Verifying Periodic Task-Control Systems

Vlad Rusu*

Abstract. This paper deals with the automated verification of a class of task-control systems with periods, durations, and scheduling specifications. Such systems are translated into Periodic Hybrid Automata for verification. We show that safety, liveness, and time-bounded properties are decidable for the considered systems.

Keywords: task-control systems, periodic hybrid automata, verification, decidability.

1 Introduction

The automated verification of real-time systems has made considerable progress in the last years, since verification methods based on *timed/hybrid automata* and *Timed Computational Tree Logic* [ACH+95] were developed. A main result is that the logic TCTL is decidable on timed automata [ACD90, HNSY94]: as a consequence, verification of TCTL properties is decidable for real-time formalisms that are translatable into timed automata [NSY92, JMO93, DOY94].

But timed automata cannot express all the aspects of real-time applications, like for instance a task *preempting* another task. Indeed, preemption needs the more powerful model of hybrid automata to be expressed; but the verification of TCTL properties is in general undecidable on hybrid automata [HKPV95]. However, in [RR96] we proved the decidability of TCTL for a class of task-control systems, under the restriction that tasks are not preempted too often, such that all of them eventually terminate in bounded time. In this paper, we prove decidability of TCTL for another class of systems, without the former restriction; in exchange, the tasks are constrained to be *periodic*.

To express the task control, we introduce in section 2 a simple language with its untimed semantics (finite automata) and timed semantics (Periodic Hybrid Automata, to model the durations and periods of tasks). Section 3 presents the decidability results, and we conclude in section 4.

2 A Language for Task-Control

We define a simple language to describe a set of tasks running and preempting each other to obtain a processor for execution. When a task has been preempted,

* Laboratoire d'Automatique de Nantes (CNRS N°823, École Centrale Nantes, Univ. Nantes) 1 rue de la Noë - BP92101 44321 Nantes Cedex 3, France. E-mail Vlad.Rusu@lan.ec-nantes.fr

its execution may be later resumed at the last preemption point. Consider for instance a computer system that iterates a task `A`, that can be preempted by tasks `D` or (exclusively) `B`; and task `B` can itself be preempted by `C`. This is modeled by the automaton of figure 2, next page (for the moment, ignore the formulæ around the automaton). Task `A` runs at location **A**; task `D` can preempt `A` and it runs (while `A` is preempted) at location **D,A_pre**. Similarly, `B` runs while `A` is preempted at location **B,A_pre**, and `C` runs while `A` and `B` are preempted, at location **C,A_pre,B_pre**. This will be expressed syntactically as **loop**(`A` **preempted by** (`D` **exclusive**(`B` **preempted by** `C`))). The language syntax is :

$$\begin{aligned} T &::= Task \;\mid\; (T\ T) \;\mid\; \mathbf{loop}(T) \;\mid\; (T\ \mathbf{exclusive}\ T) \;\mid \\ &\qquad (Task\ \mathbf{preempted\ by}\ T) \\ Task &::= \mathtt{A}, \mathtt{B}, \mathtt{C} \ldots \end{aligned}$$

with the following restriction: *the relation $\prec$ on the tasks such that "*`A` $\prec$ `B` *iff* `B` *occurs in a* `A` **preempted by** *T construct", is a strict order.* So, 'higher-level' tasks can preempt 'lower-level' ones, but not the contrary. This restriction will be used for proving the decidability results in section 3.

Untimed semantics: finite automata. The untimed semantics of a task-control program is given as a finite-state automaton, generated directly while the program is top-down parsed. For this, we define a rewriting system, whose root is a one-state automaton; the whole program labels this state (fig. 1(a)). The program is parsed top down, and when a grammar rule applies, the state is expanded into a state construction depending on the rule (fig 1(b)-(e)). The new states are connected to their neighbours in the following manner: all incoming (resp. outcoming) transitions of the replaced state, have their destination (resp. origin) in the initial (resp. final) replacing states.

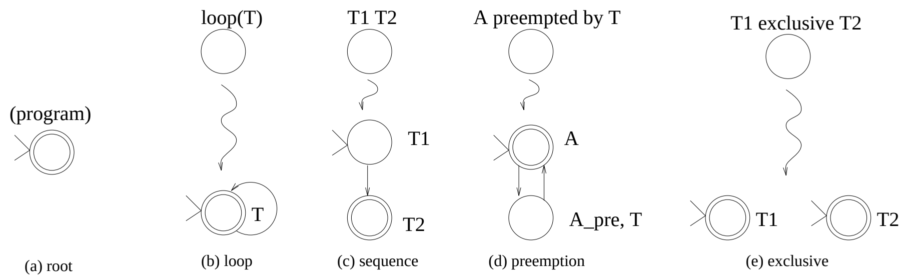

Figure 1 *Rewriting states; $>\bigcirc$ = initial, $\circledcirc$ = final state*

Timed semantics: Periodic Hybrid Automata. We now introduce in the model such quantitative features as the durations and the periods of tasks. More precisely, all tasks will be characterized by some *duration*, and tasks that can preempt other tasks will also be characterized by some *period* — time interval between two consecutive occurrences. To model this, we add to the finite automaton two sets of variables: the *duration variables* and the *period variables*. We obtain the model of Periodic Hybrid Automata.

The Periodic Hybrid Automaton (PHA) of a task-control program consists of:
— the finite automaton of the program;
— for each task $\mathtt{A}_i$ in the program, a *duration variable* a_i;
— for each task $\mathtt{A}_j$ that can preempt another task, a *period variable* α_j;
— for each location of the automaton:

• an *invariant* $a_i \leq D_{\mathtt{A}_i}$ if task $\mathtt{A}_i$ is active at that location; $D_{\mathtt{A}_i} \in \mathbb{Q}^{>0}$ is the duration of task $\mathtt{A}_i$ (there is exactly one active task at each location);

• for each duration variable a_i, a *differential law* $a_i^{'} = 1$ if task $\mathtt{A}_i$ is active at that location, $a_i^{'} = 0$ otherwise; and for each period variable α_j, a differential law $\alpha_j^{'} = 1$ if task $\mathtt{A}_j$ is active, preempted, or if it can preempt a task which is active at that location, $\alpha_j^{'} = 0$ otherwise;

— for each transition, a *guard* and a *reset*:

• for transitions that correspond to task endings, the *guard* is $a_i = D_{\mathtt{A}_i}$ if task $\mathtt{A}_i$ ends on that transition, and the *reset* applies to duration variable a_i;

• for transitions that correspond to task preemptions, the *guard* is $\alpha_j = T_{\mathtt{A}_j}$ if the transition is a preemption by task $\mathtt{A}_j$ ($T_{\mathtt{A}_j} \in \mathbb{Q}^{>0}$ is called the period of task $\mathtt{A}_j$). The *reset* applies to variable α_j, and to the period variables of the tasks that can preempt $\mathtt{A}_j$. $\square$

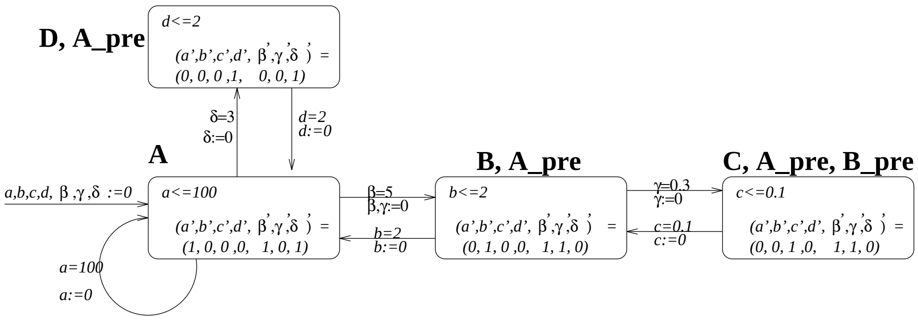

Figure 2 *Periodic Hybrid Automaton for example program.*

Figure 2 represents the PHA for program **loop**(`A` **preempted by** (`D` **exclusive**(`B` **preempted by** `C`))) in which tasks `A`, `B`, `C`, `D` have durations 100, 2, 0.1 and 2 time units, and `B`, `C`, `D` have periods 5, 0.3, and 3 time units. The duration variables are $\{a, b, c, d\}$, and period variables are $\{\beta, \gamma, \delta\}$.

A *run* of a PHA consists in letting the variables evolve at control states inside the invariants, and crossing the transitions when guards are true [ACH+95]. But unlike the previous, we shall define runs such that *period variables are frozen when they reach the period value*; and *transitions are fired as soon as possible.* These two features allow to express what happens when a task's period has been exceeded. Consider the example of figure 2 when `B` is running (at location **B, A_pre**), and suppose `B` preempted `A` more than 5 time units (`B` 's period), because `B` itself was preempted by several occurrences of `C`. When `B` completes, it should preempt `A` at once because its period is exceeded. This happens precisely because variable β was frozen at value 5 and because the transitions are urgent.

We now make more precise the semantics of PHA. Suppose the PHA has n duration variables and m period variables. A *state* is a triple $(l, \overline{a}, \overline{\alpha})$ where l is a location, $\overline{a}$ is a n-vector of real values for duration variables *that satisfy location l 's invariant*, and $\overline{\alpha}= (\alpha_j)_{j=\overline{1,m}}$ is vector of real values for period variables. For $t \geq 0$, $\overline{a}+t$ denotes the vector $(a_i+t)_{i=\overline{1,n}}$, and $\overline{\alpha}+t$ denotes the vector $(max\{\alpha_j+t, T_j\})_{j=\overline{1,m}}$ where T_j is the period of the task with period variable α_j. A *time step* of duration $t \geq 0$, denoted $(l, \overline{a}, \overline{\alpha}) \rightarrow_t (l, \overline{a'}, \overline{\alpha'})$, is defined by $\overline{a'} = \overline{a}+t$ and $\overline{\alpha'} = \overline{\alpha}+t$. A *transition step* $(l, \overline{a}, \overline{\alpha}) \rightarrow (l', \overline{a'}, \overline{\alpha'})$ consists in crossing a transition whose guard is enabled (i.e. is satisfied by $\overline{a}, \overline{\alpha}$), and resetting the values of variables according to the transition.

A *run* is a sequence of time and transition steps, such that transition steps are *urgent [BGK+96]: as soon as* a transition gets enabled, the next step consists in crossing that transition (if several transition are enabled simultaneously, one is chosen arbitrarily). The *duration of a run* is the sum of the durations of time steps, as the transition steps are instantaneous.

3 Verifying properties: symbolic analysis

The time-bounded (resp.time-unbounded) reachability properties of Periodic Hybrid Automata are of the form: *starting from an initial set of states ψ, and by remaining in an intermediary set of states ψ_1, is it* possible *to reach a final set of states ψ_2, by a run of duration at most $c \in \mathbb{Q}^{>0}$ time units (resp. of arbitrary duration).* The time-bounded (resp. time-unbounded) liveness are similar except for the modality *possible* which is replaced by *inevitable*. For example, a time-unbounded liveness property for the automaton of figure 2 is: *starting from location* **A** *with all variables equal to 0, is it inevitable that task* `A` *terminates.*

These properties can be verified by *symbolic analysis*[HNSY94, ACH+95]: infinite sets of states are represented by *symbolic states* and infinite sets of runs are represented by *symbolic runs*. A symbolic state is defined by a location and a domain of values for variables identifiable with a polyhedron in $\mathbb{R}^n$; and a symbolic run is a sequence of symbolic states, simulating all the runs that start from a given set of states and cross a given sequence of transitions. Symbolic runs can be computed algorithmically [HPR94].

We now prove that symbolic analysis terminates on PHA: we first show that time-bounded reachability is decidable; then we show that time-unbounded reachability reduces to the time-bounded version; and finally we indicate that liveness properties reduce to the previous two.

Proposition 1. *Time-bounded reachability is decidable on PHA.*
Proof. Define the cyclic runs to be the runs that cross twice some transition. Then there exists a uniform low-bound $\epsilon > 0$ for the durations of cyclic runs, since these necessarily perform a complete execution of at least one task. Then, the bounded-duration runs can be decomposed into a bounded number of cyclic runs, that cross a bounded number of transitions, which are simulated by bounded-length symbolic runs [RR96]. $\square$

Next, we will show that PHA satisfy the following *periodicity condition*: (PC) there exists $T > 0$ and a finite number N of states such that any run of the PHA meets a state among the above, in at most T time units. This property is important because if it holds, time-unbounded reachability reduces to the time-bounded version (in time $(N+1) \times T$ [RR96]), which is decidable (Prop.1).

Definition. A *remarkable point* of a task `A` is a set of values for variables of tasks greater or equal than `A`[1], at an instant when `A` is being preempted or is ending. A *total duration* of task `A` is a possible duration between the beginning and the end of `A` (including preemption time).

Proposition 2. For each task: (a) there exist a finite number of remarkable points; (b) there exist a finite number of total durations, including ∞.
Proof idea. By induction on the preemption order $\prec$ of tasks. The key point is our particular definition of runs (cf. previous page): period variables are frozen when they reach period value, and transitions are fired as soon as possible. □

Proposition 3. The periodicity condition holds for PHA.
Proof sketch. We first show (c): there exists $T > 0$ and a finite set V of reals with the property: *for each task* `A`*, there exists a task* `B` *greater or equal than* `A`*, such that when* `A` *is active, a state will be reached in time at most T, where* `B` *is active and where values of variables for tasks greater or equal than* `A` *are in V.*

Let V be the finite union of all remarkable points, T_1 the greatest finite total duration, T_2 the sum of all period values, and $T = T_1 + T_2$. Consider a task `A`. When `A` is active then either 1. it eventually terminates or 2. it never terminates. In the first case, by Proposition 2(b), it will terminate in at most T_1 time units. This ending state is a remarkable point of `A`, and at that moment, by Proposition 2(a), the values of variables for tasks greater and equal than `A` are in V.

In the second case, there exists a task `B` $\succeq$ `A`, that is preempted an infinity of times and does not terminate. We choose `B` minimal with the above property. Suppose first `B` $\succ$ `A`; then `B` runs while all tasks `A` through `B` are preempted. By construction of the PHA, starting from a state when `A` is active, one *will* reach in at most T_2 time units, a state where `B` is running, all tasks `A` through `B` are preempted, and period variables of tasks `A` through `B` are frozen at period value[2]. Moreover, since task `B` will never terminate, all tasks `A` through `B` remain preempted forever, so the duration variables of those tasks remain forever unchanged. So, starting from a state with `A` active, one will reach in at most T_2 time units a state where `B` is active, and where values of variables of tasks `A` through `B` are in V *and will remain forever unchanged.*

Next, `B` itself is preempted an infinity of times and does not terminate; but all tasks that preempt `B` terminate in at most T_1 time units (by choosing `B` minimal and Proposition 2(b)). The preemptions of `B` constitute remarkable points of `B`, in finite number by Proposition 2(a). Then, one of these preemptions is repeated an infinity of times, at least once every T_1 time units, and at the preemption instant, the values of variables for tasks greater or equal than `B` are in V.

---

[1] with regards to the preemption order $\prec$ defined with the task-control language syntax
[2] this holds because of time amount T_2, the sum of all period values

So, starting from a state where `A` is active, one will reach in at most $T = T_1 + T_2$ time units a state where `B` is active and where variables of all tasks greater or equal than `A` are in V. This proves (c) when $B \succ A$. The case $A = B$ is similar (it reduces to the previous paragraph). Note that when $B \succ A$ we have more (+): the values of variables of tasks `A` through `B` *remain forever unchanged*. We can now obtain the Periodicity Condition (PC) by applying (c+) to a a smallest task with regards to $\prec$. $\square$

Using Proposition 3, time-unbounded reachability reduces to time-bounded reachability, which is decidable (Proposition 1). Also,time-unbounded liveness reduces to time-bounded liveness, which reduces to reachability [HNSY94].

4 Conclusion

We have presented a decidability result that allows the verification of TCTL properties for a class of task-control systems with durations, periods, and preemptions. The verification model is Periodic Hybrid Automata, a variant of hybrid automata. The proof lies on the fact that the infinite timed behaviour reduces to a finite number of 'remarkable points'.

The author wishes to thank Olivier Roux, Pablo Argon, and the anonymous reviewers of HART'97 for useful comments and constructive criticism.

References

ACD90. R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proc. IEEE 5th Symp. Logic in Computer Science, LNCS*, 1990.

ACH+95. R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science B*, 137, January 1995.

BGK+96. J. Bengtsson, D. Griffioen, K. Kristoferssen, K.Larsen, F.Larsson, P.Petersson, and W.Yi. Verification of an audio protocol with bus collision using UPPAAL. In *Computer-Aided Verification, LNCS*, 1996.

DOY94. C. Daws, A. Olivero, and S. Yovine. Verifying ET-LOTOS programs with KRONOS. In *Proc. FORTE'94, LNCS*, 1994.

HKPV95. T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata. In *STOCS'95, LNCS*, 1995.

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

HPR94. N. Halbwachs, Y. E. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In *International Symposium on Static Analysis, LNCS*, 1994.

JMO93. M. Jourdan, F. Maraninchi, and A. Olivero. Verifying quantitative real-time properties of synchronous programs. In *Computer-Aided Verification, LNCS 697*, 1993.

NSY92. X. Nicollin, J. Sifakis, and S. Yovine. Compiling real-time specifications into extended automata. *IEEE Transactions on Software Engineering*, 18(9):794–804, 1992.

RR96. O. Roux and V. Rusu. Uniformity for the decidability of hybrid automata. In *Internat. Static Analysis Symposium, LNCS 1145*, pages 301–316, 1996.