Polyhedra-Based Approach for Incremental Validation of Real-Time Systems

David Doose and Zoubir Mammeri

IRIT - Paul Sabatier University - Toulouse, France

Abstract. Real-time embedded systems can be used in hightly important or even vital tasks (avionic and medical systems, etc.), thus having strict temporal constraints that need to be validated. Existing solutions use temporal logic, automata or scheduling techniques. However, scheduling techniques are often pessimistic and require an almost complete knowledge of the system, and formal methods can be ill-fitted to manipulate some of the concepts involved in real-time systems.
In this article, we propose a method that gives to the designer the advantages of formal methods and some simplicity in manipulating real-time systems notions. This method is able to model and validate all the classical features of real-time systems, without any pessimism, while guaranteeing the terminaison of the validation process. Moreover, its formalism enables to study systems of which we have only a partial knowledge, and thus to validate or invalidate a system still under design. This latest point is very important, since it greatly decreases the cost of design backtracks.

1 Introduction

Validation is a mandatory step in critical real-time systems design, and can be important even for non-critical ones. There are many ways to check the temporal behavior of a real-time systems: we can use formal methods based upon logic (LTL, CTL, etc.) or automata [3, 2, 1] (Petri nets, linear hybrid automata, etc.), or we can use scheduling techniques [7] (RMA method, etc.). Methods based on logic give precise results and have a great expressive power. They can model very specific features, but involve a strenuous designing process. Indeed, they are not initially fitted to the particular case of real-time system validation, and such notions as tasks or resources sharing do not exist in Petri nets or linear hybrid automata. Scheduling techniques have two main advantages. Firstly, they are of course completely fitted to real-time systems. Secondly, they usually have a very low complexity. However, they often are pessimistic, they require an almost complete knowledge of system parameters (tasks periods, priorities, precedences, etc.), and they give quite limited results, confined mostly to the possible schedulability of the system.

Our goal is to combine formal methods and scheduling techniques by devising a method fitted for scheduling analysis while preserving the expressivity and precision of formal methods such as linear hybrid automata. The study of linear hybrid automata solvers [10–12] led us to the use of polyhedral domains, that enable a powerful modeling and checking. This way, we devised the only method (as far as we know) that brings together the following features:

- It can model and study classical features of real-time systems (tasks, resources sharing, precedence, atomic task executions, etc.).
- It guarantees the ending of the validation process (decidability), contrary to linear hybrid automata [13, 9].
- It is non-pessimistic.
- It gives precise results on the behavior of the system: we can deduce the response times of tasks, preemptions between tasks, etc., and of course the schedulability of the system.
- It enables to validate/invalidate systems under partial knowledge. Moreover, any parameter constraint (value or linear equation on parameter value) is taken into account by the model.

The ability to check a system schedulability during its design ("under partial knowledge") can lead to important savings by avoiding backtracks. Besides, the non-pessimism implies that we can avoid system over-sizing. Lastly, since the method core works by computing all the different possible behaviors of a system, we can highlight the fact that it is similar to the use of an exhaustive set of simulations, while being far less time-consuming.

The paper is structured as follows: in section two, we introduce the notion of polyhedra and the PV-domains. In section three, we describe our method. In section four, we propose an example comparing the proposed method and a simple schedulability analysis. Finaly, we conclude in section five.

2 Notion of Polyhedra

2.1 Polyhedron

The model we propose allows to take into account any constraint based on linear inequalities. It uses polyhedra [19, 14, 15] to represent the parameters and the properties of the real-time system under construction.

Definition 1. *A* ***polyhedron*** *P is the intersection of a finite family of closed linear half-spaces of the form $\{x|ax \geq c\}$ where a is a non-zero row vector and c is a scalar constant.*

Double representation: A polyhedron has a dual representation: an implicit representation and a parametric representation. Indeed, a polyhedron P can be represented with a set of linear inequalities which represent its constraints. Its *implicit definition* is the following one: $P = \{x \mid Ax = b,\, Cx \geq d\}$ in which A and C are matrices and x, b and d are vectors. The *parametric representation* of the polyhedron is the following one: $P = \{x \mid L\lambda + R\mu + V\nu,\, \mu,\, \nu \geq 0,\, \sum \nu = 1\}$ In which the polyhedron is composed of several lines (columns of the matrix L), a convex combination of vertices (columns of the matrix V), and a combination of external rays (columns of the matrix R). Algorithms exist to compute the second representation from the first one, and reciprocally. Both representations are useful. Indeed, the implicit representation is more intuitive, so we use it to represent the different constraints of the system and the results of the validation. However, the parametric representation is more efficient to compute the operations on polyhedral domains. Thus, we will use algorithms based on the dual representation, which computes both representations simultaneously.

Polyhedral domain: We mainly use two operations on polyhedra: *intersection* and *union* of polyhedra. Contrary to the intersection, the result of a polyhedra union may not be a convex polyhedron. That is why the notion of polyhedral domain is needed.

Definition 2. *A polyhedral* ***domain*** *of dimension n is defined as $\mathcal{D} : \{i | i \subset \mathcal{Z}^n,\ i \subset \mathcal{P}\} = \mathcal{Z}^n \bigcap \mathcal{P}$ where $\mathcal{P}$ is a union of polyhedra of dimension n.*

2.2 PV-domain

We introduce a new notion that aims to create a link between the variables corresponding to the system parameters and their polyhedral representation. A **PV-domain** (Polyhedral representation of Variables) is composed of a list of variables ($\mathcal{V}$) and a polyhedral domain ($\mathcal{S}$). The variables can either be a value or a dimension of the polyhedral domain.

PV-domain operations: Adding a variable consists in adding a space dimension at the end of the polyhedral domain and adding a variable to the variables list $\mathcal{V}$ (noted: $\mathcal{D}^{'} = \mathcal{D} \oplus \{v\}$).

Removing a variable consists in removing the variable from the list and removing the corresponding dimension in the polyhedral domain $\mathcal{D}$ ($\mathcal{D}^{'} = \mathcal{D} \ominus \{v\}$).

Reducing the PV-domain consists in modifying each variable of the list representing a dimension which has a unique value in the polyhedral domain ($\forall_{v \in \mathcal{V}} \forall_{p \in \mathcal{S}},\ \{v = a\} \subset p$) by removing the corresponding dimension in the polyhedral domain and transforming the variable representing a dimension into a variable representing the value a ($\mathcal{D}^{'} = \odot(\mathcal{D})$).

The intersection (resp. union) of the PV-domain $\mathcal{D} = (\mathcal{V}, \mathcal{S})$ and $\mathcal{D}^{'}$ is noted $\cap$ (resp. $\cup$) and defined as follows: $\mathcal{D} \cap \mathcal{D}^{'} = \left(\mathcal{V}, \mathcal{S} \cap \mathcal{S}^{'}\right)$ (resp. $\cup$).

Efficiency: The complexity and thus the computation time of intersections (resp. unions) of polyhedra depends (exponentially) on their space dimension. That is why the main objective is to reduce as much as possible the space dimension of the system model. The real efficiency of the space reduction made by the PV-domain reduction operator can be shown by simulations. The simulations on figure 1 are run on one polyhedron with ten unknown dimensions and from 20 to

500 space dimensions. In each case (i.e. for each total number of dimensions), we run 500 intersections (resp. unions) with and without using the space dimension reduction. The computation time of the unions is due to the polyhedra power-set [5, 16] operations used for the representation of the disjunctions. We can see here how efficient the dimension reduction can be, but it is important to notice that the dimension reduction is not an invertible operation.

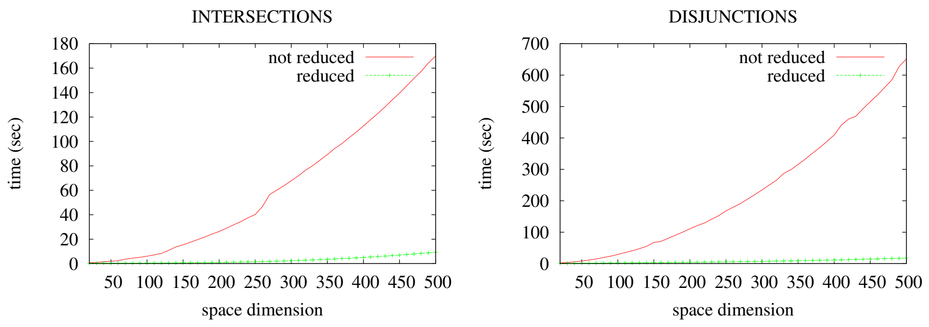

Fig. 1. PV-domain operations computation time

3 The proposed method

System parameters: Each task instance i has four *descriptive parameters*: its period (T_i), its release time (R_i), its execution time (C_i) and its absolute deadline (D_i); and two *behavioral parameters*: the time when the task instance begins its execution (b_i), and the moment when it ends its execution (e_i). The *hidden parameters* (see below) are mandatory to represent some information specific to the system. From a geometric point of view, each descriptive, behavioral or hidden parameter corresponds to a variable of the PV-domain. Last, *static parameters* are useful to take into account some specific properties of the system. The main difference between static parameters and the others is that their values are always known.

Interactions and Behavior: In the proposed model, only two interactions are needed to represent tasks instances and their evolution. Indeed, an instance can either **preempt** or **delay** another instance. The behavior of tasks instances is defined by the interactions which can either exist between the instances. In the paper we will compute *possible behaviors*. A behavior is possible for a group of instances (see below) if all the interactions characterizing this group can occur on the instances of the group, with the respect of all the constraints of the system.

Group: A group of tasks instances is characterized by:

- a list of instances with interactions (noted *interactions*);
- the first instance in interaction of the group (*first*);
- a list of context instances (noted *contexts*);
- a PV-domain representing the properties of the previous groups (i.e. the constraints of the systems before the creation of the group noted *pre_constraints*);
- a PV-domain representing the properties of the groups (*part_system*).

Each instance of the *interactions list* is at least in interaction with another instance of the list. The first instance in interaction and the number of instances of each task are useful to determine the beginning and the end of the group. The list of context instances is made of instances which, under the same constraints, are not in interaction with any instance of the list of the instances in interaction.

3.1 Method overview

Analysis of real-time systems with the proposed method has two main parts. The first part consists in taking into account the knowlegde of the system by: adding the tasks of the system, modifying the static parameters, adding the tasks constraints and adding the invariants. The second part consists in analysing the behaviors of the system by determining all the groups of interactions and analysing the schedulability of each group. At the end of the validation process the method provides the list of the situations in which the system is schedulable, and the list of the situations in which the system is not schedulable.

3.2 Static steps

The first step consists in simply adding the tasks of the system in order to determine the number of tasks.

In this first step no information about the tasks are taken into account. The second step consists in setting the static parameters of the system. Thus, we determine whether the tasks are periodic, atomic, ... We also determine the scheduler category (EDF or fixed priority). At the end of this step, the static parameters of the system and the number of tasks are known, thus we can create the first polyhedral representation of the system.

In the third static step, the designer takes into account the system information by adding constraints to the domain representing the system as previously described. For example, if the worst execution time of task τ_1 is $20\,ms$, the system is equal to $system \cap (C_1 \leq 20)$. At the end of this step, the domain representing the system contains all the information about the tasks.

In the last static step, the invariants of the system concerning the tasks are taken into account by adding the corresponding constraints. If the previously added information of the system does not respect these invariants, then the domain is empty, and the validation ends. The domain representing the system is: $system = \odot(information \cap (\forall_{\tau_i}(R_i \geq 0) \cap (C_i > 0t) \cap (D_i \geq 0)))$

3.3 Determining groups

The principle is to determine all the possible groups of instances as shown by the following algorithm:

```
groups: stack of Group ;
groups.push(Group(system)) ;
while groups.not_empty() do
    Group g = groupes.pop() ;
    if g.complete() then
        if (not g.cycle) and g.validation then
            groups.push(g.next()) ;
    else
        for each instance $\tau_i$ do
            if g.can_interact_with($\tau_i$) then
                goups.push((new Group()).add_interaction($\tau_i$)) ;
            if g.can_be_context_of($\tau_i$) then
                goups.push((new Group()).add_context($\tau_i$)) ;
```

The function *can_add_interaction* (resp. *can_be_context*) determines whether or not the instance τ_i can interact (resp. can be a context instance) in the group g. This is determined by verifying the non-emptyness of the following PV-domain representing the addition of the instance: $part\_system \cap (r_{i,j} \geq r_{first}) \cap (r_{i,j} < r_{first} + c_{tot})$ (resp. $part\_system \cap (r_{i,j} \geq r_{first} + c_{tot})$) while r_{first} represents the release time of the first instance of the group (i.e. the beginning of the group), $r_{i,j}$ the release time of the instance j of the task i and c_{tot} represents the length of the group.
The function *cycle* aims to determine if the the group g is in a behavioral cycle with another group that has already been computed. The notion of cycle is important because it guarantees the end of the validation process. A group G is in behavioral cycle with the group $G^{'}$ if the instance in interaction can correspond (verifyed with the function *mapping*) and if the constraints of the system are the same but at another moment. The function *validation* determines if the group of instances is schedulable or not. This function is described in the following subsection.

3.4 Group schedulability

In this subsection, we detail the validation of a group. The tasks instances are not linked to the corresponding tasks, thus the instances parameters are noted with lowercase letters with a single index representing the instance number.

General information and invariants: The invariants are mandatory to maintain the correctness of the model. They are added like any information about the system. The following invariants are useful in the proposed method: $\forall\tau_i,\ (b_i \geq r_i) \cap (e_i \geq b_i + c_i)$

Scheduler information: In practice, two schedulers are used: fixed priority and EDF. In this paper, a lower priority value means a greater priority.

Determining priorities: In order to represent a fixed priority scheduler, the priority of a task instance just has to be equal to the priority of the corresponding task. If the priority of one or more tasks is not known, it is important to add the following information to the system: $\forall_{i,j,k}\, P_{i,j} = P_{i,k}$; because without this equation, two task instances of the same task can have different priorities. If the scheduler of the system is EDF, then the priority of each task instance is equal to the absolute deadline: $\forall_i\, p_i = d_i$.

Determinism: A scheduler is said to be deterministic if for every situation for every system with the same parameters, the same task instance is executed. The previously defined properties and hypotheses of the system and the method do not induce that the scheduler is deterministic; because two instances can have the same priority. However, the following property must be guaranteed:

Property 1. If several task instances that need the processor at the same time have the same priority which is the smallest of the system, then the scheduler must choose one and only one instance to execute.

Property 2. Once this choice done, the scheduler must not change it until the next system's change.

These two properties aim to prevent two task instances from being executed at the same time. To represent a possible non-deterministic scheduler, we introduce another parameter, the hidden parameter Π_i. This parameter acts as a hidden secondary priority, known by the scheduler only. Consequently, a task instance (i) has a higher priority than another (j) if and only if: $prio(i,j) = (p_i < p_j) \cup ((p_i = p_j) \cap (\Pi_i < \Pi_j))$. In order to verify the previous properties, the following constraint is added to the system: $\forall_{i,j}\, such\, as\, i \neq j,\ (p_i = p_j) \cap (\Pi_i \neq \Pi_j)$. The non-determinism of the scheduler comes from the fact that the hidden parameter Π_i is not given a fixed value.

Task instances behaviors: The notion of time is introduced with parameters which represent a moment when a particular action happens and thus a behavior. The complete behavior of a task instance is described with the following equation: $instance\, behavior_i = (\bigcup (possible\, b_i)) \cap (\bigcup (possible\, e_i))$.

The execution beginning of a task instance (b_j) can be *delayed* by another task instance (τ_i) with a higher priority: $delay(i,j) = prio(i,j) \cap (b_i \leq r_j) \cap (e_i > r_j)$. It is important to notice that only the task instance with the latest end of execution really delays the task instance. If the task instance i delays the task instance j, then the task instance j begins its execution when the task instance i ends: $b_j = e_i$.

The execution end of a task instance depends on the moment the task instance begins its execution and on the interactions with other task instances during its execution. In this case the interactions are *preemptions*. The end of an instance execution: $e_j = b_j + c_j + \sum_{i\,preempt\,j} c_i$. The task instance i preempts the task instance j if: $preempt(i,j) = prio(i,j) \cap (b_i > b_j) \cap (b_i < e_j)$.

Verifying properties: The result of the behavioral step represent all the possible behaviors of the system. The question consists in asking what is possible in the system behaviors by adding the correspond constraint. Thus, the PV-domain $\mathcal{D}$ representing the system is schedulable if: $\forall_{\tau_i} \mathcal{D} \cap (e_i > d_i) = \emptyset$

Specific characteristics

Atomic task instances: A task instance that cannot be preempted is said to be atomic. Thus, we introduced the static parameter $atomic(i)$ (and $not\,atomic(i)$) to indicate whether a task instance can be preempted. This induces a modification of the previously defined interactions by adding $\cup$ (resp. $\cap$) $atomic(i)$ to the equation $delay(i,j)$ (resp. $preempt(i,j)$).

Resource sharing: Sharing resources [18, 6] is a complex problem in real-time systems and several algorithms exist to solve it, thus we won't study it in detail. But we want to show how easy it is for this method to handle this problem by considering mutexes. The static parameter $Res(i,r)$ indicates that the task instance i needs the shared resource r. The function $Resource(i,j)$ represents that the task instance i and the task instance j need a common shared resource. This function is defined as follows: $Resouce(i,j) = \exists r,\ Res(i,r)\,and\,Res(j,r)$. Thus, the equations of the interactions are modified:

$$delay(i,j) = (prio(i,j) \cup atomic(i) \cup Resource(i,j)) \cap (b_i \le r_j) \cap (e_i > r_j)$$
$$preempt(i,j) = prio(i,j) \cap \neg atomic(j) \cap \neg Resource(i,j) \cap (b_i > b_j) \cap (b_i < e_j)$$

4 Example

In this section we present a simple example in which the system is made of three aperiodic tasks. The priorities of the tasks are known: $P_0 = 1$, $P_1 = 2$ an $P_2 = 3$. The deadlines are also known: $D_0 = 6$, $D_1 = 6$ et $D_2 = 20$. The release time are partially known: $R_0 = 0$, $R_1 = 3$ and R_2 is unknown. The execution times of τ_0 and τ_2 are partially known: $2 \le C_0 \le 5$ and $C_2 = 2$. The execution time of τ_1 depends on the execution of the other tasks. Indeed, $C_1 = 3$ if τ_1 begins its execution before the end of τ_0; $C_1 = 4$ if τ_1 begins its execution before the end of τ_2 and after τ_0; and $C_1 = 5$ if τ_1 begins its execution after τ_0 and τ_2.

To validate this system with a scheduling technique we proceed as follows: first, we determines the worst case values of the system parameters. Thus $C_0 = 5$,

$C_1 = 5$ and $C_2 = 2$. This approximation induces the pessimism of this method. That is why we show that the system is not schedulable. Indeed, a simple execution trace highlights that the second task misses its deadline (figure 2).

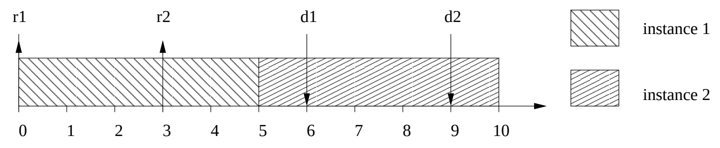

Fig. 2. Example

The main advantage of our method is it can represents the specific constraint of the execution of the second task. Thus, the following equation represent this information: $((C_1 = 3) \cap (b_1 < e_0)) \cup ((C_1 = 4) \cap (b_1 \geq e_0) \cap (b_1 < e_2)) \cup ((C_1 = 5) \cap (b_1 \geq e_0) \cap (b_1 \geq e_2))$. Then, the behavioral step is done. In the result representing the possible behaviors of the system we can highlight the invariants of the system behavior: $b_0 = 0$, $C_1 = 4$, $e_0 = C_0$. It shows that the first task is neither delayed nor preempted and also that the execution time of the second task is $C_1 = 4$. That means the second task begins its execution after the first one and before the third one, in all the possible situations. Thus, the worst value for this parameter ($C_1 = 5$) determined with the scheduling technique is impossible. The proposed method is not pessimistic; that is why we can prove that the system is schedulable with this method but not with a schedulability analysis.

With this simple example we show that it is possible to validate a real time system in which some parameters are exactly known, some are partially known, some are unknown and some parameters are defined with specific constraints. We also highlight that the proposed method is not pessimistic, contrary to a simple scheduling analysis.

5 Conclusions

We propose a new non-pessimistic method to analyse real-time systems. This method can be used to represent systems with specific behaviors and complex relationships between their parameters, because any information that can be represented with a linear inequality can be taken into account. A major advantage of this method is that can be used to analyse study a system at any design step. Indeed, this method can be used with partial knowledge and thus it is well adapted to reduce costs. To complete this work, we intend to focus on two points: taking into account new system characteristics, and implementing a complete software. Indeed, several system behaviors have to be represented in our method (dependable and/or non-preemptive tasks, resources sharing, partitioning [8, 20, 17, 4]), which implies some modifications of the previously defined equations. We also plan to include specific components such as bounded buffers.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.
2. Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, pages 209–229, 1992.
3. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
4. B. Andersson and J. Jonsson. Fixed-priority preemptive multiprocessor scheduling: To partition or not to partition. In *Proceedings of the Int'l Conf. on Real-Time Computing and Applications*, pages 337–346, Cheju Island, Korea, December 2000. IEEE Computer Society Press.
5. R. Bagnara. A hierarchy of constraint systems for data-flow analysis of constraint logic-based languages. In *Science of Computer Programming*, pages 30(1–2):119–155, 1998.
6. T. P. Baker. A stack-based resource allocation policy for realtime. In *Real-Time Systems Symposium*, pages 191–200. IEEE Computer Society Press, 1990.
7. L. P. Briand and D. M. Roy. *Meeting Deadlines in Hard Real-Time Systems: The Rate Monotonic Approach*. IEEE Computer Society, 1999.
8. Airlines Electronic Engineering Committee. Arinc specification 653, January 1997.
9. N. Halbwachs, Y-E. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In *Static Analysis Symposium*, pages 223–237, 1994.
10. T. A. Henzinger, P-H. Ho, and H. Wong-Toi. Hytech: The next generation. In *IEEE Real-Time Systems Symposium*, pages 56–65, 1995.
11. T. A. Henzinger, P-H. Ho, and H. Wong-Toi. A user guide to hytech. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 41–71, 1995.
12. T. A. Henzinger, P-H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.
13. Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? In *Proceedings of the 27th Annual Symposium on Theory of Computing*, pages 373–382, 1995.
14. Doran K.Wilde. A library for doing polyhedral operations. Technical Report 2157, December 1993.
15. Sanjay Rajopadhye Patrice Quinton and Tanguy Risset. On manipulating z-polyhedra. Technical Report 1016, Jully 1996.
16. P. M. Hill R. Bagnara and E. Zaffanella. Widening operators for powerset domains. In *Quaderno 349, Dipartimento di Matematica, Universit di Parma, Italy*, 2004.
17. J. Rushby. Partitioning in avionics architectures: Requirements, mechanisms, and assurance. Technical report, Menlo Park USA, March 1999.
18. L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39:1175–1185, September 1990.
19. H. Le Verge. A note on cherniakova's algorithm. Technical Report RR-1662, April 1992.
20. B. L. Di Vito. A formal model of partitionning for integrated modular avionics. Technical report, August 1998.