Journal of Universal Computer Science, vol. 10, no. 10 (2004), 1469-1494
submitted: 2/7/04, accepted: 6/10/04, appeared: 28/10/04 © J.UCS

Tools for Parametric Verification. A Comparison on a Case Study

Petr Matoušek
(Brno University of Technology, Czech Republic
matousp@fit.vutbr.cz)

Abstract: Protocol analysis involves several parameters in model specification, for instance, transmission delay or the length of the transmitting window. Verification of the model with parameters is a semi-decision process that depends on the number of clocks, parameters and counters in the model. Using combination of different verification tools for timed models as HyTech, TReX and Uppaal we are able to find relation between parameters satisfying desired property. The paper gives a report on the synthesis of parameters of PGM protocol. We built a formal model based on extended time automata with parameters and verified the reliability property. Our results automatically obtained from the model are consistent with previous results derived manually. The paper describes our experience with parametric verification of multicast protocol PGM. Results mentioned in the work were made with collaboration with Mihaela Sighireanu[1] from LIAFA, Paris.

Key Words: parametric verification, protocol, timed model-checking
Category: C.2.2, F.4.3, I.6.3

1 Introduction

Model-checking is a popular technique of verification of untimed as well as timed systems. In comparison with theorem proving model-checking provides an easy way of specifying and verifying a model without profound knowledge of logics. For timed systems, model checkers like Uppaal, Kronos, If, TReX can be used. The basic goal of model-checking is to verify a property on a specified model using state space search. In practice, a model checker analyses (explicitly or implicitly) all reachable configurations and tests if the model violates desired property.

PGM protocol. In our work we verified a reliability property of PGM protocol. PGM protocol defined in [SFC$^+$01] is a reliable multicast transport protocol for applications that transfers data from multiple sources to multiple receivers. It is a standard defined by Cisco for telecommunication services, for instance, video conferencing. The reliability property defined in the standard states that "PGM either receives all data packets from transmission and repairs, or is able to detect unrecoverable data packet loss". In our model we distinguish three kinds of recovery - everything is recovered (full recovery), few of lost packets are

[1] Mihaela.Sighireanu@liafa.jussieu.fr

recovered (partial recovery), or nothing is recovered (no recovery). In this paper we study *the full recovery property*, i.e., every lost packet is recovered.

Behaviour of the protocol depends on several aspects - the rate of transmission, the speed of packet generation, the length of the transmission window of the source or limits of network elements. Our goal is to find dependecies among these parameters with respect to the full recovery property.

Parametric reasoning works on a model with parameters - variables whose values are not changed during analysis. Using parametric reasoning we can either verify that the system satisfies some property for all possible values of the parameters, or find constraints on the parameters defining the set of all possible values for which the system satisfies a property.

In our case study we focus on the synthesis of parameters, i.e., finding constraints on the parameters that ensure satisfaction of the full recovery property. Our model was implemented and verified using three different tools: Uppaal [PL00], TReX [BCAS01] and HyTech [HHWT97]. Uppaal is a model checker that does not support parameters so we instantiated variables for some values. HyTech is a tool for parametric verification of hybrid systems. Timed automata are a subset of hybrid automata so we can implement our model in HyTech. HyTech had a problem with termination, so we were not able to generate full state space of the model. However, using instantiation of certain parameters and putting constraints on the others we obtained some relations between parameters. The results were confirmed by similar analysis using TReX.

TReX is a tool for parametric verification of extended timed system with counters that implements powerful acceleration that helps to terminate computation. It outputs a graph of symbolic configurations and analyses traces which we used to study system behaviour and detect configurations where the property is satisfied.

Contribution. Contribution of our work is to show how parametric verification works on a non-trivial example of a real protocol and what kind of tools can be used for parametric verification of timed systems. We identify possible bottlenecks of analysis, especially the features of the system that induce non termination. Our observation and recommendations on parametric analysis are mentioned in this work.

Outline of the paper. In the first section, we give a short summary of theoretical background of verification. The second section gives an overview of different issues related to formal analysis of real-time systems. It discusses four steps of analysis of a system - creation of a model, model validation, definition of requirements and verification. The third section introduces the analyzed protocol.

We describe a formal model of the protocol based on extended timed automata with counters [AAB00]. Then we briefly introduce tools we used for verification - Uppaal, TReX, HyTech and show results we achieved. The last section of the paper summarizes our results and experience with parametric verification and discusses our observation of parametric verification of timed systems.

Related work. There are several works on verification of PGM protocol. [BBP02] verifies a simplified timed version of PGM with a linear topology and a one-placed buffer. Two properties - lost info property and no-loss property are verified in this work. The properties are verified by instantiating parameters using Uppaal. [BL02] validates the sliding window mechanism for any number of data packets sent using LASH. The model used is untimed. A more complex timed model of PGM is considered in [BS03], where the model includes parameters. The constraints on parameters are obtained manually and then verified by instantiation. In contrast to that, we did synthesis and verification fully automatically.

2 Theoretical background

This section gives a short summary of theory of timed systems and parametric verification. It is not essential to read this section for understanding the full paper. However, it includes useful definitions of terms that occur in the following parts of the paper.

2.1 Model checking

Model checking is a technique for verifying finite state concurrent systems such as communication protocols. In comparison with theorem proving it can be performed fully automatically. The procedure uses an exhaustive search of the state space of a system to determine if some specification is true or not. The procedure can terminate with a yes/no answer.

The main disadvantage of model checking is the state explosion that occurs if the system contains many components that can make transitions in parallel. Among other sources of complexity there are, for instance, infinite domain of variables, or the number of parallel processes, etc. They cause that the number of global system states may grow exponentially with the number of processes. However, there are various techniques to decrease the size of the state space of a system - partial order reduction, binary decision diagrams (BDDs), abstraction, symmetry etc.

Partial order reduction. This technique is based on the following observation: computations that differ in the ordering of independently executed events are usually indistinguishable by the specification and can be considered equivalent. There are various approaches to the partial order reduction mentioned in [CGP99]: the stubbern sets proposed by A.Valmari in [Val90], the ample sets presented by D.Peled in [Pel94], the persistent sets of P.Godefroid introduced in [God90], or McMillan's unfolding technique in [McM92]. Using these methods we can decrease the size of the state space. As practice shows, partial order reduction is a very successfull method for software verification.

Binary decision diagrams (BDDs). One of the approaches to avoid the state explosion problem is using a compact representation of the state space called binary decision diagrams and symbolic model checking over them. Binary decision diagrams introduced by R.E.Bryant in [Bry86] provide a canonical form for boolean formulas that is more compact than conjuctive or disjunctive normal form, and very efficient algorithms for manipulating them. Because the symbolic representation captures some of the regularity in the state space determined by circuits and protocols, it is possible to verify systems with an extremely large number of states. BDDs are very successful especially in hardware verification.

Abstraction. The use of abstraction is based on the observation that there are relationships among data in the system. A mapping between the actual data values in the system and a small set of abstract data values can reduce the complexity of model checking. Abstraction can be extended to the states and transitions of the system. The abstract system is often much smaller than the actual system, and it is simpler to verify properties at the abstract level. More about abstraction can be found at [CGP99, chapter 13].

Modeling continuous time. For timed systems, clocks are real values. The timed transition system is infinite and cannot be simply used for automated verification. However, D.Dill and R.Alur in [AD94] introduced the notion of region equivalence over clock assignments and proved that reachability problem is decidable. The main idea of the region technique is that it is possible to find a finite representation of the valuation graph which represents all the necessary reachability information symbolically. At first, infinitely many symbols of the transition system are reduced to the finite number using time abstraction where time increments over time-transitions are hidden (see section about Timed Automata later in the article). Then, using equivalence over the state space the number of states (infinite) is represented by the finite number of classes according to the equivalence. In other words, the state space is factorized into a finite number of regions.

Difference Bound Matrices (DBMs). A more efficient representation of the state-space for timed systems is based on the notion of zone ([Dil89],[AD94]). A clock zone φ is a conjunction of inequalities that compare either a clock value or the difference between two clock values to an integer. We allow inequalities of the following type: $x \prec c, c \prec x, x - y \prec c$ where $\prec \in \{<, \leq\}, x, y \in \mathcal{X}$, $\mathcal{X}$ is a set of clocks, and $c \in \mathbb{N}$. A key property of the set of clock zones is closure property under three operations - the intersection, elapsed time and clock reset.

Clock zones can be efficiently represented using matrices [Dil89]. In order to express a uniform notation for clock zones we introduce a special clock x_0 that is always 0. Then, any clock zone φ can be written as a conjuction of constraints of the form $x - y \prec c$, for $x, y \in X, \prec \in \{<, \leq\}$, and $n \in \mathbb{N}$.

Let $\mathcal{D}$ be a difference bounded matrix representing clock zone φ. Each entry $\mathcal{D}_{i,j}$ is in the form $(d_{i,j}, \prec_{i,j})$ and represents the inequality $x_i - x_j \prec d_{i,j}$, where $\prec_{i,j}$ is either $<, \leq$ or $(\infty, <)$, if no such bound is known.

Reachability analysis. Reachability analysis can be used to check properties on states. The main effort on verification of timed systems has been put on safety properties that can be checked using reachability analysis by exploring the state space of timed automata. Symbolically, we can describe reachability analysis for timed automata with the following algorithm:

```
R := $S_0$                 // a set of reached states
F := $F_0$                 // a set of final states
while $((R \cap F) = \emptyset)$       // reached states are different from final
  R' := post(R)          // compute a set of successors of R
  if $(R' \subseteq R)$ return "No" // the whole state space was reached
  R := R'                // add successors to the set of all states
end while
return "Yes"             // final state was reached
```

Reachability analysis consists of two basic steps: computing the set of successors of a set of reached states - a result of procedure *post()*, and searching for states that satisfy or contradict given properties - expressed by an intersection of a set of reached states and a set of final states. In figure above, the analysis finishes if the final state is reached, or if the entire state space is generated without being intersected with a set of final states.

Symbolic reachability analysis is a powerful paradigm for verification of infinite-state systems, such as parametrized communicating systems [BCALS01]. Symbolic reachability analysis uses finite structures to represent infinite sets of configurations, and iterative exploration procedures to compute the set of all reachable configurations, or an upper approximation of this set. To help with the termination these procedures are enhanced by acceleration techniques which allow us to compute the effect of sequences of transitions in one step instead that of one single transition in the system. As we will see in next sections acceleration plays an important role in parametric verification.

Acceleration. As mentioned above, verification of an infinite state system can be enhanced by acceleration in order to help termination. Instead of repeating the same transition in the reachability graph we can replace these transitions by an acceleration step. The acceleration step corresponds to the computation of an upper approximation of the set of reachable configurations by iterating a sequence of transitions an arbitrary number of times. For instance, starting with initial value x = 0, the iteration of a transition which increments x by 2 leads to the set of configurations $\{0, 2, 4, \ldots\}$ which can be represented by constraint $x = 2n$, with $n \geq 0$. Acceleration techniques allow us to compute a finite representation in one step instead of computing the infinite sequence of approximations $\{0\}, \{0, 2\}, \{0, 2, 4\}, \ldots$

2.2 Timed Automata

Timed automata introduced by R.Alur and D.Dill in [AD94] serve as a technique for modeling FSM with explicit time what is essential for specification and analysis of real-time systems. Here, we briefly show the theory of timed automata.

Transition system with timing constraints. To express system behaviour with timing constraints, we consider finite graphs augmented with a finite set of (real-valued) *clock*. The vertices of a graph are called *locations*, and the edges are called *switches*. While the switches are instantaneous, time can elapse in a location. A clock can be reset to zero simultaneously with any switch. At any instant, the reading of a clock equals the time elapsed since the last clock reset. With each switch we associate a clock constraint, and we require that the switch may be taken only if the current values of the clock satisfy the constraint. With each location we associate a clock constraint called an *invariant*, and we require that time can elapse in a location only if its invariant stays true.

Clock constraints and clock interpretation. For a set X of clocks, the set $\Phi(X)$ of *clock constraints* φ is defined by grammar

$$\varphi := x \leq c \mid c \leq x \mid x < c \mid c < x \mid \varphi_1 \wedge \varphi_2,$$

where x is a clock in X and c is a constant in $\mathbb{Q}$. A *clock interpretation* ν for a set X of clocks assigns a real value to each clock; that is, it is a mapping from X to the set $\mathbb{R}^{\geq 0}$ of non-negative reals. For $\Delta \in \mathbb{R}, \nu + \Delta$ denotes the clock interpretation which maps every clock x to the value $\nu(x) + \Delta$. For $Y \subseteq X, \nu[Y := 0]$ denotes the clock interpretation for X which assigns 0 to each $x \in Y$, and agrees with ν over the rest of the clocks.

Syntax. A *timed automaton* A is a tuple $\langle L, L^0, \Sigma, X, I, E\rangle$, where

- L is a finite set of locations,
- L^0 is a finite set of initial locations,
- Σ is a finite set of labels,
- X is a finite set of clocks,
- I is a mapping that labels each location s with a clock constraint $\Phi(X)$, and
- $E \subseteq L \times \Sigma \times 2^X \times \Phi(X) \times L$ is a set of switches. A switch $e = \langle s, a, \varphi, \lambda, s'\rangle$ represents an edge from location s to location s' on symbol a. φ is a clock constraint over X that specifies when the switch is enabled, and the set $\lambda \subseteq X$ gives the clocks to be reset with the switch.

Semantics. The semantics of timed automaton A is defined by a transition system S_A associated with it. A state of S_A is a pair (s, ν) such that s is a location of A and ν is a clock interpretation for X such that ν satisfies the invariant $I(s)$. The set of all states of A is denoted Q_A. There are two types of transitions in S_A:

1. **Time transition**: for a state (s, ν) and a real-valued time increment $\Delta \geq 0$ there holds $(s, \nu) \xrightarrow{\Delta} (s, \nu + \Delta)$ if for all $0 \leq \Delta' \leq \Delta, \nu + \Delta'$ satisfies the invariant $I(s)$.
2. **Action transition**: for a state (s, ν) and a switch $\langle s, a, \varphi, \lambda, s'\rangle$ such that ν satisfies φ there holds $(s, \nu) \xrightarrow{a} (s', \nu[\lambda := 0])$.

2.3 Extended Timed Automata with Parameters

Classical timed automata where clocks can only be compared to constants do not allow us a parametric reasoning. Moreover, it has been shown in [AHV93] that for parametric timed automata, the reachability problem is, in general, undecidable. In [AAB00], the authors propose a semi-algorithmic approach that allows to deal with parametric counter and timed sytems. They define a new symbolic representation called Parametric DBMs (PDBMs) for use in reachability analysis, and provide powerful technique for computing representations of their sets of reachable configuration.

Parametric Timed System. A Parametric Timed System (PTS) is a tuple $\mathcal{T} = \langle L, X, P, I, \delta\rangle$, where

- L is a finite set of locations

- X is a finite set of clocks,
- P is a finite set of parameters,
- $I : L \rightarrow SC(X,P)$ is a mapping that associates invariants with locations, $SC(X,P)$ is a simple parametric constraint expressed as a conjunction of formulas of the form $x \prec t$ or $x - y \prec t$ where $x, y \in X, \prec \in \{<, \leq\}$, t is an arithmetical term over the set of parameters P defined by the grammar $t ::= 0 \mid 1 \mid p \mid t - t \mid t + t \mid t * t$ where $p \in P$ is a parameter.
- δ is a set of transitions of the form (l_1, g, sop, l_2) where $l_1, l_2 \in L, g \in SC(X,P)$ is a guard, and *sop* is a simple operation over X - a special kind of assignment of the form $x := y + t$ or $x := t$ where $x, y \in X$ and t is an arithmetical term over the set of parameters.

Clocks and parameters range over a set $\mathcal{D}$ which can be either the set of positive reals $\mathbb{R}^{\geq 0}$(dense time model) or the set of positive integers $\mathbb{N}$ (discrete time model). A configuration of $\mathcal{T}$ is a triplet $\langle l, \nu, \gamma \rangle$ where $l \in L$ is a location, $\nu \in X \rightarrow \mathcal{D}$ is a valuation of the clocks, and $\gamma : P \rightarrow \mathcal{D}$ is a valuation of parameters.

Semantics. Similarly to timed automata we define two types of transitions:

1. **Time transition**: $\langle l_1, \nu_1, \gamma_1 \rangle \rightarrow \langle l_2, \nu_2, \gamma_2 \rangle$ iff $l_1 = l_2$, $\gamma_1 = \gamma_2$ and there exists $\Delta \in \mathcal{D}$ such that $\nu_2 = \nu_1 + \Delta$ and for all $\Delta' \leq \Delta : (\nu_1 + \Delta', \gamma)$ satisfies the invariant $I(l_1)$.
2. **Action transition**: for a state $\langle l_1, \nu_1, \gamma_1 \rangle$ and a transition $\tau = (l_1, g, sop, l_2) \in \delta$ we define a transition relation $\rightarrow_\tau$ between configurations as $\langle l_1, \nu_1, \gamma_1 \rangle \rightarrow_\tau \langle l_2, \nu_2, \gamma_2 \rangle$ such that (ν_1, γ_1) satisfies g and $\nu_2 = sop(\nu_1) \ \wedge \ \gamma_1 = \gamma_2$.

Parametric Difference Bound Matrix (PDBM). Let $\mathcal{M}$ be a parametric difference bound matrix that encodes the contraints in the form $x_i - x_j \prec t$ (similar to DBMs) where $x_i, x_j \in X$ are clocks, t is an arithmetical term over the set of parameters P defined by the grammar $t ::= 0 \mid 1 \mid p \mid t - t \mid t + t \mid t * t$ where $p \in P$ is a parameter, and $\prec \in \{<, \leq\}$. A constrained PDBM is a pair $(\mathcal{M}, \varphi)$ where $\mathcal{M}$ is a PDBM and φ is a parameter constraint - quantifier-free formula over parameters given by grammar $\varphi ::= t \leq t \mid \neg\varphi \mid \varphi \vee \varphi$.

2.4 Symbolic reachability graph

Symbolic configuration. A symbolic configuration Γ is a pair (l, S), where $l \in L$ is a control state, and S is a constrained PDBM. A symbolic configuration $\Gamma = (l, S)$ includes $\Gamma' = (l', S')$ if $l = l'$ and $[S] \supseteq [S']$.

Symbolic reachability graph. Let $\mathcal{T}$ is a Parametric Timed System. Starting from a symbolic configuration Γ we construct a symbolic reachability graph $SG(\mathcal{T}) = (V, E)$ where V is a finite set of structures representing infinite set of configurations of $\mathcal{T}$ and $E \subseteq V \times V$ is a finite set of transitions between structures in V. Each vertex V is a symbolic configuration and an edge E corresponds to a transition of $\mathcal{T}$. The vertices of the symbolic graph are treated according to a depth-first traversal. The construction stops when each symbolic configuration that can be generated is covered with some symbolic configuration that has been already computed. During this construction, acceleration is needed in order to help with the termination.

A symbolic reachability graph is one of the results of verification using TReX - see Figure 7. It is a powerful mean for analysis of system behaviour.

3 Formal analysis of real-time systems

In the last ten years we can see a rapid development in the field of formal methods for the specification, analysis and verification of real-time systems. Especially in continuous time modeling, there has been a great progress in verification techniques despite the fact that continuous time means dealing with an infinite state space. Thanks to pioneering approaches and new techniques we can fully automatically analyse and verify some classes of continuous time systems.

In this paper we present our experience with parametric verification of PGM protocol. We concentrate on three tools we used for analysis - Uppaal, HyTech, and TReX. Using combination of these tools we were able to automatically find relations between parameters of the protocol. Our results automatically obtained are consistent with previous results derived manually in [BS03].

In this section we give an overview of issues that are related to formal analysis of real-time systems.

Formal analysis. Formal analysis of a system consists of four basic parts:

1. Creation of a model.
   - The model is an abstraction of a real system. The question is a level of abstraction - we need to detect parts of a model that can be hidden in the abstract model without changing functionality of a system.
   - The model should be described using formal language. Description is then unambigous and precise.
   - Description should be easy to read and understand. This little bit contradicts with the previous point, especially for those who are not familiar with the reading of formal descriptions. A great advantage is in using

tools with graphical interface where the formal description is hidden behind the simple figures as we can see, for instance, in Uppaal.

- Description language should be easily transformed into an input language of a verification tool.

2. Validating the model.
   - The main issue of validation can be expressed by the question "Does the model correspond to the modeled system?"
3. Definition of required properties.
   - Among such properties there can be reliability, response of the system, safety etc.
   - Properties are written mostly as assertions or logical formulas.
4. Simulation or/and verification of required behaviour.
   - Simulation is useful for proving that the model corresponds to the modeling system. We can as well observe the behavior of the system for well-known conditions and input values.
   - Simulation can never state that the system is correct or that a specified property is true for all input values and control states of the system.
   - Verification can prove that the property is valid for all input values and states of the system.
   - If verification ends with the result that the property is not satisfied we can find a counter-example where the property is violated. It helps us to detect flaws in system design.

In our case study we analysed a system using following steps:
(i) System specification. We created an abstract model and described it using extended timed automata with parameters. The automata communicate using shared variables.
(ii) Implementation in verification tools. Formal model was translated into input languages of Uppaal, HyTech, and TReX. Transformation was straightforward, because Uppaal and TReX use timed automata for model description. HyTech models systems using hybrid automata. Timed automata can be considered as a subset of hybrid automata, so translation of the model to HyTech was not difficult too.
(iii) Verification. Because Uppaal does not support parameters, we instatiated parameters and verified the model for a set of different values. Parametric verification was made in HyTech and TReX. During verification we faced a problem of non termination. If the analysis did not terminate we used following algorithm:

1. We checked traces/runs to find a source of non-termination.
2. We refined a model - we put an additional restriction on parameters, instantiated some parameters, etc.
3. Verification was repeated.

Parametric verification resulted in relations between parameters (parameter synthesis) that satisfied the observed property.

4 Modeling PGM

This section deals with the modeling of PGM protocol. Our model of PGM was tuned during time of analysis - a few features were modeled not very precisely, first models were too large to finish verification etc. Here, we present the last version of our model where we were able to prove desired properties.

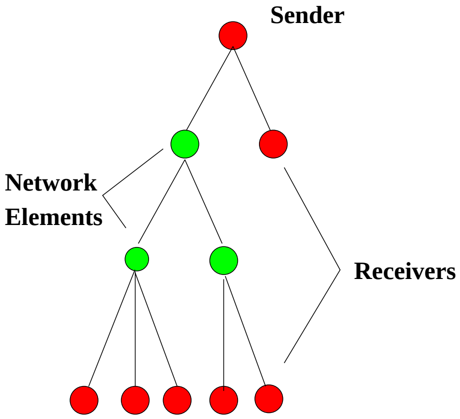


Figure 1: PGM - multicast transmission.

PGM protocol. PGM protocol defined in [SFC$^+$01] is a complex multicast protocol. It works on a network of nodes with multiple senders and multiple receivers. Its dynamic behavior is depicted in Figure 1. Transport-layer originators of PGM data packets are referred to as senders, transport-layer consumers of PGM data packets are referred to as receivers, and network-layer entities in the intervening network are referred to as network elements.

In the normal course of data transfer, a sender multicasts sequenced data packets (ODATA), and receivers unicast selective negative acknowledgments

(NAKs) for data packets detected to be missing from the expected sequence. Network elements forward NAKs hop-by-hop to the source, and confirm each hop by multicasting a NAK confirmation (NCF) in the response to the interface at which the NAK was received. Repairs (RDATA) may be provided by the sender in the response to a NAK.

Since NAKs provide the sole mechanism for reliability, PGM is particularly sensitive to their loss. To minimize the NAK loss, PGM defines a network-layer hop-by-hop procedure for reliable NAK forwarding - see Figure 2.

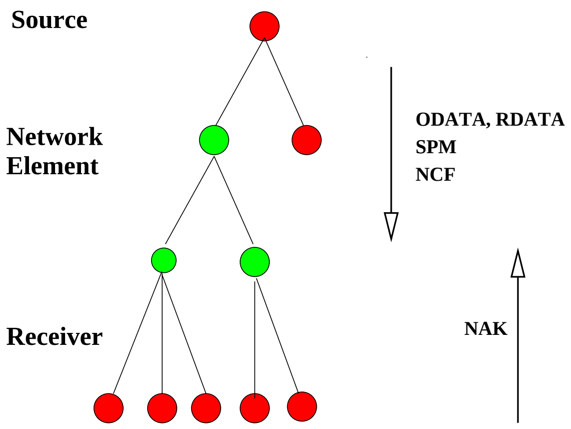


Figure 2: Data packets defined in PGM.

In our approach we abstract the model to a simple one-sender and one-receiver system. Joining and leaving multiple nodes during session can be considered as nodes missing data [BS03].

Abstract model. Analysing the full PGM protocol is beyond limits of current verification tools because of

- dynamic topology - joining/leaving out a node,
- multiple senders,
- a lot of different packet types (`SPM`, `NCF`, `NAK`),
- a lot of processes, counters and clocks.

Possible sources of complexity are the number of clocks and counters, the number of parameters, non-linear relations between variables. However, by combination of different tools for analysis we were able to prove the reliability property.

Our abstract model is based on a global view of the protocol running in the sender and one of its receivers, as presented on Figure 3. The intermediate network between the sender and the receiver is abstracted into a unreliable, unbounded FIFO queue. Only data packets (`ODATA`) are transmitted between the sender and the receiver, the other packets (`SPM`, `NAK`, `NCF`, `RDATA`) are abstracted also.

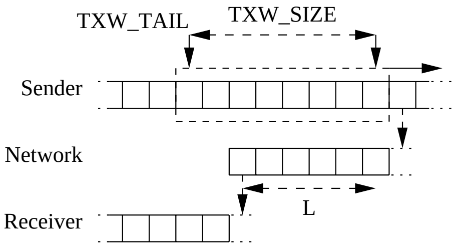


Figure 3: Abstract model of PGM.

Using the abstract model we can abstract from individual packet numbers. We are not interested in a precise sequence number to detect losses. When the receiver receives a packet it is informed by global variable that there was a loss (or multiple losses) preceding the incoming packet. The receiver knows an actuall size of the data in the network (variable `L` - the length of FIFO queue), the number of old data in the sender's transmitting window and the speed of the transmission. From these values it is possible to detect if a lost packet can be recovered. In Figure 3 we can see that the number of packets present in transmitting window for recovery is `TXW_SIZE - L`. The real possibility of recovery depends on the speed of transmission expressed by parameter `RATE`.

Formal description. For verification purposes we use extended timed automata with parameters (see previous section) to describe our abstract model. The choice of the formalism was made with regard to verification tools we intented to use. Our prime goal was to synthetize parameters of a protocol. So we looked for tools that implement parametric verification. Because the observed system is a communication protocol working in real-time, the need of explicit continuous time was raised. Following these requirements we decided to use extended timed automata with parameters to formally describe PGM protocol.

Our PGM model is composed of three automata – a sender, a network and a receiver with six parameters, one finite variable, two clocks, two counters and two communication channels, see Figures 4, 5, and 6.

The automata work simultanously and are synchronized by rendez-vous on gates SN and NR. They communicate using shared variables L and lp. The states labelled by C are urgent states, i.e., states where the time is not allowed to advance.

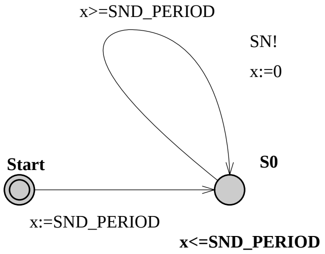


Figure 4: PGM model - sender

The sender. The sender generates new data each period (SND_PERIOD). The data sent are stored in the transmitting window that advances each time new data are sent. The transmitting window is fully opened during the session to recover as many data packets as possible. If data loss is detected, we test if an original data packet is in the transmitting window. If not, a non-recoverable data loss has happened and the full recovery property is violated.

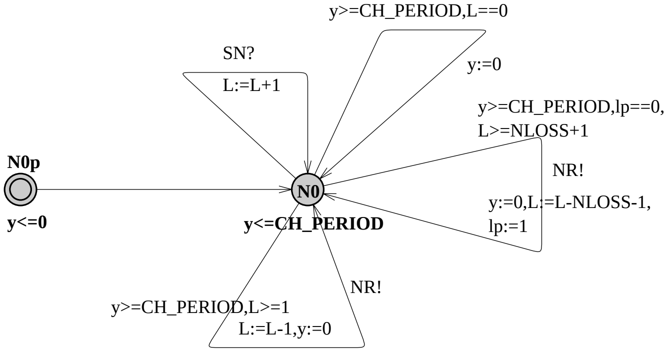


Figure 5: PGM model - network

The network. The network automaton models the transmission channel between the sender and the receiver with transmission delays and non-deterministic losses. The network receives data from the sender and increments the length of buffer L. The buffer is unbounded, it can grow without limitation. The network element either delivers data to the receiver with the speed defined by `CH_PERIOD` parameter or multiple data are discarded in order to model losses during transmission. The model allows `NLOSS` data packets to be lost, `NLOSS` being a parameter. The initial buffer length is set to `BUFFER_LENGTH`, which means the system is in process of communication - we don't model opening and closing stages of communication.

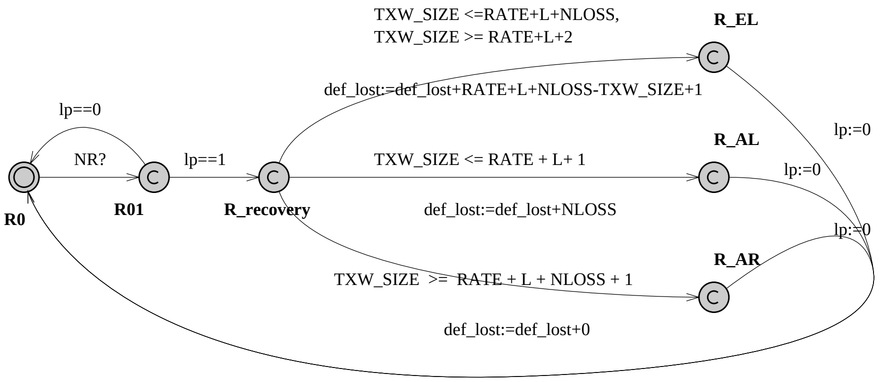


Figure 6: PGM model - receiver

The receiver. The receiver is informed about losses using a global variable *lp*. When a loss occurs the receiver calculates possibility of recovery. The result depends on `TXW_SIZE`, `BUFFER_LENGTH`, `RATE` and the current length of the buffer `L`'.

By reasoning about the recovery of transmitted data we distinguish three possible cases - every lost packet can be recovered, some lost packets can be recovered or nothing can be recovered. This depends on the size of the sender's transmitting window, the speed of transmission, the delay in the network etc. These cases can be described by following manually obtained results:

∀R All lost packets may be recovered (full recovery) if $\texttt{TXW\_SIZE} > \texttt{RATE} + \texttt{L}' + \texttt{NLOSS}$, state R_AR,

$\forall$L None of the $\texttt{NLOSS}$ lost packets may be recovered (no recovery) if $\texttt{TXW\_SIZE} \leq \texttt{RATE} + \texttt{L}' + 1$, state R_AL, or

$\exists$R Some of the lost packets may be recovered (partial recovery) if $\texttt{TXW\_SIZE} > \texttt{RATE} + \texttt{L}' + 1$ and $\texttt{TXW\_SIZE} \leq \texttt{RATE} + \texttt{L}' + \texttt{NLOSS}$ (state R_EL) .

The full recovery may be done for the first case if the parameters satisfy constraint $\texttt{SND\_PERIOD} \geq \texttt{CH\_PERIOD} \wedge \texttt{TXW\_SIZE} \geq \texttt{RATE} + \texttt{BUFFER\_LENGTH}$. This constraint on parameters was obtained manually in [BS03]. In this paper we focus on automatical synthesis of parameters. However, it is interesting to compare the manually obtained results with output of verification tools listed in the following section. It can be seen that the results are consistent.

5 Tools for parametric verfication

In parametric verification we used three tools - HyTech, TReX and Uppaal. In this part we introduce the tools and our results. As mentioned in [AHV93], a large class of parametric verification problems is undecidable. In [AAB00] the authors introduce a semi-logaritmic approach based on an expressive symbolic representation, parametric DBMs, and extrapolation techniques that allow one to speed up reachability analysis and help its termination. We will see how important an effective extrapolation technique is in comparison with TReX and HyTech.

5.1 HyTech

HyTech [HHWT95] is a tool for analysis of linear hybrid automata [ACHH93]. A hybrid automaton is a mathematical model for hybrid systems that models both their discrete and continuous behaviour. Hybrid automata can be considered a generalization of timed automata with continuous variables. Timed automata have one type of continuous variables - clocks. Generally, hybrid systems are undecidable [HHWT95]. Linear hybrid systems form a subclass of hybrid systems which can be analysed semi-automatically [ACHH93]. Invariants, guards and actions in linear hybrid systems depend linearly on time and other variables.

HyTech is a symbolic model checker for linear hybrid automata. The ability of HyTech to perform parametric analysis is an important feature. It is able to synthetize parameter values, i.e., to find the correct values for the parameters so that the system will satisfy a specified property.

Model description. HyTech takes a description of a model as in input in the form of linear hybrid system and analysis commands. System description contains variables of several types: *discrete, clock, stopwatch, parameter* and *analog.*

Guards and constraints are composed of linear terms and expressions. Each automaton is composed of locations and their transitions. Locations are labeled with their invariants. Transitions contain guards with enabling conditions and the successor location. There must be provided an initial state of an automaton and an initial value of the variables.

Model analysis. Analysis in HyTech is specified by two parts: declaration of regions, and a sequence of analysis commands. Analysis commands provide a means of manipulating and outputting regions. At any time instant, the state of a hybrid automaton is specified by a location and constraints on variables. This is called a region. HyTech computes the forward reachable region by finding the limit of the infinite sequence *I, post(I),* $post^2(I)$, ... of regions. All timed safety requirements, including bounded-time response requirements, can be verified using the reachability set. However, the iteration scheme is a semidecision process: there is no guarantee of termination.

In our first approach, we computed the reachability set of the system. The property to be verified in the system was expressed in negative form using a region that violates the property: $final\_reg := def\_lost > 0$. Term $def\_lost > 0$ describes states where the recovery property is not satisfied, i.e. number of definitely lost packets is greater then zero. Firstly, HyTech generates a set of all reachable configurations of the system. Then intersection with specified property is applied on the set. If the property holds we get a non-empty result in the form of equations between parameters that satisfy our model and specified conditions. Declaration of analysed region and analysis commands in HyTech for the first approach is following:

```
-- definition of initial and final region
init_reg, final_reg: region;

-- region inizialization
init_reg := loc[sender] = S0 & x = SND_PERIOD & loc[Node] = N0 & y =
  0 & L = BUFFER_LENGTH & lp = 0 & loc[receiver] = R0 & def_lost = 0
  & RATE >= 1 & TXW_SIZE >= 1 & NLOSS >= 1 & BUFFER_LENGTH >= 1 &
  CH_PERIOD >= 1 ;

-- a violation state (final_reg)
final_reg :=  def_lost > 0;

-- analysis
reached := reach forward from init_reg endreach;
prints "------------";
print omit all locations
        hide non_parameters in reached & final_reg endhide;
```

For the first approach computation did not terminate. In symbolic model checking there are very important techniques like acceleration that help to speed up and terminate the analysis. For above written example HyTech had a problem to accelerate and after few hours the computation failed because of the lack of memory.

The second approach analyses the model on the fly. At first, the nearest reachable region is computed using *post()* operation and then, immediately intersection of the region and the undesirable property *final_region* is tested. If the intersection is non empty, non-reliable state was reached. If the intersection is empty, we continue in the iteration. We cannot find all states satisfying the property but we can determine states that violate the property and synthetize parameters for non-allowed states. In HyTech, the second approach is written as follows:

```
init_reg, reached,old, final_reg: region;

init_reg := loc[sender] = S0 & x = SND_PERIOD & loc[Node] = N0 & y =
  0 & L = BUFFER_LENGTH & lp = 0 & loc[receiver] = R0 & def_lost = 0
  & RATE >= 1 & TXW_SIZE >= 1 & NLOSS >= 1 & BUFFER_LENGTH >= 1 &
  CH_PERIOD >= 1 ;

final_reg :=  def_lost > 0;

-- initialize region reached:
reached := init_reg;
prints "------------";
while empty(reached & final_reg) do
  old:= reached;
  reached:=post(old);
  print diff(reached, old);
endwhile;

prints "reached & final_reg:";
print omit all locations hide non_parameters in reached & final_reg
endhide;
```

Results. During the analysis of PGM we distinguish four different cases depending on the speed

- *Case 1:* `SND_PERIOD` > `CH_PERIOD` - the rate of arrivals is less than that of departures, the size of the queue converges to zero. Following constraints on parameters were synthetized:

  ```
  CH_PERIOD < SND_PERIOD & CH_PERIOD >= 1 & NLOSS >= 1 & NLOSS <= BUFFER_LENGTH
  & RATE >= 1 & TXW_SIZE >= 1 & TXW_SIZE + NLOSS <= RATE + BUFFER_LENGTH + 1
    |
  RATE >= 1 & NLOSS <= BUFFER_LENGTH & TXW_SIZE + NLOSS >= RATE + BUFFER_LENGTH + 2
  & CH_PERIOD < SND_PERIOD & CH_PERIOD >= 1 & TXW_SIZE <= RATE + BUFFER_LENGTH
  ```

  The result shows that for `TXW_SIZE` ≤ `RATE` + `BUFFER_LENGTH` − `NLOSS` (the first part of the formula) nothing can be recovered and for `TXW_SIZE` > `RATE` + `BUFFER_LENGTH` − `NLOSS` (the second part of the formula) some losses can be recovered. This corresponds to results obtained by TReX - see later.

- *Case 2:* `SND_PERIOD` = `CH_PERIOD` - arrivals are of the same speed as departures, the size of the queue decreases to zero by the number of losses `NLOSS` these are non-deterministic losses in the queue.

  The constrained obtained are the same as in the previous case.

- *Case 3:* CH_PERIOD/SND_PERIOD > NLOSS - arrivals are faster then departures and losses, the queue grows beyond any limits.

  For this case and case 4, we introduce a new parameter $q =$ CH_PERIOD/ SND_PERIOD, and we consider that $q \geq 2$. Parameter synthesis obtained by HyTech for $q = 2$ as follows.

```
q >= NLOSS + 1 & SND_PERIOD > 1 & BUFFER_LENGTH >= 1 & NLOSS <= BUFFER_LENGTH + 1
& RATE >= 1 & TXW_SIZE + NLOSS <= RATE + BUFFER_LENGTH + 2 &TXW_SIZE>= 1 & NLOSS >= 1
 |
q >= NLOSS + 1 & NLOSS <= BUFFER_LENGTH + 1 & RATE >= 1 & TXW_SIZE + NLOSS >= RATE
+ BUFFER_LENGTH + 3 & SND_PERIOD > 1 & TXW_SIZE <= RATE + BUFFER_LENGTH + 1
```

  This is for $q = 2$. If we set q equal to $\{3, 4, \ldots\}$ we obtain similar results that differ by constants in relation with TXW_SIZE.

- *Case 4:* NLOSS > CH_PERIOD/SND_PERIOD > 1 - arrivals are faster than departures but not enough to fill the losses between two deliveries, the size of the buffer does not grow fast enough because of the losses.

  The experiments and the results are similar to the third case.

5.2 TReX

TReX [BCAS01] is a tool that allows one to analyse automatically automata-based models equipped with variables of different kinds of infinite domain and with parameters. The models are parametric timed automata extended with integer counters and communicating through unbounded FIFO queues.

The verification technique is improved with an efficient extrapolation technique. TReX allows on-the-fly model checking as well as the generation of the set of reachable configuration and of a finite symbolic graph.

Model description and analysis. A model of the system is specified using an input language that is a subset of IF language [BFG$^+$00]. A model contains timed automata with counters, parameters and gates for synchronization. In .cnd file we specify initial constraints on parameters to help its termination. The output of the verification is a resulting finite graph (.sg), a set of symbolic configurations (.res) and a list of traces/runs (.tr) over a symbolic configuration graph.

Using a set of traces and a graph of symbolic configuration we can observe behaviour of the system and find a relation between parameters satisfying desired property. In our case we search for configurations where the number of definitely lost packets is zero. This configuration satisfies the full recovery property.

In HyTech we were able to verify only counter-examples, i.e., configurations where the property was violated. We did not succeed to generate a full set of reachable configuration. On the contrary, TReX successfully generates a full graph of all reachable configurations. From this graph we can synthetize parameters satisfying the desired property. For instance, in Figure 7 we can see all possible traces (runs) of the model for which the desired property $def\_lost = 0$ holds. The graph was generated from TReX (.tr file) for the full recovery property (def_lost = 0) and CH_PERIOD= SND_PERIOD. We can observe a dependency of an initial value of the buffer BUFFER_LENGTH on current length L of the buffer for all recovery property.

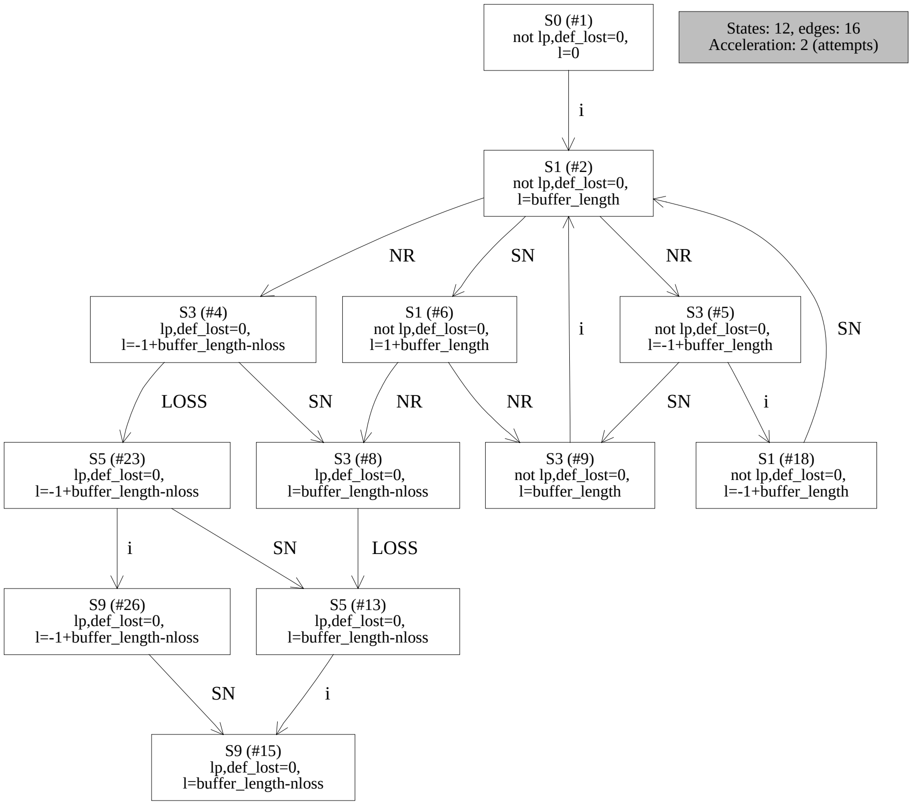


Figure 7: TReX - symbolic reachability graph

Results. The results obtained from a set of symbolic configurations cover all three cases of recovering losses. We can see that relations in the case of no recovery and of partial recovery correspond with HyTech results and full recovery with manually obtained relation. The example is for SND_PERIOD $>$ CH_PERIOD:

- R_AR - full recovery

  txw_size $\geq$ rate + buffer_length and buffer_length $\geq$ nloss + 1

- R_EL - partial recovery

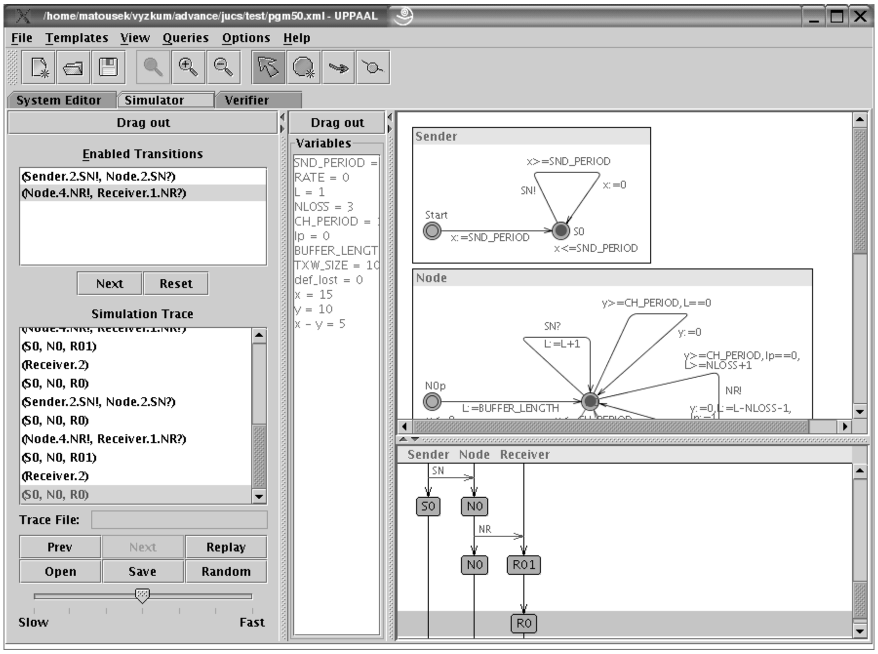
  txw_size $\geq$ rate + buffer_length -nloss - n3 - 1 and twx_size $\leq$ rate + buffer_length - n3 - 3 and buffer_length $\geq$ nloss + n3 - 3 and buffer_length $\geq$ n3 - 2 and n3 $\geq$ 0

- R_AL - no recovery

  txw_size $\leq$ rate -nloss + buffer_length - n3 - 1 and buffer_length $\geq$ nloss + n3 - 2 and buffer_length - n3 - 1 $\leq$ 0 and n3 $\geq$ 0

Variables $nX \in \mathbb{N}$ are called iteration variables. They are used by acceleration procedures to describe symbolically the number of iterations of detected loop.

5.3 Uppaal



```
/home/matousek/vyzkum/advance/jucs/test/pgm50.xml - UPPAAL
```

Enabled Transitions

```
(Sender.2.SN!, Node.2.SN?)
(Node.4.NR!, Receiver.1.NR?)
```

Simulation Trace

```
(Node.4.NR!, Receiver.1.NR?)
(S0, N0, R01)
(Receiver.2)
(S0, N0, R0)
(Sender.2.SN!, Node.2.SN?)
(S0, N0, R0)
(Node.4.NR!, Receiver.1.NR?)
(S0, N0, R01)
(Receiver.2)
(S0, N0, R0)
```

Trace File:

Variables

```
SND_PERIOD =
RATE = 0
L = 1
NLOSS = 3
CH_PERIOD =
lp = 0
BUFFER_LENGT
TXW_SIZE = 10
def_lost = 0
x = 15
y = 10
x - y = 5
```

Figure 8: Simulation of PGM in Uppaal.

UPPAAL is a tool for validation and verification of RT systems developed by colaboration of Uppsala University in Sweden and Aalborg University in Denmark [PL00]. A model is described using timed automata. Verifier checks specified properties that are expressed using simple temporal logic with operators `E<>, A[], E[], A<>`. UPPAAL verifies an existence of a deadlock using special property `A[] not deadlock`.

Model description. A part of UPPAAL tool is a simulator that performs simulation on a specified model. We used the simulator especially in the first stage of the model specification where we tuned our model and compared it with the given protocol - see Figure 8. A model was decribed graphically using a built-in editor. The specification is visual and enables the first check of the consistency of the model. Specification of our model in UPPAAL is in Figures 4, 5, and 6.

Model analysis. In our project we did the verification using UPPAAL states where `def_lost > 0` and `def_lost = 0`. UPPAAL does not support parametric verification, so we instantiated parameters. Using UPPAAL we were able to prove that our results obtained by HYTECH and TREX are consistent and that our model is deadlock-free.

Results. In this part we briefly show obtained results.

- *Result 1:* relation between the current length of the buffer and the number of definitely lost packets for different values of parameters. constants: `CH_PERIOD`: 10, `SND_PERIOD`: 15, `RATE`: 0

`TXW_SIZE`	10	10	10	10	10	12	11	10	10	12	12	13	14	10	10	2
`NLOSS`	3	3	3	3	3	3	3	3	3	3	4	4	4	4	5	5
`BUFFER_LENGTH`	0	5	8	9	10	10	10	11	12	12	12	12	12	5	5	4
max `L`	1	6	9	10	11	11	11	12	13	13	13	13	13	6	6	5
def_lost	0	0	0	1	2	0	1	3	3	2	2	1	0	0	0	0

From the above table we see that:

1. max L = `BUFFER_LENGTH`+ 1
   Because of `CH_PERIOD` < `SND_PERIOD` the buffer cannot grow except for the initial phase of the transmission.

2. $L \geq \texttt{NLOSS} + 1$
   We can produce losses only if the queue is greater then `NLOSS`+1 packets.

- *Result 2:* relation between TXW_SIZE and BUFFER_LENGTH.

TXW_SIZE	10	7	7	7	7
NLOSS	3	3	4	5	5
BUFFER_LENGTH	5	5	5	5	6
def_lost	0	0	0	0	1

  The second table shows that we have a definitely lost packets if $L >=$ NLOSS$+$ 1 and if TXW_SIZE $>$ BUFFER_LENGTH $+ 1$. Losses occur only if we reach L $\geq$ NLOSS $+ 1$, so BUFFER_LENGTH $\geq$ NLOSS $+ 1$.

- *Result 3:* relation for CH_PERIOD $>$ SND_PERIOD.

  If we test our model for CH_PERIOD $>$ SND_PERIOD, i.e, (CH_PERIOD$=$ 2 SND_PERIOD, SND_PERIOD$=$10,CH_PERIOD$=$20), than we obtain following results:

  1. queue L grows beyond every limit,
  2. def_lost is at most NLOSS.

6 Conclusion

Analysis and verification of parametrized models is difficult because verification problem is, in general, undecidable. In this paper, we showed our experience and results of synthesis of the parameters for PGM protocol. We discovered the following sources of complexity that prevent the tool to finish the verification:

The number of clocks and counters. The number of clocks and counters in the model can cause that reachability analysis does not terminate. One of the suggestion for dealing with it is to refine the model and abstract out some of the clock or counter variables.

Parameters. Parameters form another type of complexity. To speed up verification it is usefull to set strict initial constraints (bounds) on parameters that limit the size of the generated state space. For instance, by setting parameter CH_PERIOD > 0 the tool is prevented to explore states where CH_PERIOD < 0.

Nonlinear relation between parameters. During the analysis PGM protocol we discovered a big issue concerning parameters that are related non-linearly. Current tools and verification techniques cannot solve this problem. As a solution we propose to instantiate parameters which are non-linear. For instance, we introduced substitution of SND_PERIOD$=$ 2 * CH_PERIOD.

Tool	Formal specification	Data structure	Params	Acceleration	Notes
HyTech	hybrid automata	polyhedra	yes	not very good	problem with termination
Uppaal	extended timed automata	DBMs	no	yes	includes simulator, graphical interface
TReX	extended timed automata	PDBMs	yes	well implemented	generates symbolic reachability graph

Table 1: Features of verification tools

Analysis does not terminate. It is not surprising when analysis does not terminate or crashes because of the lack of memory. Timed systems with parameters are generally not decidable. The termination of the analysis is sensitive to several aspects. In our case study we synthetize following recommandations:

– Explicit on-the-fly verification. This was extremely useful in verification with HyTech.

– Analysis of traces. We used a symbolic reachability graph - at least a part of it - to find a beginning of the non-termination. By setting the initial constraints on parameters we can refine the model and narrow the state space. It may help the verification to terminate.

We worked with three different tools - Uppaal, HyTech and TReX. In combined analysis using these tools we were able to find constraints on the parameters that satisfied desired property - the full recovery property. We proved that for $\texttt{SND\_PERIOD} \geq \texttt{CH\_PERIOD} \wedge \texttt{TXW\_SIZE} \geq \texttt{RATE} + \texttt{BUFFER\_LENGTH}$ the property is satisfied. In comparison with previous work [BS03] our result was obtained automatically.

Analysis using Uppaal helped us to visually describe our model and simulate its behaviour. We used its verifier to prove the full recovery property for a model with instantiating parameters. Verification of deadlock detection proved the consistancy of the model. For parametric analysis we used HyTech and TReX. HyTech had problems with termination so we verified only the negation of the property and detected configurations that violate that property. This covers two cases - partial recovery and no recovery. Using TReX we obtained a full graph of symbolic configurations and observed relations between parameters. We synthetized parameters for all three cases - full recovery, partial recovery and no recovery. The results were consistent with those obtained manually in previous work, and those obtained using HyTech and Uppaal.

In Table 1, a brief comparison of verification tools that where used for parametric verification of PGM protocol is shown.

Acknowledgements

The work presented in the paper was made at laboratory LIAFA under supervision of professor Ahmed Bouajjani and with collaboration of Mihaela Sighireanu from the same institute. The author is grateful for their guidance and contribution. The work was a part of ADVANCE project supported by the European Commission (FET project ADVANCE, contract No. IST-1999-29082).

References

[AAB00] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In E.A. Emerson and A.P. Sistla, editors, *Proceedings of the 12th CAV*, volume 1855 of *LNCS*, pages 419–434. Springer Verlag, July 2000.

[ACHH93] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In R.L.Grossman, A.Nerode, A.P.Ravn, and H.Rischel, editors, *Hybrid systems*, volume 736 of *LNCS*, pages 209–229. Springer Verlag, 1993.

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AHV93] R. Alur, T.A. Henzinger, and M.Y. Vardi. Parametric real-time reasoning. In *ACM Symposium on Theory of Computing*, pages 592–601, 1993.

[BBP02] B. Bérard, P. Bouyer, and A. Petit. Analysing the pgm protocol with uppaal. In P. Pettersson and W. Yi, editors, *Proceedings of the 2nd Workshop RT-TOOLS, Copenhagen (Denmark)*, August 2002.

[BCALS01] A. Bouajjani, A. Collomb-Annichini, Y. Lackneck, and M. Sighireanu. Analysing fair parametric extended automata analysis. In Patrick Cousot, editor, *Proceedings of Static Analysis Symposium*, volume 2126 of *LNCS*, pages 335–355. Springer Verlag, July 2001.

[BCAS01] A. Bouajjani, A. Collomb-Annichini, and M. Sighireanu. Trex: A tool for reachability analysis of complex systems. In *Proceedings of CAV*, volume 2102 of *LNCS*, pages 368–372. Springer Verlag, June 2001.

[BFG$^+$00] M. Bozga, J.-C. Fernandez, L. Girvu, S. Graf, J.-P. Krimm, and L. Mounier. If: A validation environment for times asynchronous systems. In E.A. Emerson and A.P. Sistla, editors, *Proceedings of the 12th CAV*, volume 1855 of *LNCS*, pages 543–547. Springer Verlag, July 2000.

[BL02] B. Boigelot and L. Latour. *ADVANCE Project Deliverable Report*, chapter Verifying PGM with infinitely many packets. LIAFA, 2002.

[Bry86] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35-8:677–691, 1986.

[BS03] Marc Boyer and Mihaela Sighireanu. Synthesis and verification of constraints in the pgm protocol. In Stefania Gnesi, editor, *Proceedings of International FME Symposium*, pages 264–281, September 2003.

[CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking.* MIT Press, 1999.

[Dil89] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Proceedings of the 1st CAV*, volume 407, pages 197–212, 1989.

[God90] P. Godefroid. Using partial orders to improve automatic verification methods. In *Proceeding of the 2nd Workshop on Computer-Aided Verification*, volume 531 of *LNCS*, pages 176–185, 1990.

[HHWT95] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to HYTECH. In *Proceedings of TACAS*, volume 1019 of *LNCS*, pages 41–71. Springer Verlag, 1995.

[HHWT97] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1(1):110–122, 1997.

[McM92] K. McMillan. Using unfolding to avoid the state explosion problem in the verification of asynchronous circuits. In *Proceeding of the fourth international Workshop on Computer-Aided Verification, CAV '92*, volume 663 of *LNCS*, pages 164–177, 1992.

[Pel94] D. Peled. Combining partial order reduction with on-the-fly model-checking. In *Proceeding of the 1994 Workshop on Computer-Aided Verification*, volume 818 of *LNCS*, pages 377–390, 1994.

[PL00] P. Pettersson and K.G. Larsen. Uppaal2k. *Bulletin of the European Association for Theoretical Computer Science*, 70:40–44, February 2000.

[SFC$^+$01] Tony Speakman, Dino Farinacci, Jon Crowcroft, Jim Gemmell, Steven Lin, Dan Leshchiner, Michael Luby, Alex Tweedly, Nidhi Bhaskar, Richard Edmonstone, Todd Montgomery, Luigi Rizzo, Rajitha Sumanasekera, and Lorenzo Vicisano. PGM reliable transport protocol specification. RFC 3208, IETF, Decembre 2001. 111 pages.

[Val90] A. Valmari. A stubborn attack on state explosion. In *Proceeding of the 2nd Workshop on Computer-Aided Verification*, volume 531 of *LNCS*, pages 156–165, 1990.