsf{l}_4, \ldots, \mathsf{p}_{k+2}, \mathsf{l}_{k+2}) \\
&\quad \leftarrow \; \mathit{dist}(\mathsf{p}_1, \ldots, \mathsf{p}_{k+2}) \wedge \big((\mathsf{g}, \mathsf{l}_1) \xrightarrow{\mathsf{p}_1,\, a!} (\mathsf{g}', \mathsf{l}'_1)\big) \wedge \big((\mathsf{g}', \mathsf{l}_2) \xrightarrow{\mathsf{p}_2,\, a?} (\mathsf{g}'', \mathsf{l}'_2)\big) \wedge \\
&\qquad\; \mathit{RConj}(1, \ldots, k+2)
\end{aligned} \tag{15}$$

7.2 Unbounded Barrier Synchronisation

Besides rendezvous between pairs of processes, also barrier synchronisation involving an unbounded number of processes can be represented naturally in our model. Barriers turn out to be a powerful primitive to encode other forms of communication, among others Uppaal-style broadcast channels and BIP-style interactions (Sect. 7.3); of course, barriers are also highly relevant for analysing concurrent software programs. We assume a finite set Ba of barriers, and denote process transitions synchronising at barrier $b \in Ba$ by $(g, l) \xrightarrow{p,\, b} (g', l')$. For simplicity, we require *all* processes in a system to participate in every barrier synchronisation; a more fine-grained definition of the scope of a barrier can be achieved by adding neutral transitions $(g, l) \xrightarrow{p,\, b} (g, l)$ to those processes that are not supposed to be affected by b.

Barriers give rise to the following system transition:

$$\frac{\left\{\; (g, \bar{l}[p]) \xrightarrow{p,\, b} (g'_p, \bar{l}'[p]) \;\right\}_{p \in P} \qquad b \in Ba}{(g, \bar{l}) \to (g, \bar{l}')} \;\textsf{barrier}$$

Note that the system transition does not modify global state, but alters all local state components simultaneously; the motivation for this definition is to avoid potential clashes resulting from an unbounded number of global state updates.

Horn constraints. Barrier synchronisation can be represented by a simple Horn constraint (instantiated for every barrier $b \in Ba$) stating that all processes considered by a k-indexed invariant can do a transition simultaneously:

$$\begin{aligned}
&R(\mathsf{g}, \mathsf{p}_1, \mathsf{l}'_1, \mathsf{p}_2, \mathsf{l}'_2, \ldots, \mathsf{p}_k, \mathsf{l}'_k) \\
&\quad \leftarrow \; \mathit{dist}(\mathsf{p}_1, \ldots, \mathsf{p}_k) \wedge \big((\mathsf{g}, \mathsf{l}_1) \xrightarrow{\mathsf{p}_1, b} (\mathsf{g}'_1, \mathsf{l}'_1)\big) \wedge \cdots \wedge \big((\mathsf{g}, \mathsf{l}_k) \xrightarrow{\mathsf{p}_k, b} (\mathsf{g}'_k, \mathsf{l}'_k)\big) \wedge \\
&\qquad\; R(\mathsf{g}, \mathsf{p}_1, \mathsf{l}_1, \mathsf{p}_2, \mathsf{l}_2, \ldots, \mathsf{p}_k, \mathsf{l}_k)
\end{aligned} \tag{16}$$

7.3 BIP Interactions

BIP (Behavior, Interaction, Priority) [4] is a framework for designing component-based systems. The BIP model of a component consists of an interface (a set of ports) and a behavior (an automaton with transitions labeled by ports). Components are composed by a set of connectors that determine the interaction pattern among the components. In general, when several interactions are possible the system chooses the one which is maximal according to some given strict partial order (priority). For sake of presentation, we concentrate on a special case of interactions, *rendezvous,* for which priorities are irrelevant; the interactions in Fig. 2 are all in the form of rendezvous. However, we stress that other forms of interaction provided by BIP (including interaction governed by priorities, and ports that act as triggers) can be handled in our framework as well.

To define BIP rendezvous, we assume that $Port$ is a finite set of ports, and $I \subseteq \mathcal{P}(Port)$ is a set of *interactions.* A transition of a process $p \in P$ interacting through port $a \in Port$ is denoted by $(g, l) \xrightarrow{p,\,a} (g', l')$. The system transition for an interaction $\{a_1, \ldots, a_m\} \in I$ (with m distinct ports) is defined by the following rule; as a premise of the rule, it is required that all processes of the system arrived at a point where local (non-interacting) transitions are disabled ($(g, \bar{l}[p]) \overset{p}{\not\to}$), but m distinct processes $p_1, \ldots, p_m$ are available that offer interaction through ports $a_1, \ldots, a_m$, respectively:

$$\frac{\begin{array}{cc} \{\ (g, \bar{l}[p_j]) \xrightarrow{p_j,\,a_j} (g'_j, l'_j)\ \}_{j=1,\ldots,m} & \{a_1, \ldots, a_m\} \in I \\ \{\ (g, \bar{l}[p]) \overset{p}{\not\to}\ \}_{p \in P} & dist(p_1, \ldots, p_m) \end{array}}{(g, \bar{l}) \to (g, \bar{l}'[p_1/l'_1] \cdots [p_m/l'_m])}\ \textsf{bip-comm}$$

For the purpose of analysis within our framework, we reduce BIP rendezvous to barrier synchronisation as in Sect. 7.2. We make two simplifying assumptions: 1. in no state (g, l) of a process $p \in P$ both local transitions ($(g, l) \xrightarrow{p} \cdots$) and interacting transitions ($(g, l) \xrightarrow{p,\,a} \cdots$) are enabled, and 2. no two processes $p_1, p_2 \in P$ share the same port $a \in Port$. Both assumptions can be established through suitable transformations of a system.

We then encode BIP interaction using a single barrier $\{b\} = Ba$. To distinguish interactions, we choose a bijection $h : I \to \{1, \ldots, |I|\}$ that provides a unique integer as label for each interaction, and add a global variable $iact$ (part of the global state $g \in G$) ranging over $\{1, \ldots, |I|\}$. In addition, we denote the ports used by a process $p \in P$ by $Port_p \subseteq Port$. Each process $p \in P$ of the system is modified as follows:

- each interacting transition $(g, l) \xrightarrow{p,\,a} (g', l')$ is replaced with two barrier transitions. The first barrier transition is $(g, l) \xrightarrow{p,\,b} (g', l')$, and guarded with the test $a \in h^{-1}(iact)$. The second transition is $(g, l) \xrightarrow{p,\,b} (g, l)$, i.e., does not cause state changes, and is guarded with $Port_p \cap h^{-1}(iact) = \emptyset$.
- a transition $(g, l) \xrightarrow{p} (g[iact/*], l)$ non-deterministically assigning a value to the variable $iact$ is added to the process p; this transition is always enabled.

8 Experimental Evaluation

We have integrated our technique into the predicate abstraction-based model checker Eldarica [14], which uses Horn clauses to represent different kinds of verification problems including networks of timed automata. In Table 6 we show results for benchmarks[5] encoding timed models, verifying natural safety properties of the models. All model but the temperature control system (Sect. 2.2) are unbounded; finite instances of which are commonly used as benchmarks for model checkers. Most of the benchmarks were originally specified as Uppaal timed automata. For each benchmark we provide a correct version and an unsafe version, to demonstrate the ability of our tool to prove correctness and provide counter-examples for incorrect benchmarks. The Fischer benchmarks contain an observer process, which is the second component of the invariant schema.

The results demonstrate the feasibility of our approach. Further, it can be seen that the majority of the benchmarks require stronger invariant schemata, highlighting the benefits of k-indexed invariants. All benchmarks are solved within a few seconds, with the exception of the Lynch-Shavit [17] protocol.

Benchmark	$\sharp Cl_i$	$\sharp Cl_f$	N^{th} *(sec)*	*Inv Schema*	*Total (sec)*
Temperature Control System (unsafe)	48	110	0.37	(1,1,1)	3.86
Temperature Control System	48	110	1.12	(1,1,1)	4.31
Fischer	47	221	5.62	(2,1)	12.21
Fischer (unsafe)	47	221	2.84	(2,1)	8.91
CSMA/CD	50	162	3.60	(2,1)	8.22
CSMA/CD (unsafe)	50	793	2.91	(4,1)	13.81
Lynch-Shavit	50	299	66.11	(2)	70.10
Lynch-Shavit (unsafe)	50	299	2.58	(2)	5.35
Train Crossing	28	686	2.51	(1,3)	8.84
Train Crossing (unsafe)	28	240	1.53	(1,2)	3.91

Fig. 6. Runtime for verifying the benchmarks. Experiments were done on an Intel Core i7 Duo 2.9 GHz with 8GB of RAM. Columns $\sharp Cl_i$ and $\sharp Cl_f$ indicate the number of clauses required to model the corresponding benchmark for the initial iteration and final iteration of the counterexample-guided refinement of invariant schemata (Sect. 4.3), respectively. The N^{th} column indicates the time required to verify the benchmark on the final iteration. The *Inv Schema* column contains the invariant schema required for the final (successful) iteration and *Total* is the full verification time required.

References

1. A. Carioni, S. Ghilardi, S.R.: MCMT in the land of parameterized timed automata. In: VERIFY-2010 (2010)
2. Abdulla, P.A., Deneux, J., Mahata, P.: Multi-clock timed networks. In: LICS. pp. 345–354. IEEE Computer Society (2004)

[5] Available at: http://lara.epfl.ch/w/horn-parametric-benchmarks

3. Abdulla, P.A., Jonsson, B.: Model checking of systems with many identical timed processes. Theor. Comput. Sci. 290(1), 241–264 (2003)
4. Bensalem, S., Bozga, M., Sifakis, J., Nguyen, T.H.: Compositional verification for component-based systems and application. In: ATVA (2008)
5. Bjørner, N., McMillan, K.L., Rybalchenko, A.: On solving universally quantified horn clauses. In: SAS (2013)
6. Craig, W.: Linear reasoning. A new form of the Herbrand-Gentzen theorem. The Journal of Symbolic Logic 22(3), 250–268 (September 1957)
7. Fietzke, A., Weidenbach, C.: Superposition as a decision procedure for timed automata. Mathematics in Computer Science 6(4), 409–425 (2012)
8. Flanagan, C., Qadeer, S.: Thread-modular model checking. In: SPIN. pp. 213–224 (2003)
9. Graf, S., Saidi, H.: Construction of abstract state graphs with PVS. In: CAV. pp. 72–83 (1997)
10. Grebenshchikov, S., Lopes, N.P., Popeea, C., Rybalchenko, A.: Synthesizing software verifiers from proof rules. In: PLDI. pp. 405–416 (2012)
11. Gupta, A., Popeea, C., Rybalchenko, A.: Predicate abstraction and refinement for verifying multi-threaded programs. In: POPL (2011)
12. Henzinger, T.A., Jhala, R., Majumdar, R., McMillan, K.L.: Abstractions from proofs. In: 31st POPL (2004)
13. Hoder, K., Bjørner, N.: Generalized property directed reachability. In: SAT. pp. 157–171 (2012)
14. Hojjat, H., Konecný, F., Garnier, F., Iosif, R., Kuncak, V., Rümmer, P.: A verification toolkit for numerical transition systems - tool paper. In: FM (2012)
15. Kindermann, R., Junttila, T.A., Niemelä, I.: SMT-based induction methods for timed systems. In: FORMATS (2012)
16. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. STTT 1(1-2), 134–152 (1997)
17. Mahata, P.: Model Checking Parametrized Timed Systems. Ph.D. thesis, Department of Information Technology, Uppsala University (2005)
18. Méndez-Lojo, M., Navas, J.A., Hermenegildo, M.V.: A flexible, (c)lp-based approach to the analysis of object-oriented programs. In: LOPSTR (2007)
19. Owicki, S.S., Gries, D.: An axiomatic proof technique for parallel programs i. Acta Inf. 6, 319–340 (1976)
20. Rümmer, P., Hojjat, H., Kuncak, V.: Disjunctive interpolants for Horn-clause verification. In: CAV (2013)
21. Sánchez, A., Sankaranarayanan, S., Sánchez, C., Chang, B.Y.E.: Invariant generation for parametrized systems using self-reflection. In: SAS (2012)
22. Waez, T.B., Dingel, J., Rudie, K.: A survey of timed automata for the development of real-time systems. Computer Science Review 9, 1–26 (2013)
23. Yi, W., Pettersson, P., Daniels, M.: Automatic verification of real-time communicating systems by constraint-solving. In: FORTE (1994)

A Appendix

A.1 Example Clauses

In our implementation as part of the Eldarica model checker, timed systems are translated to Horn clauses in a two-step process: first, we generate a set of local clauses representing the transitions of each process; second, those local clauses are combined to a global encoding, following the constraints in Fig. 3 and 5.

As an illustration, we show the local clauses for the example in Sect. 2.1 in Fig. 7.

$$\begin{array}{rcl}
y = c & \to & p_1(c, n, y) \\
p_1(c, n, y) & \to & p_2(c, 0, y) \\
p_2(c, n, y) \wedge (n \neq 0) \wedge \mathbf{go}! & \to & p_3(c, n, y) \\
p_2(c, n, y) \wedge (n = 0) \wedge \mathbf{appr}? & \to & p_3(c, n+1, y) \\
p_3(c, n, y) \wedge \mathbf{leave}? & \to & p_2(c, n-1, y) \\
p_3(c, n, y) \wedge \mathbf{appr}? & \to & p_4(c, n+1, c) \\
p_4(c, n, y) \wedge \mathbf{stop}? & \to & p_3(c, n, c) \\
\hline
x = c & \to & q_1(c, id, x) \\
q_1(c, id, x) \wedge \mathbf{appr}! & \to & q_3(c, id, c) \\
q_3(c, id, x) \wedge (c - x \geq 10) & \to & q_2(c, id, c) \\
q_2(c, id, x) \wedge (c - x \geq 3) \wedge \mathbf{leave}! & \to & q_1(c, id, c) \\
q_3(c, id, x) \wedge (c - x \leq 10) \wedge \mathbf{stop}? & \to & q_4(c, id, c) \\
q_4(c, id, x) \wedge \mathbf{go}? & \to & q_5(c, id, c) \\
q_5(c, id, x) \wedge (c - x \geq 7) & \to & q_2(c, id, c)
\end{array}$$

Fig. 7. The encoding of the example in Fig. 1 into a set of recursive Horn clauses.

A.2 The Fischer Protocol

We illustrate how pair invariants (2-invariants) can be used to verify the parametric Fischer protocol, i.e., the Fischer protocol under participation of infinitely many processes.

The global state of the Fischer system is defined by the following (integer) variables:

- `C, U`: the numerator and denominator of the system time.

– `idVar`: a global variable used by the protocol.
– `num`: a global variable expressing how many processes have currently entered the critical section. It is considered an error if `num` ever exceeds 1.

The local state of a process is defined by:

– `x`: a local clock. Resetting the clock is encoded as `x := C`, the time since the last reset is computed by the expression `C - x`.
– `t`: the control state, encoded as an integer in $\{0, 1, \ldots, 4\}$.

The following automaton describes the behaviour of one process:

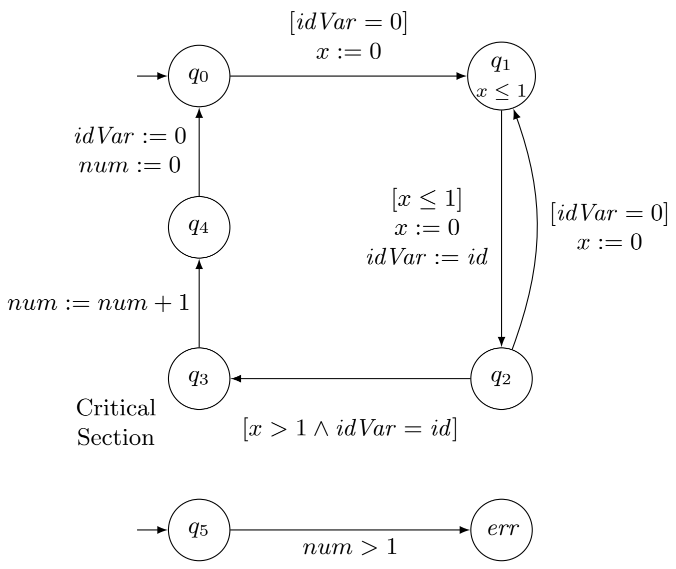


Fig. 8. Behaviour of a single process in Fischer's protocol

The resulting Horn constraints are the following. Note that the protocol uses 0 as a magic process identifier, assuming that no actual process has id 0. For reasons of simplicity, we kept this convention in the Horn clauses, which leads to additional disequalities `\+(id = 0)` in all clauses.

% Symmetry

```
P2(C, U, idVar, num, id, x, t, id2, x2, t2) :-
     P2(C, U, idVar, num, id2, x2, t2, id, x, t),
     (U > 0), \+(id = 0), \+(id2 = 0), \+(id = id2).
```

% Initiation

```
P2(C, U, idVar, num, id, x, t, id2, x2, t2) :-
     (U > 0), \+(id = 0), \+(id2 = 0), \+(id = id2),
     (num = 0), (idVar = 0), (x = C), (x2 = C), (t = 0), (t2 = 0).
```

% Consecution. One clause for every transition.

```
P2(C, U, idVar, num, id, x1P, r1, id2, x2, t2)  :-
     P2(C, U, idVar, num, id, x1, r0, id2, x2, t2),
     (U > 0), \+(id = 0), \+(id2 = 0), \+(id = id2),
     (idVar = 0), (C - x1P =< U), (x1P = C), (r1 = 1), (r0 = 0).

P2(C, U, idVar2, num, id, x1P, r2, id2, x2, t2) :-
     P2(C, U, idVar1, num, id, x1, r1, id2, x2, t2),
     (U > 0), \+(id = 0), \+(id2 = 0), \+(id = id2),
     ((C - x1) =< U), (x1P = C), (idVar2 = id), (r1 = 1), (r2 = 2).

P2(C, U, idVar, num, id, x1P, r1, id2, x2, t2)  :-
     P2(C, U, idVar, num, id, x1, r2, id2, x2, t2),
     (U > 0), \+(id = 0), \+(id2 = 0), \+(id = id2),
     (idVar = 0), ((C - x1P) =< U), (x1P = C), (r1 = 1), (r2 = 2).

P2(C, U, idVar, num, id, x, r3, id2, x2, t2)   :-
     P2(C, U, idVar, num, id, x, r2, id2, x2, t2) ,
     (U > 0), \+(id = 0), \+(id2 = 0), \+(id = id2),
     (C - x > U), (idVar = id), (r2 = 2), (r3 = 3).

P2(C, U, idVar, num2, id, x, r4, id2, x2, t2)  :-
     P2(C, U, idVar, num1, id, x, r3, id2, x2, t2) ,
     (U > 0), \+(id = 0), \+(id2 = 0), \+(id = id2),
     (num2 = num1 + 1), (r3 = 3), (r4 = 4).

P2(C, U, idVar2, num2, id, x, r0, id2, x2, t2) :-
     P2(C, U, idVar1, num1, id, x, r4, id2, x2, t2) ,
     (U > 0), \+(id = 0), \+(id2 = 0), \+(id = id2),
     (idVar2 = 0), (num2 = 0), (r4 = 4), (r0 = 0).
```

% Consecution. Only transitions that modify global state are considered.

```
P2(C, U, idVar2, num, id3, x3, t3, id2, x2, t2) :-
     P2(C, U, idVar1, num, id3, x3, t3, id2, x2, t2),
     P2(C, U, idVar1, num, id, x1, r1, id2, x2, t2),
     P2(C, U, idVar1, num, id, x1, r1, id3, x3, t3),
     (U > 0), \+(id = 0), \+(id2 = 0), \+(id3 = 0),
     \+(id = id2), \+(id2 = id3), \+(id = id3),
     ((C - x1) =< U), (x1P = C), (idVar2 = id), (r1 = 1), (r2 = 2).

P2(C, U, idVar, num2, id3, x3, t3, id2, x2, t2)  :-
     P2(C, U, idVar, num1, id3, x3, t3, id2, x2, t2),
     P2(C, U, idVar, num1, id, x, r3, id2, x2, t2) ,
     P2(C, U, idVar, num1, id, x, r3, id3, x3, t3),
     (U > 0), \+(id = 0), \+(id2 = 0), \+(id3 = 0),
     \+(id = id2), \+(id2 = id3), \+(id = id3),
     (num2 = num1 + 1), (r3 = 3), (r4 = 4).

P2(C, U, idVar2, num2, id3, x3, t3, id2, x2, t2) :-
     P2(C, U, idVar1, num1, id3, x3, t3, id2, x2, t2),
     P2(C, U, idVar1, num1, id, x, r4, id2, x2, t2) ,
     P2(C, U, idVar1, num1, id, x, r4, id3, x3, t3),
     (U > 0), \+(id = 0), \+(id2 = 0), \+(id3 = 0),
     \+(id = id2), \+(id2 = id3), \+(id = id3),
     (idVar2 = 0), (num2 = 0), (r4 = 4), (r0 = 0).
```

% Time elapse.

```
P2(C2, U, idVar, num, id, x, t, id2, x2, t2)   :-
     P2(C1, U, idVar, num, id, x, t, id2, x2, t2),
     (U > 0), \+(id = 0), \+(id2 = 0), \+(id = id2),
     (C2 >= C1), \+(t = 1), \+(t2 = 1).

P2(C2, U, idVar, num, id, x, r1, id2, x2, t2)   :-
     P2(C1, U, idVar, num, id, x, r1, id2, x2, t2),
     (U > 0), \+(id = 0), \+(id2 = 0), \+(id = id2),
     (C2 >= C1), (C2 - x =< U), (r1 = 1), \+(t2 = 1).

P2(C2, U, idVar, num, id, x, t, id2, x2, r1)   :-
     P2(C1, U, idVar, num, id, x, t, id2, x2, r1),
     (U > 0), \+(id = 0), \+(id2 = 0), \+(id = id2),
     (C2 >= C1), (C2 - x2 =< U), \+(t = 1), (r1 = 1).

P2(C2, U, idVar, num, id, x, r1, id2, x2, r1)   :-
     P2(C1, U, idVar, num, id, x, r1, id2, x2, r1),
     (U > 0), \+(id = 0), \+(id2 = 0), \+(id = id2),
     (C2 >= C1), (C2 - x =< U), (C2 - x2 =< U), (r1 = 1).
```

% Safety.

```
false :-
     P2(C, U, idVar, num, id, x, t, id2, x2, t2),
     (U > 0), (num > 1).
```