B. Bérard and L. Sierra

Comparing verification with HyTech, Kronos and Uppaal on the railroad crossing example

Research Report LSV–00–2, Jan. 2000

Laboratoire Spécification et Vérification

Ecole Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

http://www.lsv.ens-cachan.fr/Publis/
Research Report LSV-00-2, Lab. Spécification et Vérification, CNRS & ENS de Cachan, France, Jan. 2000

Comparing verification with HyTech, Kronos and Uppaal on the railroad crossing example

Béatrice Bérard[1] and Luis Sierra[2]

[1] LSV, CNRS URA 2236, ENS de Cachan, 61 av. du Prés. Wilson,
F-94235 Cachan Cedex, France
berard@lsv.ens-cachan.fr
[2] Instituto de Computación, Facultad de Ingeniería,
Universidad de la República, Montevideo, Uruguay
sierra@fing.edu.uy

Abstract. Timed finite automata, introduced by Alur and Dill, are one of the most widely studied models for real-time systems. We focus in this paper on the practical issue of verification for such systems. More precisely, we investigate the comparative performances of three well known verification tools : HyTech, Kronos and Uppaal.

1 Introduction

Over the last few years, an extensive amount of research has been devoted to the formal verification of reactive systems. A now classical approach is *model-checking* ([QS82,CES86]), whose aim is to answer the question "does $\mathcal{A}$ satisfy φ ?", for a model $\mathcal{A}$ of a system and a formula φ. While this method could initially be applied only to finite state systems, it was successfully extended to systems where an explicit notion of time appears both in $\mathcal{A}$ and φ (see e.g. [ACD93,HNSY94]). The main practical limitation of classical algorithms was the size of the model, which grows exponentially with the number of parallel components. Adding time to such systems increases the algorithmic complexity of their analysis. Nevertheless, new symbolic methods have been investigated ([AHH96,DY96,TY96,BTY97,LLPY97,AnL98]), leading to the development of real-time model-checking tools. For instance in HyTech ([HHW97]), sets of states are represented as convex polyhedra leading in the general case to complex decision procedures. In Kronos ([Yov97]) and Uppaal ([LPY97]), other efficient symbolic data structures and (possibly on-the fly) algorithms have been implemented. The power and efficiency of these tools are illustrated in numerous case studies.

The practical issues of verification. Although many examples handled by these tools were "toy" problems, designed to show how they can be used, several practical studies proposed in an industrial framework have also been solved with

significant results. It seems then that, today, some transfer for industrial purposes has been made feasible. For this reason, quantitative comparison between existing tools about computation time and size limitation becomes both relevant and necessary. A first attempt to compare the performances of HyTech, Kronos, Uppaal and Epsilon ([ČGL93]) was proposed in [LPY95], using a simplified version of Fischer's mutual exclusion protocol. However, no details were given regarding the methodology or the modelization and synchronization problems.

Unfortunately, many problems arise when such a comparison must be done. First of all, up to now, there is no uniform framework for the description of timed systems: each tool has its specific way for representing the models. For instance, the three tools HyTech, Kronos and Uppaal use different extensions of the basic model of timed automata ([AD94]): linear hybrid systems ([ACH+95]), timed automata with integer variables or multilabels. As a consequence, even if similar examples have sometimes been analyzed, they are usually based on different models, designed to fit the specification language. Furthermore, there is a surprisingly striking variety in the synchronization methods (binary or n-ary synchronization, based on single or multi labeling) proposed by the tools. Since the systems are usually given by interacting components, this is a real difficult point. The second problem comes from the language used to specify the properties. In this area as well, there is no standard, so that a given tool can use such particular idiom from the temporal logic framework or even an algorithmic language. Finally, verification methods themselves can be different: either based on *backward* or *forward analysis*, or based on so-called *on the fly* techniques.

Contribution of the paper. In this work, our purpose is to explore in a more systematic way how to compare the practical performances of the three tools mentioned above: HyTech, Kronos and Uppaal. For this, we first felt it necessary to identify a large set of features directly available in these tools, so that a common framework can be used. It then becomes possible to describe a model $\mathcal{A}$ and a formula φ with specifications as near as possible.

The second step is to find a significative example, but simple enough to be handled by the restrictive modeling power. We chose the Railroad Crossing case study, first proposed as a benchmark in [HJL93,HL94] and already largely used under different versions to illustrate verification of timed systems (see e.g. [AD94,HHW95,AB96]). The model proposed here can be directly analyzed by the three tools and its size, which is measured as the number of components and clocks, grows with the number n of trains. Once this system has been specified in a common framework, it is translated into the specification language adequate for each tool. This translation was implemented in Haskell ([Pet96]), allowing us to generate and modify different instances of the problem with the certitude that every testing would be done on the same model.

Finally, we consider the usual safety property for this problem: whenever a train is inside the crossing, the gate must be closed. We show how this property is expressed in the different tools and we compare the backward and forward

analysis used by HyTech and Kronos, as well as the on the fly approach of Kronos and Uppaal.

2 Specification of timed systems

The family of timed models is built according to a general rule, associating a set of variables with an untimed transition system. The resulting transition system is thus based on a finite graph, whose vertices represent control locations of the system. In the general framework of hybrid system, the (real) values of the variables increase continuously with time in the locations and can be changed by the transitions through discrete jumps. It is possible in HyTech to specify linear hybrid automata, in which variables have a constant rate in a given location. In Kronos and Uppaal, the model is based on timed automata: the real variables are clocks (with rate 1), which increase exactly at the same rate as time. Although it is possible with Uppaal to handle integer variables, there is no direct way to do so in Kronos. The solution usually consists in coding these integer variables in the locations. Finally, although some transformations from linear hybrid automata to timed automata are known, they have not been fully integrated into these tools. This is the reason why we chose to restrict ourselves to standard timed automata in this work.

2.1 Basic features

Clock constraints. The set of clock constraints over a set X of clocks is written $\mathcal{C}(X)$ and consists of boolean combinations of atomic constraints of the form $x \ \# \ c$, where x is a clock, $c \in \mathbb{N}$ is a constant and $\#$ is a binary operator in $\{<, <=, =, >=, >\}$.

A clock valuation $v : X \mapsto \mathbb{R}_+$ is a mapping that assigns to each clock x a time value $v(x)$. The set of all clock valuations is $\mathbb{R}_+^X$. Writing $v \models \varphi$ when valuation v satisfies the clock constraint φ, this constraint can be viewed as the subset of valuations v such that $v \models \varphi$. If t is an element of $\mathbb{R}_+$ and α a subset of X, the valuations $v + t$ and $v[\alpha \leftarrow 0]$ are defined respectively by $(v+t)(x) = v(x) + t$, for each clock x in X and $(v[\alpha \leftarrow 0])(x) = 0$ if $x \in \alpha$, $v(x)$ otherwise.

Timed automata. A timed automaton is a tuple $\mathcal{A} = (\mathcal{L}, X, Q, T, I)$, where $\mathcal{L}$ is a finite set of transition labels, X is a finite set of clocks, Q is a finite set of locations, $T \subseteq Q \times [\mathcal{C}(X) \times \mathcal{L} \times \mathcal{P}(X)] \times Q$ is the set of transitions, and $I : Q \mapsto \mathcal{C}(X)$ associates with each location a clock constraint, called an *invariant*. For a transition $(q, \varphi, L, \alpha, q')$, φ is a clock constraint called a *guard*, L is a transition label and α is a subset of X of clocks to be reset. We may also write the transition as $q \xrightarrow{\varphi, L, \alpha} q'$.

The semantics of a timed automaton $\mathcal{A}$ are described by the transition system $\mathcal{T}_\mathcal{A} = (S, \mapsto)$. The set S of states is the subset of $Q \times \mathbb{R}_+^X$ containing the elements

(q, v) such that $v \models I(q)$. The relation $\mapsto$ is defined on S by one of the two conditions below:

1. the system can execute a time-step $(q, v) \xrightarrow{t} (q, v + t)$, meaning that t units of time elapse in location q, if for all t', $0 \leq t' \leq t$, $v + t' \models I(q)$.
2. if there exists a transition in T, $q \xrightarrow{\varphi, L, \alpha} q'$, $\varphi \in \mathcal{C}(X)$, such that $v \models \varphi$, the system can also execute an action-step $(q, v) \xrightarrow{L} (q', v')$, such that $v' = v[\alpha \leftarrow 0]$.

Note that we do not consider the special constraint *urgent*, which can be added on transitions to forbid time-steps from some states of the system. This is a very useful notion, available in the three tools. However, it does not appear in the example and we do not feel it is a crucial part of the comparison.

Atomic propositions and location names. In KRONOS, a set of atomic propositions is associated with each location. A less general method is used in HYTECH and UPPAAL: the name of a particular location is considered as the only atomic proposition in this location. Except for a special atomic proposition INIT, needed in KRONOS (as explained later in Section 3), we will keep hereafter the restricted view for atomic propositions.

2.2 Synchronization of timed automata

Like in most verification tools, a system is described by the parallel composition (or product) of a tuple of timed automata, representing its components. A location of the product is a tuple of locations from the components. When the components do not interact, the automata move independently from each other. Interactions are modeled by the synchronous execution of transitions, according to a specified synchronization function. This function associates with a tuple of labels from the components, a resulting label for the global system. Within the reduced framework presented above, some significative differences between the tools remain, concerning the synchronization functions. We briefly describe the specific methods used in the three tools and propose a restriction which allows us to keep the specifications as near as possible.

Transition labels. There are two kinds of actions: observable actions have a label in a finite set Σ, and internal ones usually have no label. To define synchronization, we represent them here by a single new symbol $*$ not in Σ. In HYTECH and KRONOS, observable actions are simply elements of Σ, while in UPPAAL, they can only be emissions or receptions. In this case, Σ is of the form $Ch! \cup Ch?$, where Ch is a set of *channels*. Emission on channel a is written $a!$ and reception from channel a is written $a?$.

In HYTECH and UPPAAL, the set of labels is $\mathcal{L} = \Sigma \cup \{*\}$, while in KRONOS, a transition can have multiple labels: the corresponding set is $\mathcal{L} = \mathcal{P}_+(\Sigma) \cup \{*\}$, where $\mathcal{P}_+(\Sigma)$ is the power set of Σ minus the empty set.

Note that this notion of multilabels is not to be interpreted as in standard automata theory, where it is used as an abbreviation for multiple transitions between a given pair of locations. Although it is possible to simulate multi-labeling in HyTech and Uppaal at the price of increasing the number of transitions, the transformation of a Kronos automaton $\mathcal{A}$ into an ordinary one does not depend only on $\mathcal{A}$ itself, but also on the other components.

To keep a direct specification, we only retain single labeling, reducing the features of Kronos. In order to be compatible with Uppaal, we must also restrict both HyTech and Kronos to binary synchronization. In the formal descriptions, it is then possible to omit the ! and ?, which are clear from the context. With this convention, we may use for a whole system only one set Σ, containing all synchronization labels, and $*$ for internal actions.

Synchronization function. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two timed automata, with a common set X of clocks, a common set $\mathcal{L} = \Sigma \cup \{*\}$ of transition labels, and let 0 be a new symbol, representing no action. The synchronization function is a (partial) function f from $(\mathcal{L} \cup \{0\}) \times (\mathcal{L} \cup \{0\})$ into a set $\mathcal{L}'$ of labels, such that $f(0,0)$ is never defined. Let L_1, L_2 be labels in $\mathcal{L}$ and $q_1 \xrightarrow{\varphi_1, L_1, \alpha_1} q_1'$, $q_2 \xrightarrow{\varphi_2, L_2, \alpha_2} q_2'$ be transitions from $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. The composition rules for timed automata are the following:

1. if $f(L_1, L_2) = L$,
   then the resulting transition is $(q_1, q_2) \xrightarrow{\varphi_1 \wedge \varphi_2, L, \alpha_1 \cup \alpha_2} (q_1', q_2')$.
2. if $f(L_1, 0) = L$,
   then the transition in the product is $(q_1, q_2) \xrightarrow{\varphi_1, L, \alpha_1} (q_1', q_2)$, i.e. a move from $\mathcal{A}_1$ alone.
3. of course, there is a symmetric rule for moves from $\mathcal{A}_2$ when $f(0, L_2) = L$.

With the restrictions proposed above, the synchronization functions of the three tools are identical and yield an associative product. This function is defined by $\mathcal{L}' = \mathcal{L}$ and:
$f(a,a) = a$, if $a \in \Sigma$ and $f(*,0) = f(0,*) = *$
The operation is denoted by $\otimes$ and the resulting automaton by $\mathcal{A}_1 \otimes \mathcal{A}_2$. This product is built when backward or forward analysis are performed. The corresponding transition system $\mathcal{T}_{\mathcal{A}_1 \otimes \mathcal{A}_2}$ can also be obtained directly from $\mathcal{T}_{\mathcal{A}_1}$ and $\mathcal{T}_{\mathcal{A}_2}$ by synchronizing time-steps $\xrightarrow{t}$ for a given t, and action-steps $\xrightarrow{L_1}$ and $\xrightarrow{L_2}$ with rules consistent with points 1, 2 and 3 above. When on the fly analysis is performed, this latter method is used.

As an illustration of the simplified model, we give a specification of the well known Railroad Crossing example.

2.3 Specifying the Railroad Crossing

The Railroad Crossing system consists of n trains, a gate and a controller. The common model proposed here is represented in Figures 1, 2 and 3 below.

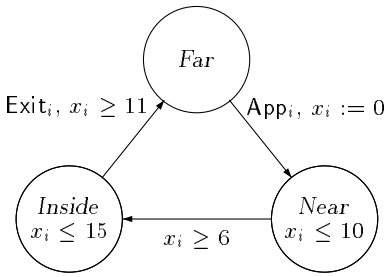


Fig. 1. A generic train

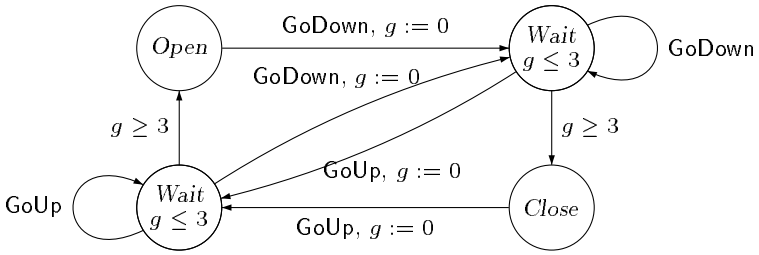


Fig. 2. The gate

The train starts moving long away from the railroad crossing. Upon arrival, it synchronizes with the controller through the label App$_i$. The train leaves the crossing synchronizes on Exit$_i$. Upon reception of these signals, the controller synchronizes with the gate by sending the corresponding GoDown or GoUp signals. As already mentioned, it is not possible in our restricted model to use a variable implementing a counter. Therefore, the number of trains currently inside the crossing must be represented by locations $Idle, C_1, \ldots, C_n$ of the controller, with an additional $Error$ location. The corresponding timed automata are called respectively `train1`, `train2` ..., `gate` and `control`, and for $n = 2$, the complete system is the synchronized product: `train1` $\otimes$ `train2` $\otimes$ `gate` $\otimes$ `control`.

This specification will be safe if temporal restrictions prevent the train to be inside the crossing while the gate is not closed. The complexity of the system grows with the number of clocks and states involved. The following table summarizes the size of the product (when needed) when increasing the number of trains.

Number of trains	1	2	3	4	5	6	7
Number of states	12	28	68	180	508	1484	4404
Number of transitions	20	64	216	764	2716	9528	32864
Number of clocks	3	4	5	6	7	8	9

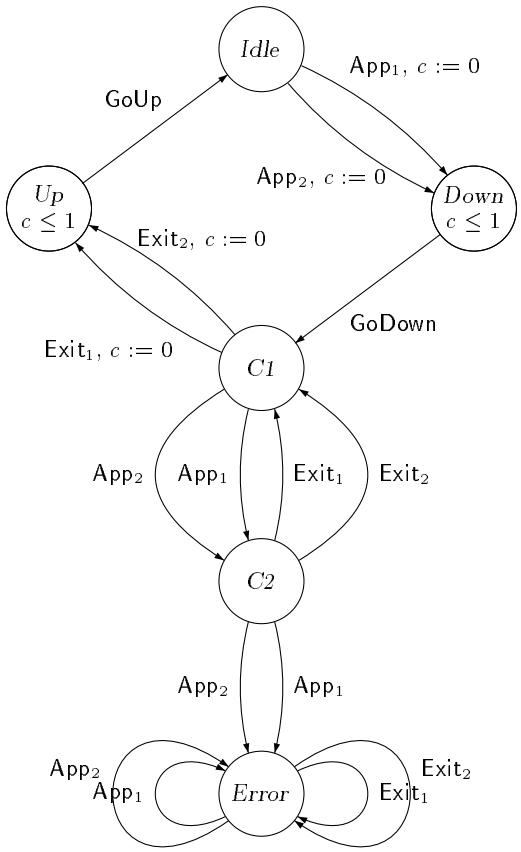


Fig. 3. The controller for the case $n = 2$

3 Verification

In the previous section we have shown a common restricted model for the specification of our problem. Now we should obtain a common description language for the safety property to prove. This is much harder to do, because the tools studied here use different approaches on how to express or prove properties. The two main points we want to emphasize are the following.

A temporal logic vs. an algorithmic language. The most important difference is the language used to specify properties. Many important system properties find a natural expression in temporal logic ([AH92,Eme90]), which was first applied to untimed systems. Among the real-time extensions of temporal logic, allowing "quantitative" temporal reasoning, TCTL [ACD93] is well suited for expressing properties of models based on timed automata.

KRONOS uses complete TCTL, while UPPAAL uses a fragment of TCTL, where quantification may only appear at the beginning of a formula. On the other hand, HYTECH handles an algorithmic language built up from commands that manipulate sets of states.

This means that all TCTL formulas (including liveness) can be expressed by KRONOS, while in HYTECH and UPPAAL, the expressive power is restricted to properties based on reachability analysis.

On the fly vs. forward/backward analysis. The other significant point concerns the way verification is related to composition. In a system with several components, the properties must be checked against the parallel composition (synchronized product) of these components. This product may be built completely before the verification takes place, or it may be partially obtained with the on the fly technique, i.e. while verification advances. UPPAAL implements reachability analysis with this last method.

HYTECH must build the entire system before the verification is performed. In this way, it can examine all executions from the initial state, looking for a system violation (forward analysis), or go backwards from some "bad" state trying to find the starting point of the system in an acceptable run (backward analysis).

Finally, KRONOS allows on the fly analysis for reachability properties, as well as forward and backward analysis. In the sequel we show the results obtained for all these verification techniques.

3.1 HYTECH: the algorithmical approach

HYTECH provides an analysis-command language to write simple programs for performing tasks based on reachability analysis. In this language we declare variables for sets of states, called *regions*, and a sequence of iterative command statements follows. For instance, the code below expresses the safety property in the railroad crossing system, when dealing with two trains.

```
1     var init, cur, bad: region;
2     init := loc[gate] = open & loc [control] = idle &
3             loc [train1] = far1 & loc [train2] = far2 &
4             g = 0 & c = 0 & x1 = 0 & x2 = 0;
5     bad := (loc [train1] = inside1 | loc [train2] = inside2) &
6             ~loc [gate] = close;
7     cur := reach forward from init endreach;
8     if empty (cur & bad)
9             then prints "Ok";
10            else prints "Nok";
11    endif;
```

The first line declares three regions. Region `init` represents the initial state of the whole system: the gate is open, the controller location is idle, all the trains are far away from the crossing, and all the clocks are reset. Line `5` characterizes the states that violate the safety requirement: some train is inside the crossing while the gate is not closed. Line `7` assigns to `cur` the set of states reachable from the initial state. In other words, this statement computes $Post^*$(`init`). The system is safe if the intersection between the set of reachable states and the region `bad` is empty. Note that atomic propositions represented by location names are expressed in HyTech by expressions like `loc[gate] = open`.

To verify the system using backward reachability we must change lines `7` and `8` by the following:

```
7'    cur := reach backward from bad endreach;
8'    if empty (cur & init)
```

3.2 Kronos and Uppaal: the temporal logic approach

TCTL is built up from a set *PROP* of atomic propositions and predicates on time by boolean connectives, a testing operator ($\in$) for clocks, and the temporal operator $\forall\Box$ (*always*). Recall that, as explained in section 2.1, atomic propositions are restricted here to location names. In the sequel, we consider the fragment of TCTL defined by the following grammar:

$$\varphi ::= p \mid x \in I \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \Rightarrow \varphi_2 \mid \forall\Box_I \varphi$$

where $p \in PROP$, $x \in X$ and I is a rational interval.

Intuitively, $\forall\Box_I\varphi$ means that every run satisfies φ in the whole interval I, that is, φ is an invariant of the system during I. The propositional part of the logic has the usual semantics.

Kronos uses full TCTL, while Uppaal restricts the use of temporal operators to the beginning of the formulas. In Kronos, invocation of the model checker will define (by an option) whether forward or backward analysis, or even on the fly analysis, will be performed.

Recall that the property to check states that for every run of the system either the gate will be closed or there will be no trains inside the crossing. If we

consider the special atomic proposition $\mathsf{INIT} \in PROP$, which is true whenever the system is in its initial state, the corresponding TCTL formula is:

$\mathsf{INIT} \Rightarrow \forall\Box(\mathsf{Close} \vee \neg \mathsf{Inside})$.

The use of INIT in this formula expresses that the runs to be considered are those that start from the initial state. As UPPAAL performs reachability analysis, this antecedent may be dropped. Moreover, as in HYTECH, location names in UPPAAL must be prefixed by the automaton name. The correct UPPAAL expression for the property to prove becomes thus:

$\forall\Box(\mathtt{gate.Close} \vee \neg(\mathtt{train1.Inside} \vee \mathtt{train2.Inside})$

3.3 Benchmarking

In this section we present the results obtained in testing the tools, with version 1.04*a* for HYTECH, 2.4*y* for KRONOS and 2.17 for UPPAAL. All the tests were carried with a Solaris system running on an Ultra Sparc workstation with 64M RAM.

The safety property was checked with backward and forward analysis in HYTECH and KRONOS, and using the on the fly approach for both KRONOS and UPPAAL. In the tables below we show the CPU time used while executing instructions of the tool (*user time*) as well as the CPU time used by the system on behalf of the calling process (*system time*).

Required inputs. Different inputs are required by the tools. HYTECH asks for only one file including the components of the system and the code, to perform the analysis. UPPAAL and KRONOS require one file with the system and another one with the logical formula to test. When on the fly analysis is not invoked, KRONOS restricts the system to be only one automaton, so that the parallel composition must be done in a preliminary step from the files containing the separate components.

The following table indicates the time needed to build the entire system from its components in KRONOS. We add this time to the KRONOS performance when backward and forward analysis are considered.

KRONOS **composition**							
Number of trains	1	2	3	4	5	6	7
user time	0.03	0.04	0.13	0.44	1.75	8.79	52.46
system	0.02	0.06	0.10	0.14	0.18	0.56	2.28
total	0.05	0.10	0.23	0.58	1.93	9.35	54.74

Backward and forward analysis. The backward analysis worked fine for both tools up to seven trains. However, the forward analysis presented different problems. HYTECH could not compute the set of reachable states for a size greater than two. On the other hand, KRONOS solved only the simplest problem using a breadth first search, while depth first search lead to a performance comparable to the backward analysis.

The following tables summarizes our results.

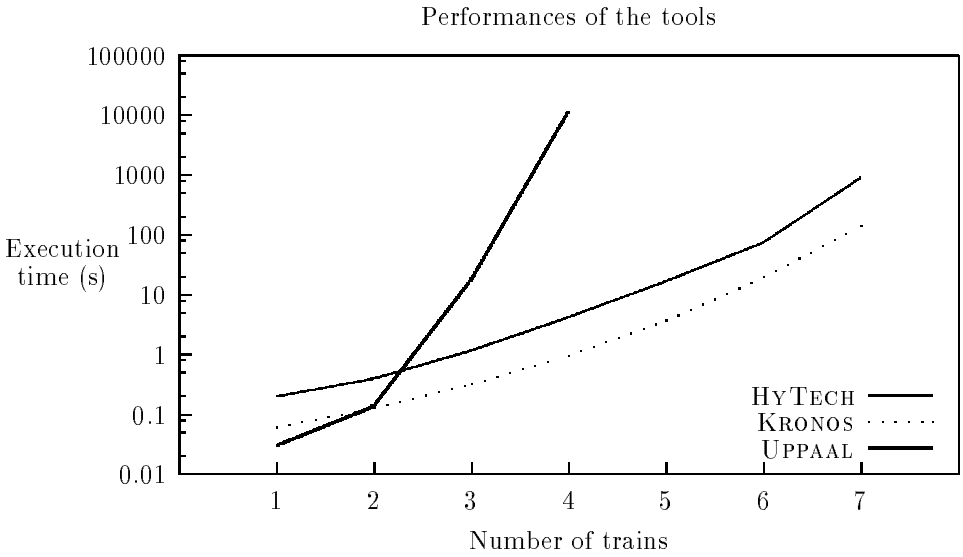
Backward analysis							
Number of trains	1	2	3	4	5	6	7
HyTech user time	0.11	0.36	1.14	4.22	16.78	76.20	913.85
system	0.09	0.04	0.06	0.08	0.25	0.85	19.04
total	0.20	0.40	1.20	4.30	17.03	77.05	932.89
Kronos user time	0.00	0.03	0.08	0.37	1.79	10.37	86.71
system	0.01	0.00	0.01	0.00	0.01	0.07	1.57
total	0.06	0.13	0.32	0.95	3.73	19.79	143.02

Forward analysis							
Number of trains	1	2	3	4	5	6	7
HyTech user time	0.13	0.37	-	-	-	-	-
system	0.04	0.04	-	-	-	-	-
total	0.17	0.41	-	-	-	-	-
Kronos user time	0.02	0.04	0.10	0.41	1.88	10.87	87.67
system	0.04	0.04	0.02	0.04	0.08	0.19	0.78
total	0.11	0.18	0.35	1.03	3.89	20.41	143.19

On the fly analysis. Kronos and Uppaal allowed us to verify the safety property on the fly. Kronos could analyse successfully only the simplest cases, while Uppaal solved the problem up to four trains. Moreover, Uppaal verified the system two times faster than Kronos. We can see that, on the whole, the values obtained are greater than those obtained with backward analysis. Thus, for our problem, on the fly analysis appears as the weakest strategy employed. Maybe this is a consequence of this system being strongly synchronized.

On the fly analysis				
Number of trains	1	2	3	4
Kronos user time	0.03	0.30	-	-
system	0.06	0.07	-	-
total	0.09	0.37	-	-
Uppaal user time	0.02	0.13	17.99	11674.40
system	0.01	0.01	0.02	0.40
total	0.03	0.14	18.01	11674.80

The following graph compares the performance of backward analysis for HyTech and Kronos and on the fly analysis for Uppaal.



4 Conclusion

Different tools have been developed for a few years by academic teams to verify properties of real time systems. In this work we compared the performances of HyTech, Kronos, and Uppaal, using as testing example the now classical problem of the Railroad Crossing. This problem was set in a common framework, and scaled until the tools could not solve it because of the size. The complexity of timed systems analysis depends on the number of clocks and states, and our approach was sensitive to these measures.

We compared backward and forward analysis in HyTech and Kronos, and on the fly analysis in Kronos and Uppaal. Kronos showed a better perfomance than HyTech in the backward analysis. In the forward analysis case, HyTech could not manage the problem in a satisfactory way. Kronos showed the same inefficiency when using a breadth first search strategy, but overcame this with a depth first search approach. On the fly analysis was more efficient with Uppaal than with Kronos. Nevertheless, its performance was inferior to backward analysis, with respect both to time and maximum size of problem solved. This last point came as a surprise.

Many points remain to be tested, regarding the comparative efficiency of real-time analysis, the most subjective feature being the ergonomy of the tools. Among the practical issues raised by the present study, the first one is the question of possible influence of synchronization on the verification methods (on the fly or backward). These methods should now be compared on systems with

tight vs. loose synchronization, as well as on examples where synchronization produces almost sequential vs. very parallel systems.

In the area of temporal logic, other properties should be tested, like for instance bounded response properties, which play an important role in verification issues. However, this is more difficult, because such a property can be directly expressed in KRONOS, but not in HYTECH or UPPAAL. In these tools, the usual solution consists in introducing an additional automaton, sometimes called an "observer", thus transforming bounded liveness into safety. But this would mean departing from our requirement to use identical models.

References

[AB96] J. Armstrong and L. Barroca. Specification and verification of reactive system behaviour: The railroad crossing example. *Real-Time Systems*, 10(2):143–178, 1996.

[ACD93] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, Pei-Hsin Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AH92] R. Alur and T. A. Henzinger. Logics and models of real time: A survey. In *Real-Time: Theory in Practice, Proc. REX Workshop, Mook, NL, June 1991*, volume 600 of *Lecture Notes in Computer Science*, pages 74–106. Springer, 1992.

[AHH96] R. Alur, T. A. Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.

[AnL98] L. Aceto, A. Burgue no, and K. G. Larsen. Model checking via reachability testing for timed automata. In *Proc. Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98), Lisbon, Portugal, March 1998*, volume 1384 of *Lecture Notes in Computer Science*, pages 263–280. Springer, 1998.

[BTY97] A. Bouajjani, S. Tripakis, and S. Yovine. On-the-fly symbolic model-checking for real-time systems. In *Proc. 18th IEEE Real-Time Systems Symposium (RTSS'97), San Francisco, CA, USA, Dec. 1997*, 1997.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[ČGL93] K. Čerāns, J. C. Godskesen, and K. G. Larsen. Timed modal specifications – theory and tools. In *Proc. 5th Int. Conf. Computer Aided Verification (CAV'93), Elounda, Greece, June 1993*, volume 697 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 1993.

[DY96] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *Proc. 17th IEEE Real-Time Systems Symposium (RTSS'96), Washington, DC, USA, Dec. 1996*, 1996.

[Eme90] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, vol. B*, chapter 16, pages 995–1072. Elsevier Science, 1990.

[HHW95] T. A. Henzinger, Pei-Hsin Ho, and H. Wong-Toi. HYTECH: the next generation. In *Proc. 16th IEEE Real-Time Systems Symposium (RTSS'95), Pisa, Italy, Dec. 1995*, pages 56–65. IEEE Comp. Soc. Press, 1995.

[HHW97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model-checker for hybrid systems. *Journal of Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.

[HJL93] C.L. Heitmeyer, R.D. Jeffords, and B.G. Labaw. A benchmark for comparing different approaches for specifying and verifying real-timed systems. In *Proceedings of the 10th IEEE International Workshop on Real-Time Operating Systems and Software*, 1993.

[HL94] C. L. Heitmeyer and N. Lynch. The generalized railroad crossing : a case study in formal verification of real-time systems. In *Proc. 15th IEEE Real-Time Systems Symposium (RTSS'94), San Juan, Puerto Rico, Dec. 1994*, pages 120–131, 1994.

[HNSY94] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[LLPY97] Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Efficient verification of real-time systems: Compact data structure and state-space reduction. In *Proc. 18th IEEE Real-Time Systems Symposium (RTSS'97), San Francisco, CA, USA, Dec. 1997*, 1997.

[LPY95] K. G. Larsen, P. Pettersson, and Wang Yi. Model-checking for real-time systems. In *Proc. 10th Int. Conf. Fundamentals of Computation Theory (FCT'95), Dresden, Germany, Aug. 1995*, volume 965 of *Lecture Notes in Computer Science*, pages 62–88. Springer, 1995.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Journal of Software Tools for Technology Transfer*, 1(1–2):134–152, 1997.

[Pet96] J. Peterson et al. *Report on the Programming Language HASKELL. A Non-strict, Purely Functional Language*, May 1996. Available at `http://haskell.org`.

[QS82] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. Int. Symp. on Programming, Turin, Italy, Apr. 1982*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

[TY96] S. Tripakis and S. Yovine. Analysis of timed systems based on time-abstracting bisimulations. In *Proc. 8th Int. Conf. Computer Aided Verification (CAV'96), New Brunswick, NJ, USA, July-Aug. 1996*, volume 1102 of *Lecture Notes in Computer Science*, pages 232–243. Springer, 1996.

[Yov97] S. Yovine. Kronos: A verification tool for real-time systems. *Journal of Software Tools for Technology Transfer*, 1(1–2):123–133, 1997.