Discrete Geometry Applied in Hard Real-Time Systems Validation

Gaëlle Largeteau-Skapin, Dominique Geniet, Éric Andres

Discrete geometry applied in hard real-time systems validation.

Gaëlle Largeteau[1], Dominique Geniet[1], and Éric Andrès[2]

[1] Laboratoire d'Informatique Scientifique et Industrielle,
Université de Poitiers & ENSMA,
Téléport 2 - 1 avenue Clément Ader
BP 40109 86961 Futuroscope Chasseneuil cédex, France
largeteau@ensma.fr, dgeniet@ensma.fr,
[2] SIC

Abstract. Off-line validation of hard real-time systems usually stands on state based models. Such approaches always deal with both space and time explosions. This paper proposes a geometrical approach to model applications and to compute operational feasability from topological properties. Thanks to this model, we can decide the feasability of real-time synchronous systems composed of periodic tasks, sharing resources, running on multiprocessor architectures. This method avoids state enumeration and therefore limits both space and time explosion.

Key words: Real-time, operational validation, multiprocessors, resource sharing, geometrical modeling.

1 Introduction

In a real-time system, the correctness of a computation depends on both the logical results of the computation and the dates when results are produced. Time constraints are called strict if not respecting them involves irreparable consequences on the system safety. In this case, the system is called *hard* [But97]. On the opposite, if not respecting deadlines keeps the system safe, the system is called *soft*. In this study, we only consider hard real-time systems: time constraints are strict.

A real-time system is a task set: each task is a process designed to react to an external incoming event. The systems we study use resources and run on centralized multiprocessor architectures. All processors are identical; tasks are preemptive and can move from a processor to another one at any time. Each task τ_i is specified by time characteritics: its first activation date r_i, its deadline D_i, its period T_i, and its execution time C_i [LL73]. We assume that tasks are periodic and no reentrant: $\forall i \in [1, n], D_i \leq T_i$.

The operational validation of a real-time system is reached by proving that no task misses deadline, i.e by proving that there exists at least one time valid

scheduling sequence for the system. This proof is obtained by feasability conditions or simulations. Validation is performed off-line for systems sharing resources which run on multiprocessor architectures, since there exists no necessary and sufficient feasability condition in this case [Mok83]. Off-line methods are usually based on state models (Petri nets, automata) both in timed or time-free versions. Timed models consider that time is explicit [ALU94], associating time intervals with transitions or durations with states. Time free models use implicit time [CHO96] (implicit timed models): each transition is associated with the same duration and constraints are expressed as numbers of transitions. The main advantage of implicit timed models is that very efficient tools are provided to analyse them. Therefore time is discrete in these models. In [LG02], we have defined an implicit timed model, based on finite automata, that enumerates states of systems and therefore involves both space and time explosion.

Observing the graphs of the automata we obtained in this approach, we have conceived a new model, based on discrete geometry, that is presented in this work. Our goal is to reduce notably both time and space explosion in the validation process. In this model, we associate each task with a geometrical figure which only depends on time characteristics r, C, D, T. Geometrical operations (extrusion, cartesian product, intersection) allow to model parallelism and synchronization in a geometrical way. This model makes state enumeration implicit and therefore decreases both space and time explosion while keeping a strong expression capacity.

We define the geometrical model for a single task in section 2. In section 3, we present compositional operations to integrate both parallelism and synchronization in the model. Section 4 is dedicated to the presentation of a software implementing this modeling process.

2 Model definitions

2.1 A two dimensional figure to model single tasks

A task is usually models by an automaton (see figure 1). Each transition of this automaton is associated to a duration of one time unit (the time is discrete). A task state is defined by the execution progress $x{=}C(t)$ of the task and the total time t since the system activation. Therefore, we consider, for each task, a two-dimensional space (t, c): t adresses absolute time and c adresses x.

A task execution corresponds to the successive executions of its instances. At time t, k instances of task τ (r,C,D,T) are completed (corresponding to $k \times C$ time units of CPU owning) and the $(k+1)^{\text{th}}$ instance is running (C_{k+1}(t) time units of past CPU owning). So, at time t, $C(t){=}k \times C + C_{k+1}(t)$ time units have been executed. During its execution, an instance of task τ goes through several states wether it owns the CPU or not: the initial state e_0 of τ's first instance is $(r, 0)$, its last state is $(r+T, C)$. The final state of any instance is the first state of the next instance. The final state e_k of the k^{th} instance of τ is $(e_{k-1} + (T, C))$. We denote $r^k = r + k \times T$ the activation date of the k^{th} task's instance of τ.

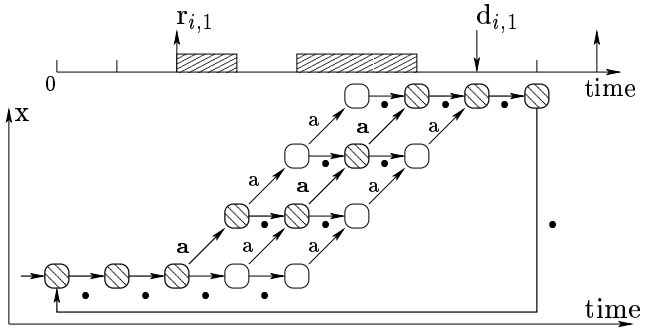

Fig. 1. Automaton model for task τ (r=2,C=3,D=5,T=7).

A task execution is then totaly defined by the set of all instance states. This set is the graph of a function in space (t, c). This function is called *"trace"* (see figure 2) . Note that for τ, this function is not unique.

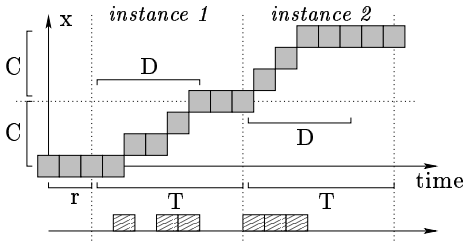

Fig. 2. A trace of task τ (r=2,C=3,D=5,T=7).

Let us now characterize traces. Since tasks can not be parallelized, a task can not be in more than one state at once and its state can not be undefined. Therefore, a task trace is a mapping between time and task execution progress. This mapping is an increasing function: either the task is progressing during execution; or the task is suspended and its execution progress state keeps the same value. Moreover, during any time interval [t,t+1], no task can progress more than 1 since it is the maximal CPU time that can be allocated to the task for its execution during this interval. Therefore, a trace is a mapping.

Definition 1. *We call* ***trace*** *of task τ an increasing mapping $Tr(\tau)$:*
$Tr_\tau : \mathbb{Z}^+ \to \mathbb{Z}^+$, $t \to Tr_\tau(t)$ *such that* $\forall t \geq 0,\ Tr_\tau(t+1) \in \{Tr_\tau(t), Tr_\tau(t)+1\}$.

Task τ must deal with its temporal characterisation (r,C,D,T): this property imposes geometrical constraints on execution traces of τ (see Figure 3). Some task traces are comptatible with task operational charateristics: they are *"valid task traces"*. Others are not compatible: they are unvalid.

Definition 2. *A **valid trace** of τ (r,C,D,T) is an execution trace $Tr_\tau V$ such that:*

$Tr_\tau V([0,r]) = \{0\}$
$Tr_\tau V([r^k, r^k+D]) = [(i\text{-}1)\times C, k\times C]$
$Tr_\tau V([r^k+D, r^{k+1}]) = \{k\times C\}$

We call $TV(\tau)$ the set of τ's valid traces. If τ misses no deadline, no instance of τ run on time intervals [0,r] and $[r^k+D, r^{k+1}]$. Therefore, traces $Tr_\tau V$ are constant functions over these intervals. The equation $Tr_\tau V(r^k+D)=k\times C$ means that the execution requirement C_i of the k^{th} instance of τ_i is completed before its deadline. Thus, such a trace is a valid trace. Then, we can characterize the set $\Omega(\tau)$ which collects all points of valid traces.

Definition 3. *The validity space $\Omega(\tau)$ of a task τ is:*

$$\Omega(\tau) = \bigcup_{\psi \in TV(\tau)} \{(t, \psi(t))\}.$$

We note $T(\Omega(\tau))$ for $TV(\tau)$.

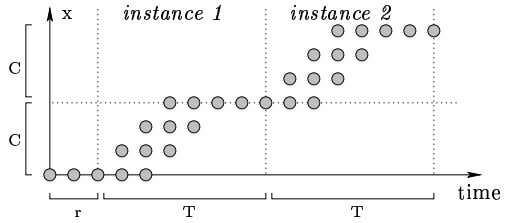

Fig. 3. Discrete model for two instances of task τ (r=2,C=3,D=5,T=7).

Figure 3 presents the validity space for the two first instances of a task. Each valid state of τ is associated with a point of $\mathbb{Z}^2$.

2.2 Concurrency modeling

Let $\Gamma=(\tau_i)_{i\in[1,n]}$ be a set of tasks, designed to run concurrently. The state of Γ at time t is defined by the states of all τ_i. At time t, Γ is valid if and only if all tasks of Γ are valid.

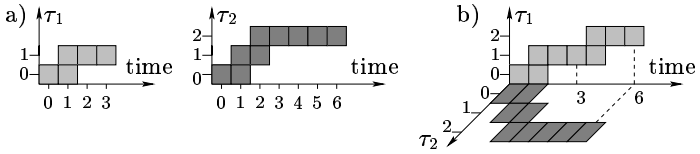

Fig. 4. Examples: τ_1 (r=0, C=1,D=2,T=3), τ_2 (r=0,C=2,D=3,T=6)

Definition 4. *A valid state P of a system Γ at time t is defined by:*

$$P = (t, x_1, ..., x_n) / \ \forall i \in [1, n],\ (t, x_i) \in \Omega(\tau_i)$$

Therefore, the space of Γ's valid states is (n+1)-dimensional (see Figure 4).

$$\Omega(\Gamma) = \{(t, x_1, ..., x_n) / \ \forall i \in [1, n]\ (t, x_i) \in \Omega(\tau_i)\}$$

2.3 Computing $\Omega(\Gamma)$

We consider a system Γ of n tasks. However, for drawings, we get n=2 (see Figure 4). In this case $\Omega(\Gamma)$ is a three-dimensional object, easy to view on a page.

Geometrical basic notions:

Two discrete points p and q in n dimension are **k-neighbour**, for $0 \leq k \leq n$, if $\forall 1 \leq i \leq n,\ |p_i - q_i| \leq 1$ and if $k \leq n - \sum_{i=1}^{n} |p_i - q_i|$. A **$k$-path** in a discrete obect A is a A discrete point list such that two consecutive points of this list are k-neighbour (A task trace is a 0-path in 2-dimension). k-connexite et k-composantes !!!!A FAIRE!!!

Definition 5. *Let $\mathcal{I}$=$\{i_1, ..., i_{|\mathcal{I}|}\}$ ($i_1 < i_2 < ... < i_{|\mathcal{I}|}$). We define the injection operation $\mathcal{J}_{\mathcal{I},n}$ in the following way: $\mathcal{J}_{\mathcal{I},n} :\ \mathbb{Z}^{|\mathcal{I}|} \rightarrow \mathbb{Z}^{n+1}$*

$$(x_1, ..., x_{|\mathcal{I}|}) \rightarrow \overbrace{(0, ..., 0, \underset{i_1}{x_1}, 0, ..., 0, \underset{i_j}{x_j}, 0, ..., 0, \underset{i_{|\mathcal{I}|}}{x_{|\mathcal{I}|}}, 0, ..., 0)}^{n+1}$$

Notations:
- $\mathcal{J}_{i,n}$=$\mathcal{J}_{\{1,i+1\},n}$: (a,b) $\rightarrow$ (a,0,..,0,b,0...,0)
  1 i+1
- $\Lambda_{\mathcal{I},n}$ is the following cartesian product:

$$\Lambda_{\mathcal{I},n} = \mathbb{Z}^{i_1 - 1} \times \{0\} \times \mathbb{Z}^{i_1 - i_2 - 1} \times \{0\} \times ... \times \mathbb{Z}^{i_{|\mathcal{I}|} - i_{|\mathcal{I}|-1} - 1} \times \{0\} \times \mathbb{Z}^{n+1-i_{|\mathcal{I}|}}.$$

- $\Lambda_{i,n}$ is the cartesian product $\Lambda_{i,n} = \{0\} \times \mathbb{Z}^{i-1} \times \{0\} \times \mathbb{Z}^{n-i}$=$\Lambda_{\{1,i+1\},n}$.

Definition 6. *We define the interleaved cartesian product $\mathcal{J}^{\Lambda}_{\mathcal{I},n} :\ \mathbb{Z}^{|\mathcal{I}|} \rightarrow \mathcal{P}(\mathbb{Z}^{n+1})$*
$(x_1, ..., x_{|\mathcal{I}|}) \rightarrow \{\mathcal{J}_{\mathcal{I},n}((x_1, ..., x_{|\mathcal{I}|})) + \lambda,\ \lambda \in \Lambda_{\mathcal{I},n}\}$

Notations:
- $\mathcal{J}^{\Lambda}_{i,n}$=$\mathcal{J}^{\Lambda}_{\{1,i+1\},n}$=$\{\mathcal{J}_{i,n}((a,b)) + \lambda,\ \lambda \in \Lambda_{i,n}\}$.

Definition 7. *We call "concurrent product" (denoted $\otimes$) of $\Omega(\tau_1)$ and $\Omega(\tau_2)$ the set $\Omega(\tau_1) \otimes \Omega(\tau_2)$= $\mathcal{J}^{\Lambda}_{1,n}(\Omega(\tau_1)) \cap \mathcal{J}^{\Lambda}_{2,n}(\Omega(\tau_2))$.*

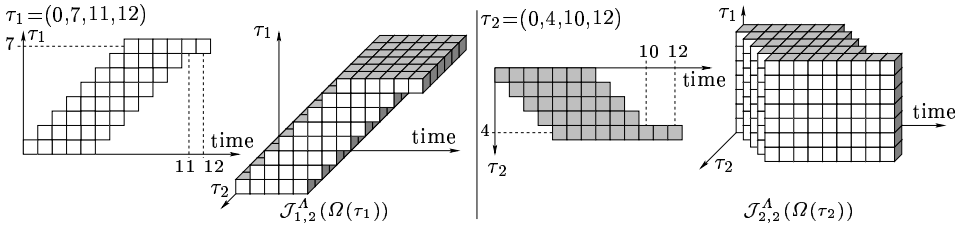

Fig. 5. Example: $\mathcal{J}^{\Lambda}_{1,2}(\Omega(\tau_1))$ and $\mathcal{J}^{\Lambda}_{2,2}(\Omega(\tau_2))$.

Remarks:
-The operation $\otimes$ is associative, therefore we can generalize the notation:

$$\Omega(\tau_1, ..., \tau_n) \;=\; \bigotimes_{i=1}^{i=n} \Omega(\tau_i)$$

- $\forall\ \mathcal{J} \in \mathcal{J}^{\Lambda}_{i,n}((a,b))$, $\forall k \in [1, i-1] \cup [i+1, n], \exists\ x_k \in \mathbb{Z}$ such that:

$$\mathcal{J} \;= (\mathbf{a}, x_1, ..., x_{i-1}, \mathbf{b}, x_{i+1}, ..., x_n)$$

- Projection Π_i: (y_1,...,y_{i+1},...,y_{n+1}) $\rightarrow$ (y_1,y_{i+1}) is a reverse operation of $\mathcal{J}^{\Lambda}_{i,n}$: $\Pi_i(\mathcal{J}^{\Lambda}_{i,n}(a,b))=\{(a,b)\}$.

Theorem 1. $\Omega(\Gamma) \;=\; \bigotimes_{i=1}^{i=n} \Omega(\tau_i)$.

Proof: Let us show that $\bigotimes_{i=1}^{i=n} \Omega(\tau_i) \subset \Omega(\Gamma)$:

Let $P=(t, x_1, x_2, ...x_n) \in \bigotimes_{i=1}^{i=n} \Omega(\tau_i)$. The definition of $\otimes$ gives:
$P=(t, x_1, x_2, ...x_n) \in \bigcap_{i\in[1,n]} \left(\mathcal{J}^{\Lambda}_{i,n}\big(\Omega(\tau_i)\big)\right)$. Since P belongs to an intersection set, we get: $\forall i \in [1,n],\ P \in \mathcal{J}^{\Lambda}_{i,n}\big(\Omega(\tau_i)\big)$. Using the two-dimensional projection on τ_i-space (t,τ_i): $\forall i \in [1,n],\ \Pi_i(P) \in \Pi_i\big(\mathcal{J}^{\Lambda}_{i,n}\big(\Omega(\tau_i)\big)\big) = \Omega(\tau_i)$. And then $\forall i \in [1,n],\ (t, x_i) \in \Omega(\tau_i)$. Therefore, $P \in \Omega(\Gamma)$.

Let us show that $\Omega(\Gamma) \subset \bigotimes_{i=1}^{i=n} \Omega(\tau_i)$:

Let $P=(t, x_1, x_2, ...x_n) \in \Omega(\Gamma)$. We consider the projection of P on each plane (t,τ_i): $\forall i \in [1,n]\ \Pi_i(P) \in \Omega(\tau_i)$. We then consider $\mathcal{J}^{\Lambda}_{i,n}$ for each Π_i (P): $\forall i \in [1,n]\ \mathcal{J}^{\Lambda}_{i,n}\big(\Pi_i(P)\big) \subset \mathcal{J}^{\Lambda}_{i,n}\big(\Omega(\tau_i)\big)$. Finally, we consider the intersection of all sets we have obtained:
$\bigcap_{i\in[1,n]} \left(\mathcal{J}^{\Lambda}_{i,n}\big(\Pi_i(P)\big)\right) \subset \bigcap_{i\in[1,n]} \left(\mathcal{J}^{\Lambda}_{i,n}\big(\Omega(\tau_i)\big)\right)$ Since $\bigcap_{i\in[1,n]} \left(\mathcal{J}^{\Lambda}_{i,n}\big(\Pi_i(P)\big)\right) =$
$\{(t, x_1, x_2, ...x_n)\} = \{P\}$, we get: $P \in \bigotimes_{i=1}^{i=n} \Omega(\tau_i)$. □

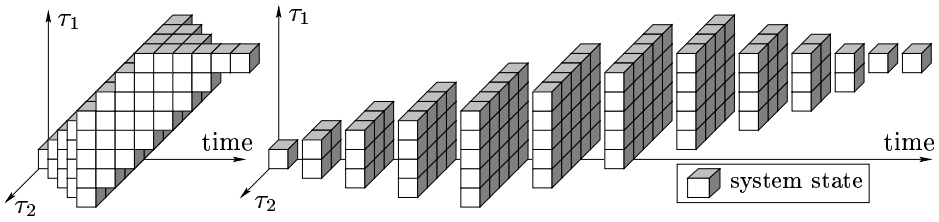

Fig. 6. Example: geometrical modeling for a two tasks system, $\Omega(\tau_1) \otimes \Omega(\tau_2)$.

2.4 Synchronization integration

Resource sharing

In real-times systems, tasks use critical resources (there is mutual exclusion): only one task can use resource R at time t. Therefore, some states of $\Omega(\Gamma)$ become invalid from the resource sharing point of view. On a geometric plan, the k^{th} instance of a task corresponds to the task execution progress interval [$C_i \times k$, $C_i \times k + C_i$]. We define $u^k(1)$ and $u^k(2)$ such that this instance uses R during the execution time interval: $]u_i^k(1), u_i^k(2)[\subset [C_i \times k, C_i \times k + C_i]$.

We denote by u_i the time interval union $\bigcup_{k \in \mathbb{N}}]u_i^k(1), u_i^k(2)[$, corresponding to all resource requirement intervals for all instances of τ_i (see Figure 7).

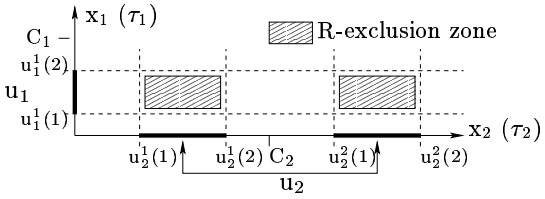

Fig. 7. Example: both τ_1 and τ_2 use resource R

Since a task cannot own the resource if another task already uses it, two tasks cannot be in R critical section at the same time. Then if P=(t,$x_1, ..., x_n$) and $x_i \in u_i$, we must get $\forall j \neq i, x_j \notin u_j$. Therefore, for a valid state P=$(t, x_1, ..., x_n)$ of Γ, the following property stands: $|\{x_i / x_i \in u_i\}| \leq 1$. The validity space of Γ respecting resource R sharing is (see figure 8):

$$\Omega_R(\Gamma) = \{(t, x_1, ..., x_n) \in \Omega(\Gamma) / |\{x_i / x_i \in u_i\}| \leq 1\}$$

Here, we deal only with one resource R. For many resources, since resources are independant, the technique can be applied by induction. Let $\Gamma_R = (\tau_j)_{j \in \mathcal{I}_R}$ be the set of tasks sharing resource R (we note $\mathcal{I}_R$ the set of indices of tasks sharing R). The states of Γ_R which are associated with a simultaneous use of R are unvalid. The R-exclusion zone $\Omega(R)$ of R collects all states of Γ_R corresponding R incorrect uses. $\Omega(R)$ is part of the subspace associated with all tasks sharing R, since it implies the simultaneous run of at least two of these tasks.

Definition 8. *The R-exclusion zone is (see Figure 7):*

$$\Omega(R) = \{(x_i)_{i\in\mathcal{I}_R} / \ |\{x_i / x_i \in u_i\}| > 1\}.$$

A state of $\Omega(R)$ only concerns tasks of Γ_R. An state s=(t, $x_1, ..., x_n$) is unvalid if $\Pi_{\Gamma_R}(s) \in \Omega(R)$. Therefore, the set $\eta_R = \{(t, x_1, ..., x_n)/t \in \mathbb{Z},\ |\{x_i/x_i \in u_i\}| > 1\}$ of unvalid states can be obtained thanks to a concurrent cartesian product and an extrusion operation.

Theorem 2. $\eta_R = Extr\left(\mathbb{Z}, \mathcal{J}^{A}_{\mathcal{I}_\mathcal{R},n}(\Omega(R))\right)$.

Proof: This theorem comes directly from the definitions of Ω(R), the extrusion operation and interleaved cartesian product operation. Ω(R) collects all unvalid states of Γ_R: it is a $|\mathcal{I}_R|$-dimensional object. The interleaved cartesian product associates these states with all possible states of $\Gamma \setminus \Gamma_R$. Then all states of Γ that are unvalid in the R-sharing point of view are reached. $\mathcal{J}^{A}_{\mathcal{I}_\mathcal{R},n}(\Omega(R))$ is an n-dimensional object. Now one must integrate that these states are always unvalid. This is done by extruding this objet following the time direction ($\mathbb{Z}$). This operation collects all R-sharing unvalid states of Γ. Then we get $\eta_R = Extr\left(\mathbb{Z}, \mathcal{J}^{A}_{\mathcal{I}_\mathcal{R},n}(\Omega(R))\right)$. η_R is a (n+1)-dimensional object. □

Theorem 3. $\Omega_R(\Gamma) = \Omega(\Gamma) \setminus \eta_R$.

Therefore, the set of valid traces including resource sharing is:

$$T(\Omega_R(\Gamma)) = \{\psi \in T(\Omega(\Gamma))/\ \forall t \in \mathbb{Z}^+,\ \psi(t) = (x_1, ..., x_n),\ (t, x_1, ..., x_n) \in \Omega_R(\Gamma)\}.$$

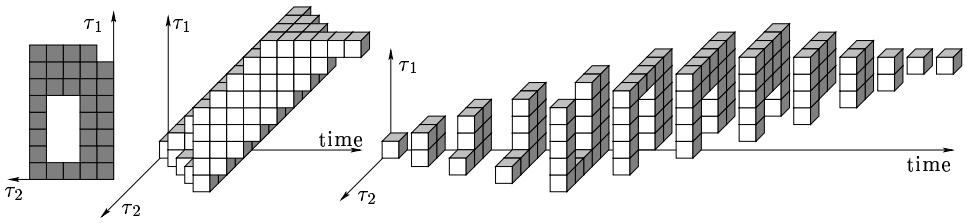

Fig. 8. Geometrical model of a system including resource sharing.

Processor sharing

While building $\Omega_R(\Gamma)$, we have not considered the number of processors. However, this parameter makes each trace ω in $T_R(\Gamma)$ valid or unvalid according to the minimal number of processors useful to execute ω.

During an execution, the number of active processors is constant between two consecutive context switches. Then, to decide the validity of a trace, we only have to look at it at context switch times. Let us note by q the scheduling quantum. If there are k running tasks between two given context switches a and b, the trace is called k-concurrent between a and b.

Definition 9. *A trace $\psi \in T(\Omega_R(\Gamma))$ is k-concurrent between two context switch times $i \times q$ and $(i+1) \times q$ if and only if:*
$\psi(i \times q) = (i \times q, (x_i)_{i \in [1,n]})$ *and* $\psi((i+1) \times q) = ((i+1) \times q, (y_i)_{i \in [1,n]}) \Rightarrow$

$$\sum_{i \in [1,n]} (y_i) - \sum_{i \in [1,n]} (x_i) = k \times q.$$

We said that discrete points $\psi(i \times q)$ and $\psi((i+1) \times q)$ are k-concurrent. This definition of k-concurrency in a $n+1$ dimensional space corresponds to the definition of the $n-k$-neighbourhood. A k-concurrent trace is then a $n-k$-path in $\Omega_R(\Gamma)$. We denote by $\mathrm{T}_{R,k}(\Gamma)$ the set of k-concurrent traces of $\mathrm{T}(\Omega_R(\Gamma))$.

Definition 10. *A set $\Omega_R(\Gamma)$ is k-concurrent if there exists at least one k-concurrent trace ψ in $T_R(\Gamma)$.*

Remark: If a set $\Omega_R(\Gamma)$ is k-concurrent, then it is a $n-k$-componant and there exists at least one valid sheduling sequence for Γ on a k processor architecture.

2.5 Feasability decision

For a system running on a k processor architecture and sharing a resource R, a valid scheduling is a k-concurrent trace in $\mathrm{T}\Omega_R(\Gamma)$. The feasability decision is reached by evaluating of the predicate: $T_{R,k}(\Gamma) \neq \emptyset$.

3 Implementation

We have developped the software GemSMARTS (Geometric Scheduling Modeling and Analysis of Real-Time Systems) which computes the set $\Omega_R(\Gamma)$. We have tested a discret data structure implemented through classical matrices. In this first version of our tool, we only consider synchronous task systems $(\forall i \in [1,n], r_i = 0)$. System states are elements of $\mathbb{N}^{n+1}$ and since an execution cycle duration is known and is equal to $\mathrm{PPCM}((\mathrm{T}_i)_{i \in [1,n]})$, point sets are finite. On drawings, each cube models a system state at a context switch time.

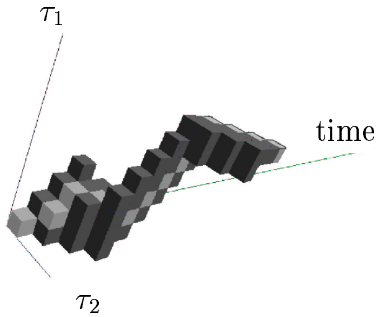

Fig. 9. System geometrical model : τ_1=(0,7,10,12), τ_2=(0,3,6,6)

Figure 9 shows $\Omega_R(\Gamma)$ for a two-tasks system sharing a resource. The optimal number p of processors is obtained in the following way: for each possible value of k, we evaluate $T_R(\Gamma) \neq \emptyset$, considering only k-concurrent traces. p is the minimum of the obtained values. For the example, $p = 2$: this system is not feasible on a single processor but it is with two. A two-processors scheduling sequence is represented by light grey cubes on Figure 9.

4 Conclusion

Validity spaces are useful to model hard real-time systems running on multiprocessor architectures and sharing resources. Feasability of task systems and optimal numbers of processors can be computed thanks to the k-concurrency concept.

Feasability is usually decided using state based model and model checkers [LG02]. Using validity spaces involves a noteworthy improvement: the time saved on a time free automata model is about 85%. The data structure we have tried (matrices) is useful to implement our model and developed algorithms are already more efficient than the previous ones (using automata) although they are far from being optimized. We have reached a limited time complexity depending mainly on the number of tasks whereas classical models follow complexities which depend on time. This method allows to drive back both space and time explosion.We are studying space and time complexities for optimized version of the geometrical algorithms.

Ongoing works concern definitions of both topological and geometrical properties to precisely characterize scheduling sequences. We plan, for example, to define topological properties for validate on-line classical scheduling (RM, ED, and so on), in order to propose multiprocessor versions of on-line validation techniques integrating resource sharing.

References

[ALU94] R.Alur, D.Dill: A theory of timed automata. Theoretical Computer Science, vol. 126, pp 183-235.(1994).

[And03] E.Andres: Discrete linear objects in dimension n: the standard model. Graphical Models 65(1-3), pp 92-111. (2003).0

[But97] G. Buttazzo: Hard real-time computing systems: Predictable scheduling algorithms and applications. Kluwer Academic Publishers. (1997).

[CHO96] A.Choquet-geniet, D.Geniet, F.Cottet: Exhaustive computation of scheduled task execution sequences of real-time application. Proc FTRTFT'96.(1996)

[LL73] C.L. Liu et J.W. Layland : Scheduling algorithms for multiprogramming in real-time environnement. Journal of the ACM, vol 20, pp 46-61.(1973)

[LG02] G.Largeteau, D.Geniet: Term Validation of Distributed Hard Real-time Applications. Conference on Implementation and Application of Automata. (2002).

[Mok83] A.K.Mok: Fundamental design problems for the hard real-time environments. Ph.D. MIT.(1983).