Unité Mixte de Recherche 5104 CNRS - INPG - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

Modeling and Analysis of Real Time Systems with Preemption, Uncertainty and Dependency

Marcelo Zanconi

January 12, 2005

Modeling and Analysis of Real Time Systems with Preemption, Uncertainty and Dependency

Marcelo Zanconi

Verimag

January 12, 2005

Abstract

In this work we are interested in the specification and analysis of real time systems, in particular, we cope with the problem of schedulability of real time tasks, that is a set of tasks with time constraints, such as deadline, execution time, periodicity; we also treat the problem of data sharing under a mutual exclusion hypothesis.

Time is of most importance in real time systems since we deal with critical applications, whose failure may cause serious or fatal accidents and also with different technologies which must be integrated. We have chosen to use timed automata to represent tasks and its timing constraints and their composition to represent system evolution. In particular, based on the idea of a particular class of stopwatch automata we have developped a technique to model and analyse a real time system:

- We first show a proper and new technique to deal with the problem of preemptive scheduling and decidability under an asynchronous paradigm;
- We show an evolutive application of this method starting from a very simple policy and finishing to a general scheduling policy;
- In each step of this evolution we show that our method is decidable, that is, that its application can leave the system in a safe state and that this state can be reachable.
- We also show a complete admission analysis that can be performed off line in case of a set of periodic tasks and on-line in case of an hybrid set of periodic and aperiodic tasks; in any case, the admission is the simple computation of a formula.

Keywords: Scheduling, Real Time Systems

Tutor: Sergio Yovine

Notes: This report presents the main results of Zanconi's Ph.D thesis

Contents

The heart of the problem

1 Motivation

The behavior of real-time systems with preemptive schedulers can be modelled by *stopwatch automata*. Nevertheless, the expressive power of stopwatch automata discouraged for a long time their use for verification purposes. Indeed, the reachability problem (even for a single stopwatch) has been proven to be undecidable [6, 7, 2].

There are, however, some decidable sub-classes such as the so-called *integration graphs* [7] and *suspension automata* [10]. The latter are actually useful for modelling and analyzing systems made up of a set of tasks with fixed execution times. SWA can be translated into timed automata with updates, specifically decrementation by a constant, for which the reachability problem is indeed decidable [1].

The result of [10] has been extended in [4] to a more general model TAT, though still requiring constant execution times of tasks. The approach via timed automata with decrementation suffers of two main problems. First, it requires a costly translation. Second, it only allows modelling tasks with fixed execution times. A technique to cope with the first problem has been proposed in [3].

In this paper we focus on preemptive scheduling of systems of tasks with uncertain but lower and upper bounded execution times. The behavior of these systems cannot be straightforwardly translated into a decidable extension of timed automata with updates. Our approach consists in encoding the value of stopped clocks as the difference of two running ones. We do allow tasks to be restarted again; initially we forbid preempting a task more than once, then we extend to a more general model. We show that the system can be modelled by formulas involving *difference bounded matrices*, DBM, that is, difference constraints on clocks, and time-invariant equalities capturing the values of stopped clocks. This result implies decidability and leads to an efficient implementation. Moreover, it gives a precise symbolic characterization of the state space for the considered class of systems.

2 Model

A real time application is modelled as a collection $\mathbb{T} = \{\tau_1, \tau_2, \ldots, \tau_m\}$ of all tasks for the application, which are triggered by external events, including a timed event such as period. Each task τ_i is characterized by a vector of parameters $[G_i, D_i]$ $1 \leq i \leq m$ where $G_i = [E_i^{\min}, E_i^{\max}]$ is the execution time-interval, i.e. the best and worst case execution time and D_i is the relative deadline. For each task τ_i, we have two timed variables, namely r_i and e_i, that measure the *release* time and the *accumulated executed* time, respectively. Both variables are reset to zero whenever task τ_i arrives. Task arrival is denoted by $\tau_i\!\uparrow$ and task completion by $\tau_i\!\downarrow$.

The *environment* is any untimed relation between arrivals and completions of all tasks (it should respect the precedence relationship between $\tau_i\!\uparrow$ and $\tau_i\!\downarrow$, though).

Example .1 *In figure 1 we can see an example of a model of system of four tasks.*

The general model of an application is then a graph $\mathcal{G} = (V, A)$, where the set of vertices $V \subseteq \{\tau_i\!\uparrow, \tau_i\!\downarrow\}_{1\leq i\leq m}$ where $|\mathbf{in}(V)| \leq 1$ (no more than one incoming edge per vertex) and a set of directed edges

$$A \subseteq \{\tau_i\!\uparrow \to \tau_i\!\downarrow\}_{1\leq i\leq m} \cup \{\tau_i\!\downarrow \to \tau_j\!\uparrow\}_{i\neq j,\ 1\leq i,j\leq m} \cup \{\tau_i\!\uparrow \to \tau_j\!\uparrow\}_{i\neq j,\ 1\leq i,j\leq m}$$

Let $T \subseteq \mathbb{T}$ be the finite set of *active* tasks in the system, that is, those that have already arrived and are currently being handled by the scheduler. At any moment, at most one instance of a task may be active. The predicate $\mathbf{exec}(T_i)$ indicates whether T_i is executing or not and for the set T, $\mathbf{exec}(T)$ denotes the executing task of T. The predicate $\mathbf{accept}(\tau_i)$ indicates whether τ_i is accepted or not at its arrival due to some scheduling or modelling reasons; for instance, τ_i could be rejected because it would produce some

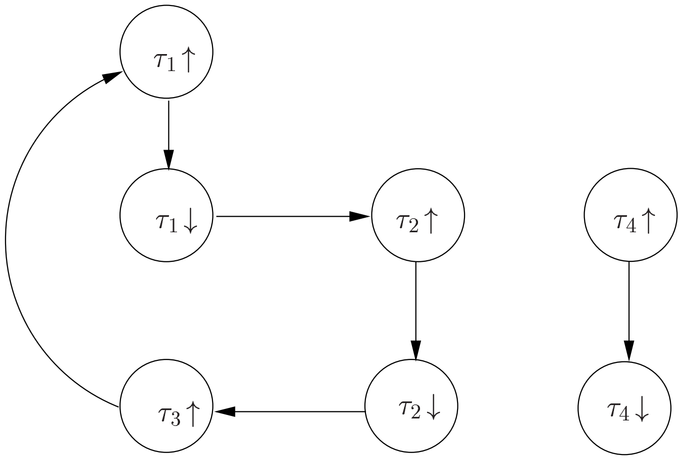

Figure 1: A model of a system

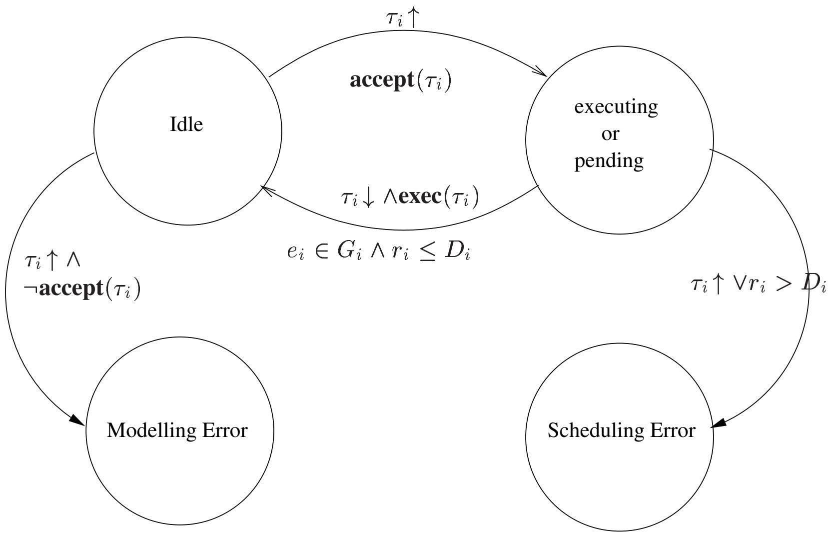

Figure 2: Task automaton

tasks to miss their deadlines or because there is an active instance of this task. A detailed description of this predicate will be given when we precise the scheduling policy. Figure 2 shows a task automaton.

The dynamic behavior of the system is represented by a transition system $(\mathcal{S}, T, \rightarrow)$ where $\mathcal{S}$ is a set of states, T is the set of active tasks, and $\rightarrow$ is the transition relation. $\mathcal{S}$ is a tuple of control locations of the task automaton (Fig. 2) and of valuations of timed variables. The following rules give a sketched behaviour of the system; formal and complete rules will be given later when analysing particular scheduling policies, SCH.

- **Task completion**: $\tau_i\downarrow$ If $e_i \in G_i$, $r_i \leq D_i$, and **exec**(τ_i), then

$$(\mathcal{S}, T) \stackrel{\tau_i\downarrow}{\rightarrow} (\mathcal{S}', T \ominus \{\tau_i\})$$

  where $\mathcal{S}'$ is obtained according to SCH and $\ominus$ is the operation of removing a task from T.

- **Task arrival**: $\tau_i\uparrow$

If $\tau_i \in T \Rightarrow$

$$(\mathcal{S}, T) \stackrel{\tau_i\Uparrow}{\rightarrow} \text{Schedulling Error}$$

(no more than one instance of each task)

If $\tau_i \notin T \wedge \neg$**accept**$(\mathcal{S}, T, \tau_i) \Rightarrow$

$$(\mathcal{S}, T) \stackrel{\tau_i\Uparrow}{\rightarrow} \text{Modelling Error}$$

If **accept**$(\mathcal{S}, T, \tau_i) \Rightarrow$

$$(\mathcal{S}, T) \stackrel{\tau_i\Uparrow}{\rightarrow} (\mathcal{S}', T \oplus \{\tau_i\})$$

where S' is obtained according to SCH and $\oplus$ is the operation of inserting a task in T.

- **Deadline violation**: If $r_i > D_i$ for some $\tau_i \in T \Rightarrow$

$$(\mathcal{S}, T) \rightarrow \text{Schedulling Error}$$

- **Time passing**: Let **exec**(τ_i), $\delta \geq 0$, and $e_i + \delta \leq E_i^{\max} \Rightarrow$

$$(\mathcal{S}, T) \stackrel{\delta}{\rightarrow} (\mathcal{S}', T)$$

where S' is obtained from S by adjusting the values of timed variables according to SCH.

The first rule expresses the completion of the executing task, leaving to the scheduler the choice of choosing the next task to be executed. The definition of **exec**(T) and the computation of the next state are left unspecified since they are dependant of the SCH.

The second rule expresses the arrival of a new task, which can be accepted or rejected. We distinguish two transitions leading to an error state, one for unschedulability and the other for a behaviour not satisfying the modelling assumptions.

The third rule expresses the case of a deadline violation. The fourth rule expresses time passing of δ units of time, adjusting the values of the timed variables in T.

In general we will assume the existence of an acceptance test at task arrival. This test is related to some assumptions of our system and of course, to the scheduling policy. Once the task passes the test, it can enter the system, either waiting for its turn or executing immediately, preempting the currently executing task. Predicate **accept**$(\mathcal{S}, T, \tau_i)$ will be analysed in detail for different scheduling policies.

3 LIFO scheduling

To show our analysis, we start with a very simple scheduling policy: a LIFO scheduler, that is, a scheduler where the current executing task is always preempted by the recently arrived task. We also suppose that each task can be preempted for at most once.

Intuitively speaking, a one preemption LIFO scheduler accepts tasks in the stack, until the task on the top finishes; at this moment, as all tasks beneath it had already been preempted, the scheduler will reject any new task, until the stack is empty; note, then, that all tasks in the stack had been preempted once, except the task on the top which could have never been preempted.

Example .2 *Let* $\mathbb{T} = \{\tau_1(4, 12), \tau_2(5, 10), \tau_3(2, 10), \tau_4(3, 6)\}$ *be a set of tasks, where the numbers in parentheses represent execution times and deadlines. In this example, deadlines are sufficiently long to let all tasks execute on time.*

Figure 3 shows the reaction of a LIFO scheduler at arrival of each task. For instance at time $t = 3$, $\tau_2 \uparrow$ and preempts τ_1, noted $\tau_2 \dagger \tau_1$; note that at time $t = 8$, $\tau_3 \downarrow$, and τ_2 resumes execution, noted as $\tau_2 \nearrow$; remark that the arrival of τ_4 at time $t = 9$ is ignored by the scheduler, since τ_2 had already been preempted. We also show the evolution of clocks.

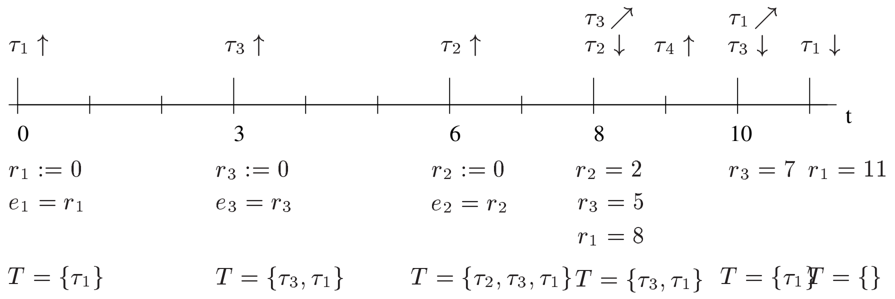

Figure 3: One preemption LIFO Scheduler

3.1 LIFO Transition Model

Let $T = \{T_1, T_2, \ldots T_n\}$ be the *stack* of active tasks in the system and let T_n be the task in execution, i.e. $T_n = \textbf{exec}(T)$; if τ_i arrives to the system and it is accepted, then τ_i preempts T_n, written as $\tau_i \dagger T_n$.

We define a function μ for renaming tasks, $\mu : T \to \{1, 2, \ldots, m\}$, $\mu(T_i) = j,\ 1 \leq i \leq n,\ 1 \leq j \leq m$ gives the "name" j in $\mathbb{T}$ of the task τ_j placed in position i in stack T.

An hybrid transition system $(\mathcal{S}, T, \to)$ for a LIFO scheduler is composed of :

1. a collection of states, $\mathcal{S} = (S, \vec{e}, \vec{r}, \vec{\dot{e}}, \vec{p})$, where:
   (a) S is a control location,
   (b) $\vec{e}$ a vector of clocks counting execution time, where $\vec{e}_j$ is the cumulated execution time for task τ_j,
   (c) $\vec{r}$ a vector for releasing times, where $\vec{r}_j$ is the released time for task τ_j,
   (d) $\vec{\dot{e}}$ a vector indicating those execution clocks which are stopped,
   (e) $\vec{p}$ a vector for preemption where $\vec{p}_j = l$ means that task τ_l preempts τ_j, $\tau_l \dagger \tau_j$, if $l \neq 0$ or that τ_j has never been preempted, otherwise. In particular, in the LIFO scheduler, if task τ_j is at position $k, 1 \leq k < n$ in T, that is $\tau_j = T_k$, then task τ_l is at position $k+1$, that is[1] :

$$\mu(T_k) = j,\ \mu(T_{k+1}) = l,\ \vec{p}_{\mu(T_k)} = \mu(T_{k+1}) \equiv \vec{p}_j = l$$

2. a stack of tasks, $T \subseteq \mathbb{T}$ (with the usual operations **pop**, **top** and **push**).
3. a transition relation $\to$

We give the operations over our transition system; recall that the arrival of a task is captured by the scheduler, who decides over the admission. As a convention, we will use index j for "names" of tasks, $1 \leq j \leq m$ and k for active tasks in T, $1 \leq k \leq n$.

- Task arrival, $\tau_i \uparrow$ ($\tau_i \in \mathbb{T}$ and $\textbf{accept}(\mathcal{S}, T, \tau_i)$):

$$(\mathcal{S}, T) \stackrel{\tau_i \uparrow}{\to} (\mathcal{S}', T')$$

where $T' \equiv \textbf{push}(\tau_i, T)$ and $\mathcal{S}' = (S', \vec{e'}, \vec{r'}, \vec{\dot{e}'}, \vec{p'})$ is:

$$\vec{e'}_j = \begin{cases} 0 & \text{if } i = j \\ \vec{e}_j & \text{otherwise} \end{cases}$$

[1] We could simplify $\vec{p}$ as a boolean vector where $\vec{p}_j = \theta$, and $\theta \in$ [true, false]

$$\vec{r'}_j = \begin{cases} 0 & \text{if } i = j \\ \vec{r}_j & \text{otherwise} \end{cases}$$

$$\vec{p'}_j = \begin{cases} i & \text{if } j = \mu(\mathbf{top}(T)) \\ 0 & \text{if } j = i \\ \vec{p}_j & \text{otherwise} \end{cases}$$

$$\vec{\dot{e}}'_j = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{otherwise} \end{cases}$$

That is, as a new task is accepted, its execution and release clocks are both reset, its rate execution clock is set to 1 to mark it is running, while all other execution clocks are stopped; we mark preemption to the current executing task.

- Task completion: $T_n\downarrow$

$$(\mathcal{S}, T) \overset{T_n\downarrow}{\rightarrow} (\mathcal{S}', T')$$

where $T' = \mathbf{pop}(T)$ and $\vec{e}_{\mu(T_n)} = \vec{r}_{\mu(T_n)} = \perp$, all other variables are unchanged.

- Task resumption: $T_n \nearrow$ (we assume $T_n = \text{top}(T)$ is a task preempted in the past, which regains the processor).

$$(\mathcal{S}, T) \overset{T_n\nearrow}{\rightarrow} (\mathcal{S}', T)$$

where $\mathcal{S}' = (S', \vec{e'}, \vec{r'}, \vec{\dot{e}}', \vec{p'})$ is:

$$\vec{\dot{e}}'_j = \begin{cases} 1 & \text{if } j = \mu(T_n) \\ 0 & \text{otherwise} \end{cases}$$

all other variables remain unchanged.

- Time passing: δ is an elapsed time not enough to finish the current executing task.

$$(\mathcal{S}, T) \overset{\delta}{\rightarrow} (\mathcal{S}', T)$$

where $\mathcal{S}' = (S', \vec{e'}, \vec{r'}, \vec{\dot{e}}', \vec{p'})$ is:

$$\vec{e'}_j = \begin{cases} \vec{e}_j + \delta & \text{if } j = \mu(T_n) \\ \vec{e}_j & \text{otherwise} \end{cases}$$

and $\vec{r'}_j = \vec{r}_j + \delta \, \forall \tau_j, \tau_j \in T$, all other variables remain unchanged.

3.2 LIFO Admittance Test

We propose the following admittance test for our LIFO scheduler:

$$\mathbf{accept}_{\text{LIFO}}(\mathcal{S}, T, \tau_i) \equiv \neg\mathbf{active}(T, \tau_i) \wedge \neg\mathbf{preempted}(T, T_n)$$

that is, we do not accept a task if:

- There is an active instance of the same task, that is

$$\mathbf{active}(T, \tau_i) \equiv i = \mu(T_k) \text{ for some } k, 1 \leq k \leq n$$

- It will preempt an already preempted task, that is

$$\textbf{preempted}(T, T_n) \equiv \vec{p}_{\mu(T_n)} \neq 0$$

For the instant being we do not consider timing constraints; in particular, we are not considering in the acceptance test the fact that a new accepted task may lead to some other tasks in T miss their deadlines. Later, we propose a refinement in that direction.

3.3 Properties of LIFO scheduler

Under the one-preemption assumption the following properties hold:

1. If $\vec{p}_{\mu(T_n)} = 0 \Rightarrow e_{\mu(T_n)} = r_{\mu(T_n)}$
2. $e_{\mu(T_{n-1})} = r_{\mu(T_{n-1})} - r_{\mu(T_n)}$
3. $\forall T_k, T_k \in T, k < n$, we have:
   (a) Preemption: $\textbf{active}(T_k) \wedge \textbf{preempted}(T_k)$
   (b) Time invariant condition:

$$I_{\text{LIFO}}(T) \equiv \bigwedge_{k=1}^{n-1} e_{\mu(T_k)} = r_{\mu(T_k)} - r_{\mu(T_{k+1})}$$

$$I_{\text{LIFO}}(T) \equiv \bigwedge_{k=1}^{n-1} e_{\mu(T_k)} = r_{\mu(T_k)} - r_{\vec{p}_{\mu(T_k)}}$$

   (c) Schedulability: $r_{\mu(T_k)} < D_{\mu(T_k)}$

Property 1 simply says that if the currently executing task T_n has never been preempted since its arrival, then both clocks, $e_{\mu(T_n)}$ and $r_{\mu(T_n)}$ have the same value.

Property 2 is the consequence of preemption. When T_{n-1} was preempted, (i.e. T_{n-1} was executing and hence on top of T), we know by the previous rule, that $e_{\mu(T_{n-1})} = r_{\mu(T_{n-1})}$ and as $r_{\mu(T_n)}$ is set to zero, we can establish the property which is time-invariant while T_{n-1} is suspended. This observation leads by induction to property 3b, which we call the *execution invariant* under a LIFO scheduling policy.

Property 3a says that all tasks in stack T are active and were preempted in the past (except eventually the task in the top).

Property 3c says that all tasks in T are schedulable, (remember our extension of the admittance test will go in that direction).

Example .3 *Let us reconsider the example .2; we can observe that (see figure 4):*

- At $t = 0$ $r_1 = e_1 = 0$, τ_1 begins its execution. $I_{\text{LIFO}}(\tau_1) = true$,
- At $t = 3$, τ_3 arrives, it is accepted and preempts τ_1 (later, we will give an admission test dealing with schedulability conditions). The execution invariant $I_{\text{LIFO}}(\tau_1, \tau_2) \equiv \{e_1 = r_1 - r_3\}$
- At $t = 6$, $\tau_2\uparrow$ and $\tau_2 \dagger \tau_3$. The execution invariant $I_{\text{LIFO}}(\tau_1, \tau_2, \tau_3) \equiv \{e_1 = r_1 - r_3 \wedge e_3 = r_3 - r_2\}$.
- At $t = 8$, τ_2 completes its execution and the scheduler resumes τ_3. The computed time is recovered from the difference $e_3 := r_3 - r_2$. Note that $\vec{p}_3 = 2$ and $e_3 = r_3 - r_{\vec{p}_3}$.
- At $t = 9$, T_4 arrives but it is rejected by the admittance test since: $\textbf{exec}(T) = \tau_3 \wedge \textbf{preempted}(\tau_3)$.

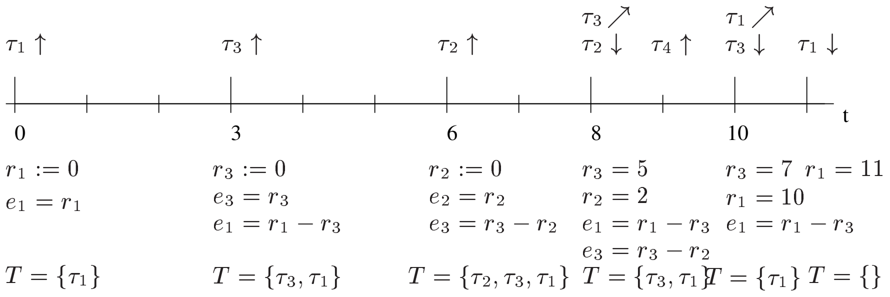

Figure 4: Invariants in LIFO Scheduler

- Finally, at $t = 10$, τ_3 completes and the scheduler resumes τ_1; once again e_1 is recovered from the difference $r_1 - r_3$; τ_1 ends at $t = 11$.

The previous properties motivate the following definition:

Definition .1 *The* **execution invariant** under a LIFO scheduling policy is

$$I_{\text{LIFO}}(T) = \bigwedge_{1 \leq k \leq n-1} e_{\mu(T_k)} = r_{\mu(T_k)} - r_{\mu(T_{k+1})}$$

If the currently executing task has already been preempted, the equation $r_{\mu(T_n)} = e_{\mu(T_n)}$ may not hold and in this case, we cannot simply express $e_{\mu(T_n)}$ as the difference of $r_{\mu(T_n)}$ and $r_{\mu(T_{n+1})}$. So for the time being, we still retain our assumption of one-preemption. To ensure that $r_{\mu(T_n)} = e_{\mu(T_n)}$ holds, we can constrain the predicate **accept**$(\mathcal{S}, T, \tau_i)$ for every task τ_i by the constraint

$$e_{\text{exec}(T)} = r_{\text{exec}(T)}$$

3.4 Reachability Analysis in LIFO Scheduler

Let Ψ be the set of formulas generated by the following grammar:

$$\psi ::= x - y \leq d \mid \psi \wedge \psi \mid \neg\psi \mid \exists x.\psi$$

where $x, y \in \mathcal{C}$ are clocks and $d \in \mathbb{Q}$ is a rational constant.

To facilitate notation, we will skip in this analysis the use of the function μ, and replace it by the position a task occupies in the stack. Remember then that when saying, for instance, e_k we really mean the execution clock e of the task which is in the k position in the stack, that is $e_{\mu(T_k)}$.

Let ϕ be a constraint characterizing a set of states. We define $T_{n+1}\uparrow(\phi)$ to be the set of states reached when task T_{n+1} arrives, that is:

$$T_{n+1}\uparrow(\phi) = \{s' : \exists s \in \phi.\ s \overset{T_{n+1}\uparrow}{\rightarrow} s'\}$$

Let ϕ be of the form $I_{\text{LIFO}}(T) \wedge \psi$, with $\psi \in \Psi$ a quantifier free formula. Without loss of generality, we can assume that either :

1. T_{n+1} is rejected:
$$\psi \implies \neg\textbf{accept}(\psi, T, T_{n+1})$$
in this case, $T_{n+1}\uparrow(\phi) \equiv \phi$ and the system moves to an error state.

2. T_{n+1} is accepted:
$$\psi \implies \textbf{accept}(\psi, T, T_{n+1})$$

   We have that:

$$T_{n+1}\!\uparrow\!(\phi) \equiv I_{\text{LIFO}}(T \oplus \{T_{n+1}\}) \wedge e_{n+1} = r_{n+1} = 0 \wedge \exists e_n.\psi$$

   Moreover, since $e_n = r_n$, $I_{\text{LIFO}}(T \oplus \{T_{n+1}\})$ contains the equality $e_n = r_n - r_{n+1}$, we have that:

$$T_{n+1}\!\uparrow\!(\phi) \equiv I_{\text{LIFO}}(T + \{T_{n+1}\}) \wedge e_{n+1} = r_{n+1} = 0 \wedge \psi'$$

   Hence, $T_{n+1}\!\uparrow\!(\phi)$ has the same structure than ϕ, that is, it is the conjunction of an execution invariant and a formula in Ψ. Moreover, if ψ is a quantifier-free formula, that is, a difference constraint (or DBM), we have that $\exists e_n.\psi$ is indeed a DBM. Note that ψ is a formula containing clocks measuring release times and only *one* execution clock (that of the task on top).

Then we have:

Proposition .1 *Let ϕ be of the form $I_{\text{LIFO}} \wedge M$, where M is a* DBM *and I_{LIFO} is a one preemption* LIFO *execution invariant, then, $T_{n+1}\!\uparrow(\phi)$ has the same structure as ϕ.*

Now, let $\phi\nearrow$ the set of states reached from ϕ by letting time advance, that is:

$$\phi\nearrow = \{s' : \exists s \in \phi, \delta \geq 0.s \xrightarrow{\delta} s'\}$$

Clearly, if ϕ is of the form $I_{\text{LIFO}}(T) \wedge M$, we have that

$$\phi\nearrow = (I_{\text{LIFO}}(T) \wedge M)\nearrow = I_{\text{LIFO}} \wedge M\nearrow$$

Proposition .2 *Let ϕ be of the form $I_{\text{LIFO}} \wedge M$, where M is a* DBM *and I_{LIFO} is a one preemption* LIFO *execution invariant, then, $\phi\nearrow$ has the same structure as ϕ.*

Thus, given a sequence of task arrivals $T_1\!\uparrow, \ldots, T_n\!\uparrow$, the set of reached states can be represented by the conjunction of the execution invariant $I_{\text{LIFO}}(T)$, characterizing the already executed time of the suspended tasks, namely $T_1, \ldots, T_{n-1}$, and a DBM M, characterizing the relationship between the corresponding released times and the equality $e_n = r_n$.

A DBM M has the following form:

$$\begin{array}{lllllll}
 & u & r_1 & r_2 & \ldots & r_n & e_n \\
u & - & M_{ur_1} & M_{ur_2} & \ldots & M_{ur_n} & M_{ue_n} \\
r_1 & M_{r_1u} & - & M_{r_1r_2} & \ldots & M_{r_1r_n} & M_{r_1e_n} \\
r_2 & M_{r_2u} & M_{r_2r_1} & - & \ldots & M_{r_2r_n} & M_{r_2e_n} \\
\vdots & & & & & & \\
r_n & M_{r_nu} & M_{r_nr_1} & M_{r_nr_2} & \ldots & - & M_{r_ne_n} \\
e_n & M_{e_nu} & M_{e_nr_1} & M_{e_nr_2} & \ldots & M_{e_nr_n} & -
\end{array}$$

As $e_n = r_n$, then $M_{xe_n} = M_{xr_n}$ and $M_{e_nx} = M_{r_nx}$, so from here on we omit e_n in M.

In our case, M is constructed in a very particular way and therefore has a special structure. Let us analyse it:

Event	*Equation*	*Explanation*

$T_1\uparrow$	$\emptyset$		
$T_2\uparrow$	$M_{r_1r_2} = M_{r_1u}$	$\blacklozenge\ e_1 = r_1 - r_2 \leq M_{r_1r_2} = M_{r_1u} \because r_2 = u$	(1)
$T_3\uparrow$	$M_{r_2r_3} = M_{r_2u}$	$\blacklozenge\ e_2 = r_2 - r_3 \leq M_{r_2r_3} = M_{r_2u} \because r_3 = u$	(2)
	$M_{r_1r_3} = M_{r_1r_2} + M_{r_2r_3}$	$(e_1 = r_1 - r_2 \leq M_{r_1r_2} \wedge e_2 = r_2 - r_3 \leq M_{r_2r_3}) \Rightarrow$	
		$r_1 - r_3 \leq M_{r_1r_3} = M_{r_1r_2} + M_{r_2r_3}$	(3)
$T_4\uparrow$	$M_{r_3r_4} = M_{r_3u}$	$\blacklozenge\ e_3 = r_3 - r_4 \leq M_{r_3r_4} = M_{r_3u} \because r_4 = u$	(4)
	$M_{r_2r_4} = M_{r_2r_3} + M_{r_3r_4}$	$(e_2 = r_2 - r_3 \leq M_{r_2r_3} \wedge e_3 = r_3 - r_4 \leq M_{r_3r_4}) \Rightarrow$	
		$r_2 - r_4 \leq M_{r_2r_4} = M_{r_2r_3} + M_{r_3r_4}$	(5)
	$M_{r_1r_4} = M_{r_1r_2} + M_{r_2r_3} + M_{r_3r_4}$	$(e_1 = r_1 - r_2 \leq M_{r_1r_2} \wedge e_2 = r_2 - r_3 \leq M_{r_2r_3}$	
		$\wedge e_3 = r_3 - r_4 \leq M_{r_3r_4}) \Rightarrow$	
		$r_1 - r_4 \leq M_{r_1r_4} = M_{r_1r_2} + M_{r_2r_3} + M_{r_3r_4}$	(6)

We can observe that equality 3 is deduced from 1 and 2; 5 from 2 and 4 and finally 6 from 1, 2 and 4. So we see that the matrix is constructed from a set of **base formulae** corresponding to the difference of the task being executed and that which preempts it, while all other differences can be constructed from this base set. Base formulae are marked with a ♦.

In general, when T_n arrives:

$$M_{r_{n-1}r_n} = M_{r_{n-1}u}$$

and

$$M_{r_{n-1}e_{n-1}} = M_{e_{n-1}r_{n-1}} = 0$$

As T_n preempts T_{n-1} and

$$M_{r_jr_n} = \sum_{i=j}^{n-1} M_{r_ir_{i+1}},\ j < n-1 \tag{7}$$

Equation 7 may be re-written as a recursive formula:

$$M_{r_jr_n} = M_{r_jr_{n-1}} + M_{r_{n-1}r_n},\ j < n-1 \tag{8}$$

and

$$M_{r_ju} = M_{r_jr_n} + M_{r_nu}$$

What do these differences mean? Figure 5 shows a geometric interpretation of the following relations:

$$\begin{array}{ccccc}
e_1 & = & r_1 - r_2 & \leq & M_{r_1r_2} \\
e_2 & = & r_2 - r_3 & \leq & M_{r_2r_3} \\
\vdots & \vdots & \vdots & \vdots & \\
e_{n-1} & = & r_{n-1} - r_n & \leq & M_{r_{n-1}r_n} \\
\hline
\sum_{1\leq i<n} e_i & = & \sum_{1\leq i<n} r_i - r_{i+1} & \leq & \sum_{1\leq i<n} M_{r_ir_{i+1}}
\end{array}$$

$$\sum_{1\leq i<n} e_i \ = \ r_1 - r_n \ \leq \ \sum_{1\leq i<n} M_{r_ir_{i+1}} \tag{9}$$

On the other hand, we know that:

$$r_1 - r_n \leq M_{r_1r_n} \tag{10}$$

From the expression (9) and (10) and (7) we can deduce that:

$$\sum_{1\leq i<n} e_i = r_1 - r_n \leq \sum_{1\leq i<n} M_{r_ir_{i+1}} = M_{r_1r_n}$$

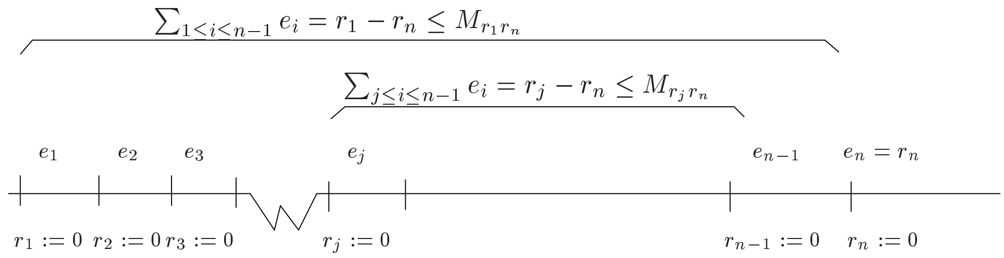

Figure 5: Clock Differences in LIFO Scheduler

The expression 9 can be generalized as:

$$\sum_{j \leq i < n} e_i = r_j - r_n \leq M_{r_j r_n}$$

That is, when T_n arrives we have that:

$$M_{r_{n-1} r_n} = M_{r_{n-1} u} \quad (11)$$
$$M_{r_{n-1} e_{n-1}} = M_{e_{n-1} r_{n-1}} = 0 \quad (12)$$

and for all $j < k \leq n$,

$$M_{r_j u} = M_{r_j r_k} + M_{r_k u} \quad (13)$$
$$M_{u r_j} = M_{u r_k} + M_{r_k r_j} \quad (14)$$
$$M_{r_j r_k} = M_{r_j r_{k-1}} + M_{r_{k-1} r_k} \quad (15)$$
$$M_{r_k r_j} = M_{r_k r_{k-1}} + M_{r_{k-1} r_j} \quad (16)$$

Let us call a DBM M that satisfies properties 11 to 16 a **nice** DBM.

We have therefore proved that:

Proposition .3 *$T_{n+1}\!\uparrow(\cdot)$ and $\nearrow(\cdot)$ preserve nicety.*

We have shown so far that task arrival and time passing preserve the structure of the symbolic characterization of the state space for a LIFO scheduler if tasks are accepted only under the one-preemption restriction. The question that arises then, is whether task completion has the same property. If this is the case, we have a complete symbolic characterization of the state space of such schedulers. Indeed, the answer is yes, though the reasoning is a bit more involved.

Let $T_n\!\downarrow(\phi)$ be the set of states reachable from ϕ when T_n terminates:

$$T_n\!\downarrow\!(\phi) = \{s' : \exists s \in \phi . s \stackrel{T_n\downarrow}{\to} s'\}$$

Let ϕ be of the form $I_{\text{LIFO}}(T) \wedge M$ and G_n be the interval $[E_n^{\min}, E_n^{\max}]$. We have that:

$$\begin{aligned} T_n\!\downarrow(\phi) &\equiv \exists e_n, r_n . I_{\text{LIFO}}(T) \wedge M \wedge G_n \\ &\equiv I_{\text{LIFO}}(T - \{T_n\}) \wedge \exists r_n . e_{n-1} = r_{n-1} - r_n \wedge \exists e_n . (M \wedge G_n) \\ &\equiv I_{\text{LIFO}}(T - \{T_n\}) \wedge M'[r_n \leftarrow r_{n-1} - e_{n-1}] \end{aligned}$$

where $M' \equiv \exists e_n . (M \wedge G_n)$, that is we eliminate e_n from M, since we do not need it. The question is: " Is M' still a nice DBM matrix?"

We have to show now that $M'[r_n \leftarrow r_{n-1} - e_{n-1}]$ is equivalent to a nice DBM.

When substituting r_n by $r_{n-1} - e_{n-1}$ in M we get:

$$\text{from } r_n - r_{n-1} \leq M_{r_n r_{n-1}} \Rightarrow -e_{n-1} \leq M_{r_n r_{n-1}} \tag{17}$$

$$\text{from } r_n - r_j \leq M_{r_n r_j} \Rightarrow r_{n-1} - e_{n-1} - r_j \leq M_{r_n r_j} \tag{18}$$

which is not a difference constraint, but from (17) and

$$r_{n-1} - r_j \leq M_{r_{n-1} r_j}$$

we derive that

$$r_{n-1} - e_{n-1} - r_j \leq M_{r_n r_{n-1}} + M_{r_{n-1} r_j}$$

Since M is nice, we have that:

$$M_{r_n r_j} = M_{r_n r_{n-1}} + M_{r_{n-1} r_j}$$

which means that (18) is an implied constraint and it can be eliminated. The same applies for all the non-difference constraints that appear after subsitution. Since no other new constraints on released time variables appear, nicety is preserved.

In summary:

Proposition .4 *Let ϕ be of the form $I \wedge M$, where M is a nice* DBM *and I is a* LIFO *execution invariant. Then, $T_n\!\downarrow(\phi)$ has the same structure than ϕ.*

Since all variables are bounded, the above results imply the following:

Theorem .1 *The symbolic reachability graph of a system of tasks for a* LIFO *scheduler satisfying the one-preemption constraint is finite.*

Hence, the reachability (and therefore the schedulability) problem for our class of systems is decidable. More importantly, our result gives a fully symbolic characterization of the reach-set.

3.5 Refinement of LIFO Admittance Test

We have "skipped" the analysis of deadlines; in this section we give a refinement of our LIFO admittance test.

We propose to test at $\tau_i\!\uparrow$ the predicate:

$$\textbf{accept}_{\text{LIFO}}(T, \tau_i) \equiv \neg\textbf{active}(T, \tau_i) \wedge \neg\textbf{preempted}(T, T_n) \wedge \textbf{schedulable}(T, \tau_i)$$

that is, we do not accept a task if:

- There is an active instance of the same task, that is **active**$(T, \tau_i) \equiv i = \mu(T_k)$ for some $k, 1 \leq k \leq n$.
- It will preempt an already preempted task, that is **preempted**$(T, T_n) \equiv \vec{p}_{\mu(T_n)} \neq 0$.
- It will miss its deadline or cause other tasks in T miss their deadlines, i.e,

$$\textbf{schedulable}(T, \tau_i) \equiv \forall \tau_j \in \{T \cup \tau_i\},\ r_j + B_j + (E_j^M - e_j) \leq D_j \tag{19}$$

  that is we must calculate how many units of time the r_j's will be shifted after computing those tasks which have higher priorities, including τ_i, and of course how many units of time must at most execute τ_j. The time a task τ_j will be suspended as a consequence of the execution of higher priority tasks is called the *blocking time*, B_j.

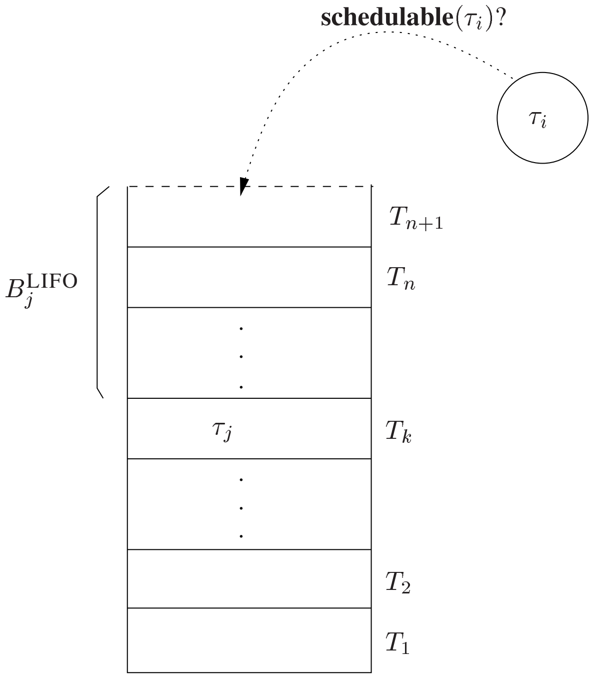

Figure 6: Tasks in a LIFO scheduler

We should note that a LIFO scheduler is in some kind a dynamic priority protocol, since the arrival of a new task will, in principle, preempt the currently executing one. That is, priorities are given by task arrival and hence by stack position, $\pi_{\mu(T_1)} < \pi_{\mu(T_2)} \ldots < \pi_{\mu(T_n)}$.

At task arrival, the schedulability test must assure no deadline missing for all active tasks, which, due to the one-preemption hypothesis it does not necessarily mean that the new task will be accepted.

For each task $T_k \in T$, its blocking time when a new task τ_i arrives can be calculated as follows $(j = \mu(T_k), 1 \leq k \leq n0)$:

$$B_j^{\text{LIFO}} = \sum_{\pi_{\tau'} > \pi_j} (E_{\tau'} - e_{\tau'})$$

under the LIFO scheduler we know that $e_j = r_j - r_{\vec{p}_j} \; \forall \, k < n, T_k \in T, \mu(T_k) = j$. Obviously $e_{\mu(T_n)} = r_{\mu(T_n)}$ (if not, τ_i would violate the one preemption hypothesis and hence it should not be accepted) and $e_{\mu(T_{n+1})} = 0$, so (see figure 6 for a graphical interpretation of these formula):

$$\begin{aligned}
B_j^{\text{LIFO}} &= \sum_{l=k+1}^{n+1} (E^M_{\mu(T_l)} - e_{\mu(T_l)}) \\
B_j^{\text{LIFO}} &= \sum_{l=k+1}^{n-1} (E^M_{\mu(T_l)} - e_{\mu(T_l)}) + (E^M_{\mu(T_n)} - e_{\mu(T_n)}) + E^M_{\mu(T_{n+1})}
\end{aligned}$$

Using our execution time invariant and the fact that $\tau_{\mu(T_n)}$ cannot have been preempted (if this were the case, then surely τ_i cannot be accepted), we have:

$$
\begin{aligned}
B_j^{\text{LIFO}} &= \sum_{l=k+1}^{n-1} (E^M_{\mu(T_l)} - (r_{\mu(T_l)} - r_{\mu(T_{l+1})})) + (E^M_{\mu(T_n)} - r_{\mu(T_n)}) + E^M_{\mu(T_{n+1})} \\
B_j^{\text{LIFO}} &= \sum_{l=k+1}^{n+1} (E^M_{\mu(T_l)}) - \sum_{l=k+1}^{n-1} (r_{\mu(T_l)} - r_{\mu(T_{l+1})}) - r_{\mu(T_n)} \\
B_j^{\text{LIFO}} &= \sum_{l=k+1}^{n+1} (E^M_{\mu(T_l)}) - (r_{\mu(T_{k+1})} - r_{\mu(T_n)}) - r_{\mu(T_n)} \\
B_{\mu(T_k)}^{\text{LIFO}} &= \sum_{l=k+1}^{n+1} (E^M_{\mu(T_l)}) - r_{\mu(T_{k+1})}
\end{aligned} \tag{20}
$$

Replacing in 19 with 20, using $j = \mu(T_k)$ we have:

$$
\forall T_k \in \{T \cup \tau_i\},\ \ r_j + \sum_{l=k+1}^{n+1} (E^M_{\mu(T_l)}) - r_{\mu(T_{k+1})} + (E^M_j - e_j) \leq D_j
$$

which can be rewritten as

$$
\forall T_k \in \{T \cup \tau_i\},\ \sum_{l=k}^{n+1} (E^M_{\mu(T_l)}) + (r_{\mu(T_k)} - r_{\mu(T_{k+1})} - e_{\mu(T_k)}) \leq D_{\mu(T_k)}
$$

but $e_{\mu(T_k)} = r_{\mu(T_k)} - r_{\mu(T_{k+1})}$ and the test is reduced to:

$$
\forall T_k \in \{T \cup \tau_i\},\ \sum_{l=k}^{n+1} (E^M_{\mu(T_l)}) \leq D_{\mu(T_k)} \tag{21}
$$

This test is pessimistic, since we are using the worst case execution time for tasks in order to calculate blocking times and schedulability. If we consider applications where execution times are *controllable*, that is, applications where we can influence in someway the time spent in the execution, we could use minimum execution times. This could be acceptable for applications where execution times are related to some quality of service, for instance performing an approximative calculus instead of an exact one or composing an image in different qualities.

On the contrary, if execution times are *uncontrollable*, then we need maximum execution times, since we must accept to work under a worst case perspective.

4 EDF Scheduling

Let us analyse another scheduling policy, *earliest deadline first*, EDF, which is considered to be optimal in the sense that if a set of tasks is schedulable under some policy, then it is also schedulable under EDF, [9].

Under this policy we know that tasks are chosen by the scheduler according to their deadlines, with that having the shortest deadline being in execution. The policy is generally preemptive, but we could imagine an EDF scheduler not preemptive. We will consider a one-preemption EDF scheduler.

Let T be the set of active tasks ordered by deadline; in fact, T is a queue and by convention T_n is the *head* of the queue and hence currently executing; the rest of T is the *tail*. We also assume the existence of the renaming function μ, as explained for LIFO and the universe $\mathbb{T}$ of tasks, such that $T \subseteq \mathbb{T}$.

Conceptually speaking, the EDF scheduler is quite simple, when a new task arrives it is accepted or rejected by the scheduler for schedulability reasons and if accepted it is inserted in T according to its deadline. Once a task is finished, the scheduler can choose the next one, which is that behind the head,

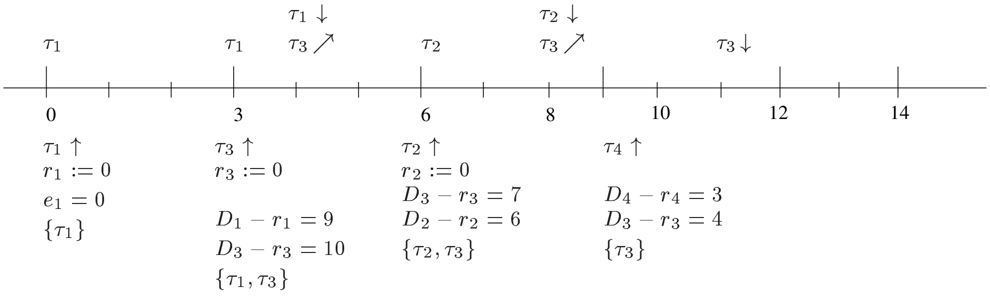

Figure 7: One preemption EDF Scheduler

and so on. Note that a task can be accepted and put in T in some position according to its deadline, not necessarily preempting the task in the head of T.

A one-preemption EDF scheduler works quite similarly to an EDF scheduler except that if a new task τ must preempt the currently executing one which has already been preempted, then τ is rejected even if the whole system is schedulable. Once again, the reason to do this is our manipulation of clocks.

Example .4 *Let us consider a set* $\mathbb{T} = \{\tau_1(4, 12), \tau_2(2, 6), \tau_3(5, 10), \tau_4(1, 3)\}$. *In figure 7 we see the behaviour under a one preemption* EDF *scheduling policy. Some remarks:*

- *At time* $t = 3$, $\tau_3\uparrow$ *but its deadline (10) is longer than* τ_1*'s, so it waits in the queue.*
- *At time* $t = 4$, τ_1 *finishes and* τ_3, *gains the processor.*
- *At time* $t = 6$, $\tau_2 \uparrow$, *and its deadline (6) is shorter than* τ_3*'s, so it preempts it. This is the first preemption for* τ_3 *since it is its first execution.* τ_3 *rejoins the queue.*
- *At time* $t = 8$ τ_2 *finishes its execution and* τ_3 *resumes its execution.*
- *At time* $t = 9$ *task* τ_4 *arrives and its deadline (3) is shorter than* τ_3*'s (4) so it should preempt it* but τ_3 *had already been preempted, son our scheduler rejects* τ_4.
- *At time* $t = 11$ τ_3 *finishes.*

As usual, we distinguish the arrival of a task $\tau \uparrow$, from the resuming of a task $\tau \nearrow$. Note that in our one-preemption EDF scheduler is not optimal, since the system is feasable (we could have accepted τ_4) but we rejected it.

4.1 EDF Transition Model

We model an EDF application as a transition system $\mathcal{S}, T, \rightarrow)$ composed of:

1. A collection of states $\mathcal{S} = (S, \vec{e}, \vec{r}, \vec{p}, \vec{\dot{e}}, \vec{w})$, where S, $\vec{e}$, $\vec{r}$, $\vec{p}$, $\vec{\dot{e}}$ have the same meaning as for LIFO and $\vec{w}$ is an auxiliary vector of clocks, $\vec{w}_j$ notes the time when task τ_j begins its execution, which is different from its released (or arrival) time; note that in LIFO scheduling, the most recent arrived task preempts the executing one, so, immediately accepted, a task begins execution. Under EDF scheduling this is not the case, since an accepted task may go somewhere in the queue, being its execution delayed until more urgent tasks finish their executions. Clocks w's will serve to note this gap in time.
2. A collection of active tasks T
3. A transition relation $\rightarrow$.

We introduce the operations in our transition system; we note as τ_c the currently executing task, that is $\tau_c = \textbf{exec}(T) = \tau_{\mu(T_n)}$

- Task arrival, $\tau_i\!\uparrow$ (remember: $\tau_i \in \mathbb{T}$ and $\textbf{accept}(\mathcal{S}, T, \tau_i)$):

$$(\mathcal{S}, T,) \overset{\tau_i\uparrow}{\to} (\mathcal{S}', T')$$

where

$T' \equiv T \oplus \tau_i$ and $\oplus$ is an ordered insert operation over T for τ_i according to its deadline. $\mathcal{S}' = (S', \vec{e}', \vec{r}', \vec{p}', \dot{\vec{e}}', \vec{w}')$ is defined as:

$$\vec{e'}_j = \begin{cases} \vec{e}_j & \text{if } j \neq i \\ 0 & \text{for } j = i \wedge D_c - r_c > D_i \text{ (Execute } \tau_i) \\ \perp & \text{otherwise} \end{cases}$$

$$\vec{w'}_j = \begin{cases} \vec{w}_j & \text{if } j \neq i \\ 0 & \text{if } j = i \wedge D_c - r_c > D_i \text{ (Execute } \tau_i) \\ \perp & \text{otherwise} \end{cases}$$

$$\vec{r'}_j = \begin{cases} \vec{r}_j & \text{if } j \neq i \\ 0 & \text{otherwise} \end{cases}$$

$$\vec{p'}_j = \begin{cases} i & \text{if } j = c \wedge D_c - r_c > D_i \text{ (} \tau_i \dagger \tau_c) \\ 0 & \text{if } j = i \\ \vec{p}_j & \text{otherwise} \end{cases}$$

$$\dot{\vec{e}}'_j = \begin{cases} 1 & \text{if } j = i \wedge D_c - r_c > D_i \text{ (Execute } \tau_i) \\ 0 & \text{if } j = i \wedge D_c - r_c \leq D_i \text{ (Do not start } \tau_i) \\ 0 & \text{if } j = c \wedge D_c - r_c > D_i \text{ (Stop } \tau_c) \\ \vec{e}_j & \text{otherwise} \end{cases}$$

- Task completion: $T_c\!\downarrow$

$$(\mathcal{S}, T) \overset{T_c\downarrow}{\to} (\mathcal{S}', T')$$

where $T' = \textbf{tail}(T)$ and

$$\vec{e'}_j = \begin{cases} \perp & \text{if } j = c \\ \vec{e}_j & \text{otherwise} \end{cases}$$

$$\vec{w'}_j = \begin{cases} \perp & \text{if } j = c \\ \vec{w}_j & \text{otherwise} \end{cases}$$

$$\dot{\vec{e}}'_j = \begin{cases} \perp & \text{if } j = c \\ 0 & \text{otherwise} \end{cases}$$

$$\vec{r}_j = \begin{cases} \perp & \text{if } j = c \wedge \nexists p_{\mu(T_k)} = c, \; 1 \leq k < n \\ \vec{r}_j & \text{otherwise} \end{cases}$$

Variable $\vec{p}$ remains unchanged.

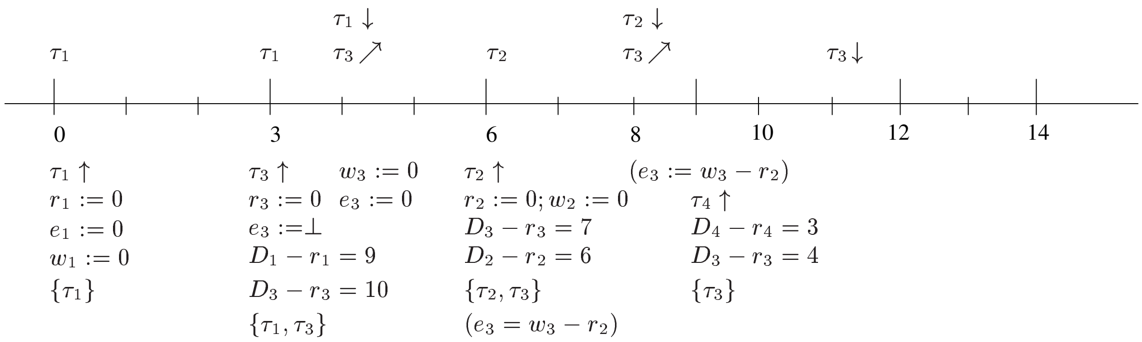

Figure 8: Usage of difference constraints

- Task resumption: $\tau_i \nearrow$ (we assume $\tau_i = \textbf{top}(T)$, arrived and eventually preempted in the past).

$$(\mathcal{S}, T) \xrightarrow{\tau_i \nearrow} (\mathcal{S}', T)$$

where

$$\vec{e'}_j = \begin{cases} 0 & \text{if } j = i \wedge p_i = 0 \ (\tau_i \text{ was never executed}) \\ \vec{e}_j & \text{otherwise} \end{cases}$$

$$\vec{w'}_j = \begin{cases} 0 & \text{if } j = i \wedge p_i = 0 \ (\tau_i \text{ was never executed}) \\ \vec{w}_j & \text{otherwise} \end{cases}$$

$$\dot{\vec{e}}'_j = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{otherwise} \end{cases}$$

Variables $\vec{p}$ and $\vec{r}$ remain unchanged.

- Time passing: δ is an elapsed time not enough to finish the current executing task.

$$(\mathcal{S}, T) \xrightarrow{\delta} (\mathcal{S}', T)$$

where

$$\vec{e'}_j = \begin{cases} \vec{e}_j + \delta & \text{if } j = c \\ \vec{e}_j & \text{otherwise} \end{cases}$$

and

$$\vec{w'}_j = \begin{cases} \vec{w}_j + \delta & \text{if } j = c \\ \vec{w}_j + \delta & \text{if } \vec{p}_j \neq 0 \\ \perp & \text{otherwise} \end{cases}$$

$\vec{r'}_j = \vec{r}_j + \delta \ \forall \tau_j, \tau_j \in T$ and variables $\vec{p}$ and $\vec{e}$ remain unchanged.

For the instant being, our operations do not show the utility of defining the new auxiliary clocks $\vec{w}$; although this is explained in the next section, let us give an example of their usage.

The automata model defined behind our transition system is a SWA where clocks $\vec{e}$ are stopped at preemption time. We want to eliminate this operation and replace it difference constraint using $\vec{w}$, as we have done for a LIFO scheduler.

In figure 8 we show example 7 using $\vec{w}$; we can see that:

- At time $t = 3$, $\tau_3 \uparrow$, $p_3 = r_3 := 0, w_3 = e_3 = \perp$, and τ_3 joins the queue.
- At time $t = 4$, $\tau_3 \nearrow$ and we set $w_3 := 0$ (note $r_3 = 1$).
- At time $t = 6$, $\tau_2 \uparrow$ and $\tau_2 \dagger \tau_3$; we see that e_3 can be expressed as the difference $w_3 - r_2$ and we see the utility of variable $\vec{w}$, since we could not express the value e_3 as $r_3 - r_2$, as we have done for LIFO, since τ_3 arrived and was not immediately executed; we need another clock to mark the first execution of τ_3. Observe that $p_3 := 2$.
- At time $t = 8$, $\tau_2 \downarrow$ and $\tau_3 \nearrow$; e_3 is recovered from the invariant difference $w_3 - r_2$.
- At time $t = 9$, $\tau_4 \uparrow$ and even if its deadline priority is shorter than τ_3's, it cannot preempt it, ($p_3 = 2 \neq 0$).
- At time $t = 11$, τ_3 finishes.

4.2 EDF Admittance Test

As in LIFO, each time a new task, say τ_i, arrives, we perform an acceptance test according to EDF and one-preemption policy. For EDF we propose:

$$\textbf{accept}_{\text{EDF}}(T, \tau_i) \equiv \neg\textbf{active}(T, \tau_i) \wedge \neg\textbf{preempted}(T, \tau_i)$$

that is, we do not accept a task if:

- There is an active instance of the same task:

$$\textbf{active}(T, \tau_i) \equiv (\exists\, T_k \in T.\, 1 \leq k \leq n)(\mu(T_k) = i \vee p_{\mu(T_k)} = i)$$

- It will preempt an already preempted task (in fact τ_c) :

$$\textbf{preempted}(T, \tau_i) \equiv p_c \neq 0 \wedge D_c - r_c > D_i$$

The first term, rejects a new instance of an uncompleted task or a task whose release clock is still active; the second one deals with the one preemption hypothesis under EDF which is rather tricky, since a new task may have a shorter deadline than the currently executing one, but the latter has already been preempted in the past, so the new task is rejected (even if there is enough time to execute it) or the new task may go beneath τ_c (which was not possible under LIFO).

Later, we give a refinement of this admittance test, considering deadlines, execution times and system state.

4.3 Properties of EDF scheduler

Let T be the set of active tasks, with $\tau_c = \tau_{\mu(T_n)} = \textbf{head}(T)$ the task under execution. We can enumerate the following properties:

1. if $p_j = 0 \Rightarrow e_j = w_j, \forall \tau_j, \tau_j \in T$
2. if $\exists\, p_j = c \Rightarrow e_j = w_j - r_{p_j} \equiv e_j = w_j - r_c$
3. if $\tau_c \nearrow \wedge p_c = 0 \Rightarrow e_c = w_c = \perp \wedge \neg\exists p_j = c$
4. $\forall p_j \neq 0, j \neq c \;\Rightarrow e_j = w_j - r_{p_j}$

Property 1 says that a task τ_j in T which has never been preempted respects $e_j = w_j = \perp$. In fact, if $\tau_j \in T$ and $\tau_j \neq \tau_c$, then τ_j arrived in the past, its deadline was not urgent enough to preempt the currently executing task, and it was put in the queue according to its deadline with $e_j = w_j = \perp$; on the contrary if $\tau_j = \tau_c$ and it has never been preempted, then $e_j = w_j \geq 0$.

Property 2 is a consequence of preemption; if $p_j = c$ it means that τ_c preempted τ_j, in fact, when τ_j was running, $p_j = 0$ (one-preemption assumption) which implies $e_j = w_j$ (property 1); as τ_j was preempted by τ_c, its computation time can be put as $e_j = w_j - r_c$ (since $r_c = 0$ when τ_c arrived). As time passes, while $\dot{e}_i = 0$, $e_j = (w_j + \delta) - (r_c + \delta) = w_j - r_c$. This property shows that execution times can be re-written as differences of some clocks for those stopped tasks.

Property 3 is a consequence of the EDF policy. It means that τ_c resumes but it had never been preempted; so the scenario is as follows: when τ_c arrived, its deadline was longer than that of the currently executing task and hence, it was put in the queue, but never executed, so $e_c = w_c = \perp$; as it has not executed, it could not have preempted any other task (in particular the one executing at its arrival time). Note that τ_c can be preempted during its execution.

The last property 4 is our execution invariant for EDF, which says that for all preempted tasks, (except the current executing task), we can express its computed time as a difference. This is an extension of property 2 and 3, since there may be tasks in T never preempted and never executed. This is a great difference compared to LIFO. This property can be put as:

$$I_{\text{EDF}}(T) \equiv \bigwedge_{\tau_j \in T, p_j \neq 0} e_j = w_j - r_{p_j}$$

recall that $e_j = w_j = \perp$, if $\tau_j \in T \wedge p_j = 0$.

For the current executing task, even if preempted in the past, property 4 does not hold since its execution clock is running.

Remark Note that property 4 obliges to keep clock r_{p_j} even if task τ_{p_j} has already finished; for the same reason, we cannot accept a new instance of this task if τ_j is still active. This is a restriction of our model (taken into account by the admittance test), which could be relaxed if we create a "preemptable clock" for each task instance that preempts; a rather costly solution.

We conclude the section with a theorem, analogous to that give for a LIFO scheduler, without proof, since we will give a detailed proof of the general case in section 5.

Theorem .2 *The symbolic reachability graph of a system of tasks for an* EDF *scheduler satisfying the one-preemption constraint is finite.*

4.4 Refinement of EDF Admittance Test

As in LIFO, each time a new task, say τ_i, arrives, we perform an acceptance test according to EDF and one-preemption policy. For EDF we propose:

$$\textbf{accept}_{\text{EDF}}(T, \tau_i) \equiv \neg\textbf{active}(T, \tau_i) \wedge neg\textbf{preempted}(T, \tau_i) \wedge \textbf{schedulable}(T, \tau_i)$$

The first two predicates have already been explained; in this section, we develop a test regarding execution times, deadlines and system state. The question is 'will the new arrived task, if accepted, cause other tasks in T miss their deadlines?

In principle, the predicate **schedulable** is:

$$\textbf{schedulable}(T, \tau_i) \equiv \forall \tau_j \in \{T \cup \tau_i\},\ r_j + B_j + (E_j^M - e_j) \leq D_j \tag{22}$$

where the blocking time for a task τ_j which is in position k of T is expressed as:

$$B_j^{EDF} = \sum_{l>k}^{n+1} (E^M_{\mu(T_l)} - e_{\mu(T_l)})$$

Under the EDF scheduler we know that for those $\tau_j \in T$ preempted in the past, we have $e_j = w_j - r_{p_j}$ and for those τ_j's never preempted, $e_j = w_j = \perp$, so the precedent expression can be split into:

$$\begin{aligned} B_j^{EDF} &= \sum_{l=k+1, p_{\mu(T_l)}\neq 0}^{n-1} (E^M_{\mu(T_l)} - e_{\mu(T_l)}) + (E^M_{\mu(T_n)} - e_{\mu(T_n)}) + \\ & \sum_{l=k+1, p_{\mu(T_l)}=0}^{n-1} E^M_{\mu(T_l)} + E^M_{\mu(T_{n+1})} \end{aligned}$$

using the equality for preempted task we have:

$$\begin{aligned} B_j^{EDF} &= \sum_{l=k+1, p_{\mu(T_l)}\neq 0}^{n-1} (E^M_{\mu(T_l)} - (w_{\mu(T_l)} - r_{p_{\mu(T_l)}})) + (E_{\mu(T_n)} - e_{\mu(T_n)}) + \\ & \sum_{l=k+1, p_{\mu(T_l)}=0}^{n-1} E^M_{\mu(T_l)} + E^M_{\mu(T_{n+1})} \end{aligned}$$

$$B_j^{EDF} = \sum_{l=k+1}^{n+1} E^M_{\mu(T_l)} - \sum_{l=k+1, p_{\mu(T_l)}\neq 0}^{n-1} (w_{\mu(T_l)} - r_{p_{\mu(T_l)}}) - e_{\mu(T_n)} \tag{23}$$

Unfortunately we can say nothing about the second term in 23, so we will try to find some bounds for this term in order to get necessary or sufficient conditions for our schedulability test. We deal with two facts:

1. In 23 we have
$$B_j^{EDF} \leq \sum_{l=k+1}^{n+1} E^M_{\mu(T_l)} \tag{24}$$
since all terms representing execution times are positive. This fact gives a sufficient condition for the admission test; which is too conservative but safe, in the sense that if we accept τ_i we know all tasks in T, including the new one, will be scheduled within their deadlines.

2. Using minimum execution times:
$$B_j^{EDF} \geq \sum_{l=k+1}^{n+1} E^m_{\mu(T_l)} \tag{25}$$
since minimum execution times represent the fastest execution, this bound is a necessary condition, more laxative but unsafe. If after considering minimum execution times, the test of schedulability is **not** satisfied, then no admission is possible; if the test **is** satisfied, then we can accept but we know that there may be some executions leading to error states and hence we need some dynamic control.

Reconsidering our schedulability test 22:

$$\forall\, \tau_j \in \{T \cup \tau_i\},\ r_j + \sum_{l=k+1}^{n+1} E^M_{\mu(T_l)} - \sum_{l=k+1, p_{\mu(T_l)}\neq 0}^{n-1} (w_{\mu(T_l)} - r_{p_{\mu(T_l)}}) + (E^M_j - e_j) \leq D_j$$

$$\forall \tau_j \in \{T \cup \tau_i\}, \sum_{l=k}^{n+1} E^M_{\mu(T_l)} - \sum_{l=k+1, p_{\mu(T_l)} \neq 0}^{n-1} (w_{\mu(T_l)} - r_{p_{\mu(T_l)}}) + (r_j - e_j) \leq D_j \tag{26}$$

Now we analyse 26 to find some bounds; we consider two cases:

1. $p_j = 0$
   - for $k < n$, we know $e_j = 0$ and so 26 becomes:

$$\forall \tau_j \in \{T \cup \tau_i\}, \sum_{l=k}^{n+1} E^M_{\mu(T_l)} - \sum_{l=k+1, p_{\mu(T_l)} \neq 0}^{n-1} (w_{\mu(T_l)} - r_{p_{\mu(T_l)}}) + r_j \leq D_j \tag{27}$$

   - for $k = n$, we know $e_j = w_j \geq 0$ and expression 26 is:

$$(E^M_j + E^M_{\mu(T_{n+1})} - (w_j - r_j) \leq D_j \tag{28}$$

2. $p_j \neq 0$, we know $e_j = w_j - r_{\mu(T_{k+1})}$ so 26 becomes:

$$\forall \tau_j \in \{T \cup \tau_i\}, \sum_{l=k}^{n+1} E^M_{\mu(T_l)} - \sum_{l=k, p_{\mu(T_l)} \neq 0}^{n-1} (w_{\mu(T_l)} - r_{p_{\mu(T_l)}}) + r_j \leq D_j \tag{29}$$

Now considering our bounds 24 and 25 we have:

- $$\forall \tau_j \in \{T \cup \tau_i\}, \sum_{l=k}^{n+1} E^M_{\mu(T_l)} - \sum_{l=k, p_{\mu(T_l)} \neq 0}^{n-1} (w_{\mu(T_l)} - r_{p_{\mu(T_l)}}) + r_j \leq \underbrace{\sum_{l=k}^{n+1} E^M_{\mu(T_l)} + r_j}_{\alpha} \tag{30}$$

  If $\forall \tau_j \in \{T \cup \tau_i\}, \alpha \leq D_j$ then we can accept the new task τ_i. On the contrary, we reject it, but we know we are being too restrictive.

- $$\forall \tau_j \in \{T \cup \tau_i\}, \sum_{l=k}^{n+1} E^M_{\mu(T_l)} - \sum_{l=k, p_{\mu(T_l)} \neq 0}^{n-1} (w_{\mu(T_l)} - r_{p_{\mu(T_l)}}) + r_j \geq \underbrace{\sum_{l=k}^{n+1} E^m_{\mu(T_l)} + r_j}_{\beta} \tag{31}$$

  Once again, if $\exists \tau_j \in \{T \cup \tau_i\}, \beta > D_j$, we do not accept τ_i.

These hypothesis could be used according to the nature of *execution times*; if execution times are *controllable*, that is we can influence the time spent in the execution, we could use minimum execution times. This could be acceptable for applications where execution times are bound to some quality of service, for instance performing an approximative calculus instead of an exact one or composing an image in different qualities.

On the contrary, if execution times are *uncontrollable*, then we need maximum execution times, since we must accept to work under a worst case perspective.

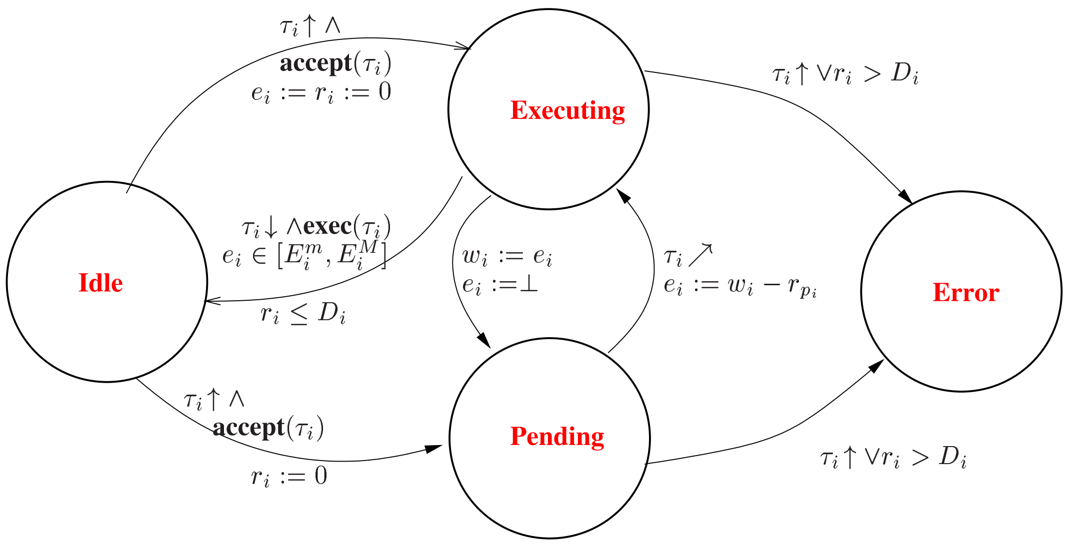

Figure 9: Automaton for a General Scheduler

5 General schedulers

In this section we consider general scheduling policies, that is, preemptive schedulers based on some priority assignment mechanism which can be fixed or dynamic. We will relax the constraint of one-preemption imposed to LIFO and EDF schedulers and we consider uncertain, but bounded, execution times.

Instead of using a stopwatch automaton as we have done in the previous sections, we use a model based on timed automata as shown in figure, 9 where to each task τ_i we add a clock w_i which initially counts the accumulated computed time for a task. The main idea is to replace a stopped clock by an operation of difference of two running clocks, to keep track of already executed time.

Clocks w's are used as follows: preemption is only possible at arrival of a new task, say τ_j and each time a task $\tau_j \dagger \tau_i$, e_i is accumulated in w_i, τ_j gains processor and when τ_i is resumed, we recover e_i as the difference $w_i - r_j$; clock e_i is then never stopped but updated. This procedure relaxes the one preemption hypothesis but still obliges to keep clock r_j even if task τ_j has finished its execution and hence it is not active.

Example .5 *Let us consider* $\mathbb{T} = \{\tau_1(4, 13), \tau_2(5, 10), \tau_3(2, 10), \tau_4(3, 6)\}$*; we show how the introduction of w's clocks can help to relax the constraint of one-preemption under an* EDF *scheduler, see figure 10.*

- At time $t = 0$, $\tau_1\uparrow$, $r_1 = e_1 := 0$
- At time $t = 2$, $\tau_2\uparrow$ and its deadline 10 is shorter than τ_1's (11), so, τ_1 is preempted and joins the queue; $w_1 := e_1 = 2$. From there on the value of $e_1 = w_1 - r_2$.
- At time $t = 4$, $\tau_3\uparrow$ and its deadline 10 is longer than τ_2's, so it joins the queue (after τ_1).
- At time $t = 5$, $\tau_4\uparrow$, its deadline 6 is shorter than τ_2's, which is preempted and we set $w_2 := e_2 = 3$; from there on $e_2 = w_2 - r_4$.
- At time $t = 8$, $\tau_4\downarrow$ and e_2 is updated as $w_2 - r_4 = 6 - 3$; taui2 resumes execution.
- The rest of the tasks proceed in a similar manner. Note that at time $t = 12$, τ_3 gains the processor for the first time, e_3 and w_3 are indefined.

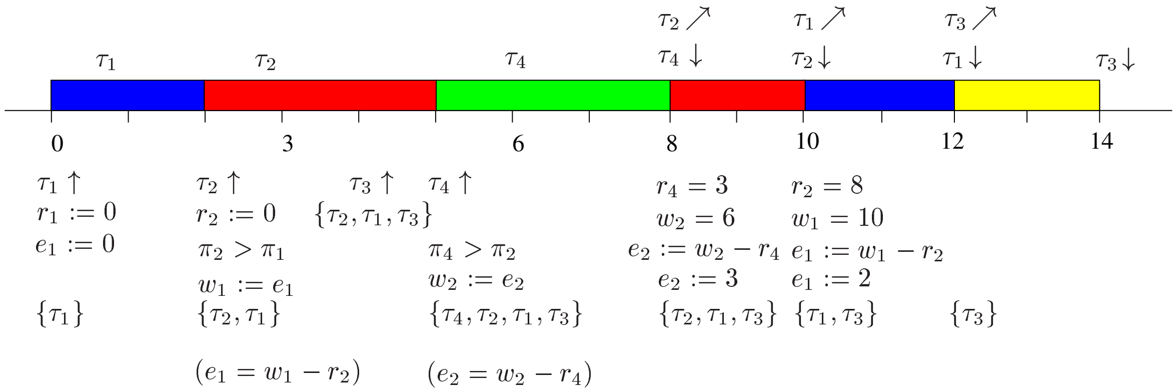

Figure 10: General EDF Scheduler

5.1 Transition Model

Formally the transition system is of the form $(\mathcal{S}, Q, \rightarrow)$ composed of discrete events and time passing transitions, as already mention in the precedent sections.

- $\mathcal{S} = (S, \vec{e}, \vec{r}, \vec{p}, \vec{w})$, where S, $\vec{e}$, $\vec{r}$ and $\vec{p}$ have the same meaning as in EDF and $\vec{w}$ is the auxiliary vector to reconstruct the execution times after preemption,
- Q is a queue of tasks, with the usual operations: $\oplus$, for adding an element, **pop** to remove the element at the head, **top**, to choose the task at the head.
- $\rightarrow$ is the transition relation.

We list the operations over our transition system; recall that the arrival of a task is captured by the scheduler, who decides over the admission. We assume also that the schedulers 'knows' the priority of each task (dynamic or fixed); of course, the currently executing task, denoted τ_c, is on the head of the queue and has the highest priority; priority of task τ_i is noted π_i, as usual; the operation $\oplus$ works on a queue according to a scheduling policy.

- Task arrival, $\tau_i\!\uparrow$ $(\tau_i \in \mathbb{T})$:

$$(\mathcal{S}, Q) \xrightarrow{\tau_i\uparrow} (S', Q')$$

where

$Q' = Q \oplus \tau_i$.

$$\vec{e'}_j = \begin{cases} \vec{e}_j & \text{if } j \neq i \\ 0 & \text{if } j = i \wedge \pi_i > \pi_c \ (\mathbf{exec}(\tau_i)) \\ \perp & \text{otherwise} \end{cases}$$

$$\vec{r'}_j = \begin{cases} 0 & \text{if } j = i \\ \vec{r}_j & \text{otherwise} \end{cases}$$

$$\vec{p'}_j = \begin{cases} i & \text{if } j = c \wedge \pi_i > \pi_c \ (\tau_i \dagger \tau_c) \\ 0 & \text{if } j = i \text{ (no task preempted } \tau_i) \\ \vec{p}_j & \text{otherwise} \end{cases}$$

$$\vec{w'}_j = \begin{cases} \vec{e}_j & \text{if } j = c \wedge \pi_i > \pi_c \ (\tau_i \dagger \tau_c) \\ \perp & \text{if } j = i \\ \vec{w}_j & \text{otherwise} \end{cases}$$

Note that $w_c := e_c$ if τ_c is preempted by τ_i and $w_i = \perp$.

- Task completion: $\tau_c \downarrow$

$$(\mathcal{S}, Q) \stackrel{\tau_c\downarrow}{\rightarrow} (S', Q')$$

where $Q' = \textbf{pop}(Q)$ and

$$\vec{e'}_j = \begin{cases} \perp & \text{if } j = c \\ \vec{e}_j & \text{otherwise} \end{cases}$$

$$\vec{w'}_j = \begin{cases} \perp & \text{if } j = c \\ \vec{w}_j & \text{otherwise} \end{cases}$$

$$\vec{r}_j = \begin{cases} \perp & \text{if } j = c \wedge \nexists p_{\mu(T_k)} = c, \ 1 \leq k < n \\ \vec{r}_j & \text{otherwise} \end{cases}$$

Variable $\vec{p}$ remains unchanged.

- Task resumption: $\tau_i \nearrow$ (we assume $\tau_i = \textbf{top}(Q)$ is a task which regains the processor).

$$(S, Q) \stackrel{\tau_i\nearrow}{\rightarrow} (S', Q)$$

where

$$\vec{e'}_j = \begin{cases} \vec{w}_j - \vec{r}_{\vec{p}_j} & \text{if } j = i \wedge \vec{p}_j \neq 0 \ (\tau_{\vec{p}_j} \dagger \tau_i) \\ 0 & \text{if } j = i \wedge \vec{p}_j = 0 \ (\tau_i \text{ was never preempted}) \\ \vec{e}_j & \text{otherwise} \end{cases}$$

Variables $\vec{r}$, $\vec{p}$ and $\vec{w}$ remain unchanged.

- Time passing: δ is an elapsed time not enough to finish the current executing task, τ_c.

$$(S, Q, \vec{e}, \vec{r}, \vec{p}, \vec{w}) \stackrel{\delta}{\rightarrow} (S', Q, \vec{e'}, \vec{r'}, \vec{p}, \vec{w'})$$

where

$$\vec{e'}_j = \begin{cases} \vec{e}_j + \delta & \text{if } j = c \\ \vec{e}_j & \text{otherwise} \end{cases}$$

and $\vec{r'}_j = \vec{r}_j + \delta$ and $\vec{w'}_j = \vec{w}_j + \delta \ \forall \tau_j \in Q$

Remark I Note that clock w_i is initially set to bottom at τ_i arrival, and it is updated to e_i if this task is preempted, so saving the cumulated executing time; from there on w_i grows (while e_i is $\perp$) and when τ_i regains processor its cumulated time is recovered from the difference between w_i and the released clock of the preempter (kept in $\vec{p}_i$). This implies that released clocks cannot disappear until the preempted task regains the processor. This condition must be tested at admission time of a new task. Figure 11 shows the evolution of clocks.

A possible more elegant way of solving the problem consists in systematically adding the new variable h_i for each task, and use it in the time-invariant equations of the form $e_c = w_c - h_i$. In this case, the r variables are eliminated at completion time but many 'instances of h' may be necessary to be created as τ_i may be a very eager task with high priority preempting different tasks at each arrival. This approach unnecessary complicates the proofs (as it requires carrying through additional invariants). Besides, it is not very useful in practice as it augments the complexity by increasing the number of clocks.

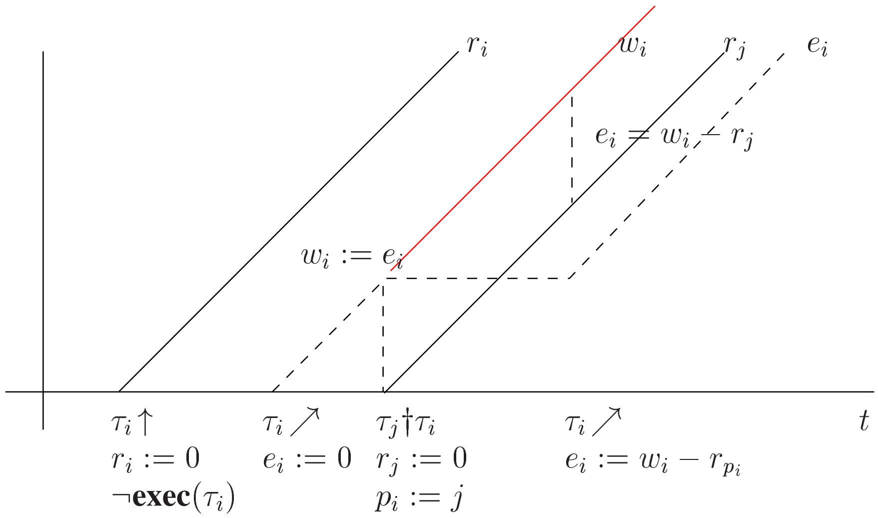

Figure 11: Evolution of $\vec{w}$ and $\vec{e}$

Remark II We will show that the fact of simulating a stopped clock e_i by a difference constraint of the form $w_i - r_{p_i}$, both running does not disturb the semantics of the systems; indeed we will prove that the relationships where e_i is involved can be replaced by this expression while e_i is stopped and still the problem of schedulability, view as the problem of reachability of an error state is decidable.

5.2 Properties of a General Scheduler

Let Q be the queue of active tasks, $Q \in \mathbb{T}$ where $\tau_c = \textbf{top}(Q)$. We can enumerate the following properties:

1. if $p_j = 0 \Rightarrow e_j = w_j, \forall \tau_j, \tau_j \in Q$
2. if $\exists\, p_j = c \Rightarrow e_j = w_j - r_c$
3. if $\tau_c \nearrow \wedge p_c = 0 \Rightarrow e_c = w_c = \perp \wedge \neg\exists p_j = c$, for any $\tau_j \in Q$
4. $\forall \tau_j \in \{Q - \textbf{top}(Q)\} \wedge p_j \neq 0 \Rightarrow e_j = w_j - r_{p_j}$

Properties 1, 2 and 3 are completely analogous to the corresponding EDF scheduler properties. The last property 4 can be reformulated to create our **general execution invariant**:

$$I_{\text{SCH}}(Q) \equiv \bigwedge_{\tau_j \in Q', p_j \neq 0} e_j = w_j - r_{p_j} \wedge \bigwedge_{\tau_j \in Q', p_j = 0} e_j = \perp$$

where $Q' = \textbf{pop}(Q)$.

The invariant says that for those tasks waiting for execution and preempted their cumulated executed time can be express as a difference of clocks; evidently, for those tasks never executed at all their cumulated executed time is unknown.

5.3 Schedulability Analysis

Let Ψ as explained in LIFO analysis and let ϕ be a constraint characterizing a set of states. We define $\tau_i\uparrow(\phi)$ to be the set of states reached when task τ_i arrives, that is:

$$\tau_i\uparrow(\phi) = \{s' : \exists s \in \phi.\ s \overset{\tau_i\uparrow}{\to} s'\}$$

Let ϕ be of the form $I_{\text{SCH}}(Q) \wedge \psi$, with $\psi \in \Psi$; that is ϕ characterizes a state with the execution invariant for all waiting tasks and clock relationships expressed as differences.

Without loss of generality, we can assume that either:

1. τ_i is rejected: $\psi \implies \neg\textbf{accept}(\psi, Q, \tau_i)$, in which case we have $\tau_i\uparrow(\phi) \equiv \phi$.
2. τ_i is accepted: $\psi \implies \textbf{accept}(\psi, Q, \tau_i)$. Does $\tau_i \dagger \tau_c$?

$$\text{if } \neg\tau_i\dagger\tau_c, \text{ then } \tau_i\uparrow(\phi) \equiv I_{\text{SCH}}(Q \oplus \tau_i) \wedge r_i = 0 \wedge e_i =\perp \wedge w_i =\perp \wedge\psi \equiv I_{\text{SCH}}(Q') \wedge \psi$$

$$\text{if } \tau_i\dagger\tau_c, \text{ then } \tau_i\uparrow(\phi) \equiv I_{\text{SCH}}(Q \oplus \tau_i) \wedge r_i = 0 \wedge e_i = 0 \wedge \psi[e_c := w_c] \equiv I_{\text{SCH}}(Q') \wedge \psi'$$

where $\psi[e_c := w_c]$ is the substitution of e_c for w_c in ψ. In summary:

$$\tau_i\uparrow(\phi) \equiv I_{\text{SCH}}(Q') \wedge \psi'$$

Hence, we have:

Proposition .5 *$\tau_i\uparrow(\phi)$ has the same structure than ϕ, that is, it is the conjunction of an execution invariant and a formula in Ψ.*

Now, let $\phi\nearrow$ be the set of states reached from ϕ by letting time advance, that is:

$$\phi\nearrow = \{s' : \exists s \in \phi, \delta \geq 0 . s \xrightarrow{\delta} s'\}$$

Clearly, if ϕ is of the form $I_{\text{SCH}}(Q) \wedge \psi$, we have that

$$\phi\nearrow = (I_{\text{SCH}}(Q) \wedge \psi)\nearrow \equiv I_{\text{SCH}}(Q) \wedge \psi\nearrow$$

As $\psi \in \Psi$ over clocks in $\mathcal{S}$, we can express these differences in a DBM. The following proposition gives this equivalence:

Proposition .6 *Let ϕ be of the form $I \wedge M$, where M is a DBM and I is an execution invariant under SCH. Then, $\tau_i\uparrow(\phi)$ and $\phi\nearrow$ have the same structure as ϕ.*

Thus, given a sequence σ of task arrivals the set of reached states $\sigma(\phi)$ can be represented by the conjunction of the execution invariant $I_{\text{SCH}}(Q)$, characterizing the already executed time of the suspended tasks and a DBM M, characterizing the relationships between the corresponding r's and w's clocks.

A DBM M has the following form (since $e_n = r_n$ we omit it; in order to facilitate comprehension, we "name" clocks according to the position of their corresponding tasks in Q):

M	u	r_1	w_1	r_2	w_2	$\ldots$	r_n	w_n
u	$-$	M_{ur_1}	M_{uw_1}	M_{ur_2}	M_{uw_2}	$\ldots$	M_{ur_n}	M_{uw_n}
r_1	M_{r_1u}	$-$	$-$	$M_{r_1r_2}$	$M_{r_1w_2}$	$\ldots$	$M_{r_1r_n}$	$M_{r_1w_n}$
w_1	M_{w_1u}	$-$	$-$	$M_{w_1r_2}$	$M_{w_1w_2}$	$\ldots$	$M_{w_1r_n}$	$M_{w_1w_n}$
r_2	M_{r_2u}	$M_{r_2r_1}$	$M_{r_2w_1}$	$-$	$-$	$\ldots$	$M_{r_2r_n}$	$M_{r_2w_n}$
w_2	M_{w_2u}	$M_{w_2r_1}$	$M_{w_2w_1}$	$-$	$-$	$\ldots$	$M_{w_2r_n}$	$M_{w_2w_n}$
$\vdots$								
r_n	M_{r_nu}	$M_{r_nr_1}$	$M_{r_nw_1}$	$M_{r_nr_2}$	$M_{r_nw_2}$	$\ldots$	$-$	$-$
w_n	M_{w_nu}	$M_{w_nr_1}$	$M_{w_nw_1}$	$M_{w_nr_2}$	$M_{w_nw_2}$	$\ldots$	$-$	$-$

Once again, M is constructed in a very special way and has a particular structure. Let us analyse it:

- The new matrix M is constructed as new tasks τ_i's arrive; we denote $M' = M^{\tau_i\uparrow}$ the values in M immediately after acceptance of τ_i.
- When $\tau_i\uparrow$, we have two possible situations (assuming it is accepted):
  - $\tau_i \dagger \tau_c$, then $r_i = e_i := 0$, we need to stop e_c and create w_c with value e_c, we have that $e_c = w_c - r_i \leq M'_{w_c r_i} = M_{e_c r_i}$, but $r_i = 0$ and so $M'_{w_c r_i} = M_{e_c r_i} = M_{e_c u} = M'_{w_c u}$.
  - $\neg(\tau_i \dagger \tau_c)$, then $r_i := 0, e_i := \perp$, we have $r_c - r_i \leq M'_{r_c r_i}$, $r_i = 0$ and so we have $M'_{r_i r_c} = M_{r_c u}$. This relation is also respected in the precedent case.
- When $\tau_c\downarrow$, we have again two situations:
  - $\nexists p_j = c,\ \tau_j \in Q$:
$$(I_{\text{SCH}}(Q) \wedge \psi)[w_c := \perp, r_c := \perp] \equiv I_{\text{SCH}}(\mathbf{pop}(Q)) \wedge \psi'$$
  - $\exists p_j = c,\ \tau_j \in Q$:
$$(I_{\text{SCH}}(Q) \wedge \psi)[w_c := \perp]$$
- When $\tau_i \nearrow$, once again two situations are possible:
  - $p_i = j$,:
$$(I_{\text{SCH}}(Q) \wedge \psi)[r_j := w_i - e_i] \equiv I_{\text{SCH}}(\mathbf{pop}(Q)) \wedge \psi'$$
  - $p_i = 0$:
$$I_{\text{SCH}}(Q) \wedge \psi \wedge e_i := 0$$

So, the characterization of each state as time passes or new tasks arrive or resume is preserved as differences of running clocks. At each operation, the representation under a DBM keeps the structure of bounded differences

Now, *what is the structure of M after a task completion?* Is it still a DBM?

We will prove that this operation still enables us to characterize the states as bounded differences in a DBM, so establishing that $\tau_i\downarrow(\phi)$ is still the conjunction of an invariant and a formula in Ψ.

Let us expose the scenario when a task τ_i finishes. At that moment, the scheduler will choose another task, say τ to regain the processor; this task had been eventually preempted in the past by another task, say $\overline{r}$ and the relation $e := w - \overline{r}$ shows the computed time for τ. Clock $\overline{r}$ can now be eliminated from M and replaced by $w - e$. What happens to all differences in M where $\bar{r}$ is named?

We have the following relations involving $\bar{r}$:

1. Base relations:

$$w - \overline{r} \leq M_{w\overline{r}} \quad \Rightarrow \quad e - u \leq M_{w\overline{r}} \tag{32}$$
$$\overline{r} - w \leq M_{\overline{r}w} \quad \Rightarrow \quad u - e \leq M_{\overline{r}w} \tag{33}$$
$$u - \overline{r} \leq M_{u\overline{r}} \quad \Rightarrow \quad e - w \leq M_{u\overline{r}}$$
$$\overline{r} - u \leq M_{\overline{r}u} \quad \Rightarrow \quad w - e \leq M_{\overline{r}u}$$

2. Let x be another clock different from w and u:

$$x - \overline{r} \leq M_{x\overline{r}} \quad \Rightarrow \quad x - (w - e) \leq M_{x\overline{r}} \tag{34}$$
$$\overline{r} - x \leq M_{\overline{r}x} \quad \Rightarrow \quad (w - e) - x \leq M_{\overline{r}x} \tag{35}$$

We can consider that these differences can be decomposed in the following ways:

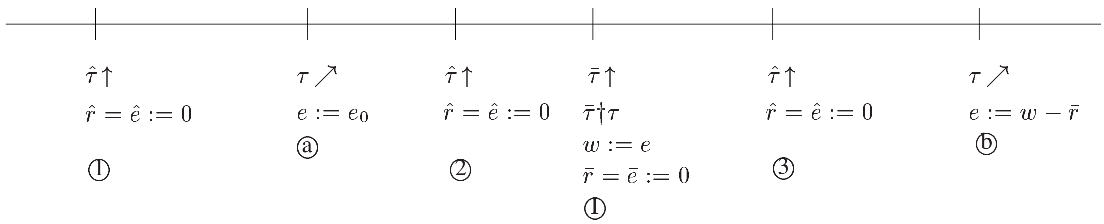

Figure 12: Analysis of DBM M

1. In 34 and considering 32:

$$\left.\begin{array}{lcl} x - w & \leq & M_{xw} \\ e - u & \leq & M_{w\bar{r}} \end{array}\right\} \Rightarrow x - (w - e) \leq M_{xw} + M_{w\bar{r}}$$

$$M_{xw} + M_{w\bar{r}} \quad \theta \quad M_{x\bar{r}} \tag{36}$$

2. In 35 and considering 33

$$\left.\begin{array}{lcl} w - x & \leq & M_{wx} \\ u - e & \leq & M_{\bar{r}w} \end{array}\right\} \Rightarrow (w - e) - x \leq M_{\bar{r}w} + M_{wx}$$

$$M_{\bar{r}w} + M_{wx} \quad \theta \quad M_{\bar{r}x} \tag{37}$$

Both expressions are not difference constraints but we will show that in 36 and 37 θ represents equality; that is, we will prove that:

$$M_{xw} + M_{w\bar{r}} = M_{x\bar{r}} \tag{38}$$
$$M_{\bar{r}w} + M_{wx} = M_{\bar{r}x} \tag{39}$$

and hence they are deductible from M, no need to keep them in the $M'^{\tau_i\downarrow}$.

To prove this we will consider a task τ which regains the processor after being interrupted by another task $\bar{\tau}$, point ① in figure 12; a third task $\hat{\tau}$ will be used to express the evolution of differences as τ is executing or waiting. In the figure we show the three different possibilities of arrival for such a task $\hat{\tau}$ in the system, namely ①,②,③; $\tau \nearrow$ at point ⓐ indicates the last execution for τ when it was preempted by $\bar{\tau}$; its cumulated executed time is e_0. We are analysing clock relationships when τ prepares to resume its execution (point ⓑ in figure 12), after it was preempted by $\bar{\tau}$

It is of extreme importance to remark two properties concerning our scenario:

- **Monotony**: clocks grow at the same rate; in our model the derivative of a clock is always 1 (if it is running) or 0 (if it is stopped).
- **Continuity**: From point ⓐ to point ⓑ clocks for τ were not reset neither updated. They were not reset, because any new instance of τ should have been rejected by the scheduler, since a previous instance is still active (and reset is only applied at task arrival). On the other hand, clocks were not updated, because the only possibility is to update e by the operation $e := w - \bar{r}$ or w by the operation $w := e$, but we are supposing that in between no resuming of τ occurs; in fact point ⓑ is the first execution after the last preemption, point ①, so no such update operation is possible.

In interval [ⓐ, ⓑ], clocks r and w are running monotonously and continuously while clock e is stopped. This means that differences such as $r - x$ and $w - x$ where x is also running, do not invalidate the respective bounds M_{rx} and M_{wx}; also, x cannot be a stopped clock, since if it were, it would be an execution clock e' of a preempted task τ' and in that case, we should have replaced it by its appropriate difference involving two continous clocks. In point ⓑ clock $\bar{r}$ can be eliminated and replaced by $w - e$ in M which leaves us with three term differences such as 34: we will prove that these differences can be deduced by simple bounded differences.

We know that if $\bar{\tau}\dagger\tau$ then it must be $\pi_{\bar{\tau}} > \pi_{\tau}$. At preemption time, that is when $\bar{\tau}\uparrow$, we set $w := e$, $\bar{r} = \bar{e} := 0$ and $\bar{w} := \perp$.

We note that arrival times are indeed intervals, since our execution times are unknown but bounded; remember that values in M are characterized by a super-index indicating its value at a certain moment; for instance $M^{\bar{\tau}\uparrow}_{eu}$ means "the maximum value for e at arrival of $\bar{\tau}$".

We will prove equality for expression 38 but it is absolutely simetric for 39.

5.3.1 Case ①

We distinguish two cases according to priority relationships; either $\pi_{\hat{\tau}} > \pi_{\tau}$ or $\pi_{\hat{\tau}} < \pi_{\tau}$

1. $\pi_{\hat{\tau}} > \pi_{\tau}$, this means that at $\tau\nearrow$, task $\hat{\tau}$ did finish its execution but there may be anohter task $\tilde{\tau}$ preempted by $\hat{\tau}$ still active, with priority $\pi_{\tilde{\tau}} < \pi_{\tau}$; under this scenario clock $\hat{r}$ is still running, but clock $\hat{w}$ has disappeared.

   Figure 13 shows the situation graphically.

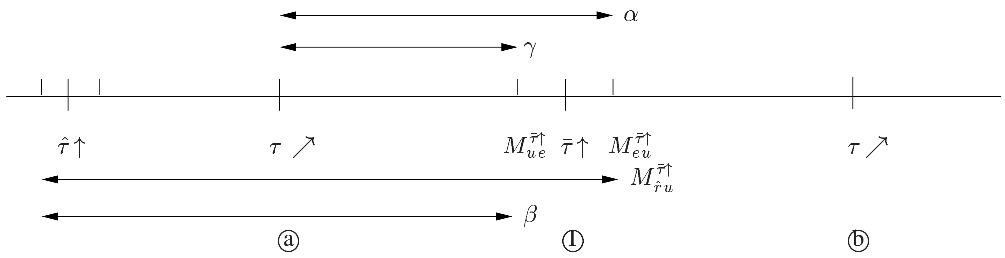

Figure 13: Case ① $\pi_{\hat{\tau}} > \pi_{\tau}$

$$M^{\tau\nearrow}_{\hat{r}w} + M^{\tau\nearrow}_{w\bar{r}}\theta M^{\tau\nearrow}_{\hat{r}\bar{r}}$$

- $M^{\tau\nearrow}_{\hat{r}w} = M^{\bar{\tau}\uparrow}_{\hat{r}e} = M^{\tau\nearrow}_{\hat{r}e}$
  This sucession of equalities is based on our properties of monotony and continuity; in fact, the difference $\hat{r} - w$ at $\tau\nearrow$ (point ⓑ) is the same since w was created, that is in point ① when $\bar{\tau}\uparrow$, which equals the value e; this difference is constant as both e and $\hat{r}$ were running (point ⓐ). The same reasoning as a chain of equalities is kept all over the proof.
- $M^{\tau\nearrow}_{w\bar{r}} = M^{\bar{\tau}\uparrow}_{eu} = \alpha + e_0$
- $M^{\tau\nearrow}_{\hat{r}\bar{r}} = M^{\bar{\tau}\uparrow}_{\hat{r}u}$
- In figure 13 we have:

  $$M^{\bar{\tau}\uparrow}_{ue} = \gamma + e_0$$

  and

  $$M^{\bar{\tau}\uparrow}_{\hat{r}u} - \beta = \alpha - \gamma \to M^{\bar{\tau}\uparrow}_{\hat{r}u} = \alpha - \gamma + \beta$$

  adding e_0 gives

  $$M^{\bar{\tau}\uparrow}_{\hat{r}u} = \underbrace{(\alpha + e_0)}_{M^{\tau\nearrow}_{w\bar{r}}}\underbrace{-(\gamma + e_0) + \beta}_{M_{\hat{r}w}} \to$$

$$M^{\tau\nearrow}_{\hat{r}w} + M^{\tau\nearrow}_{w\bar{r}} = M^{\tau\nearrow}_{\hat{r}\bar{r}}$$

proving that in fact the relationship θ is equality.

2. $\pi_{\hat{\tau}} < \pi_\tau$, this means that at $\tau\nearrow$, task $\hat{\tau}$ did not finish its execution and hence clocks $\hat{r}$ and $\hat{w}$ are both active. The analysis for $\hat{r}$ is the same as above; let us see what happens to $\hat{w}$. Figure 14 shows the situation graphically.

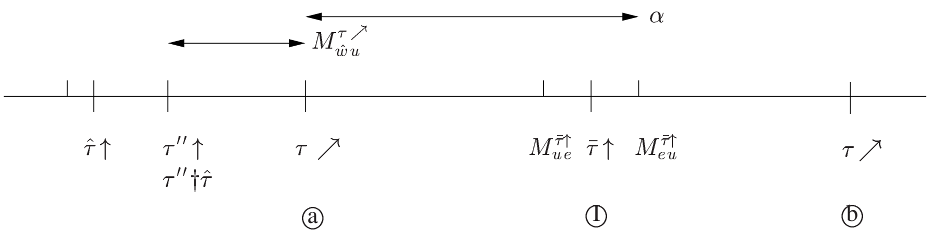

Figure 14: Case 1 $\pi_{\hat{\tau}} < \pi_\tau$

$$M^{\tau\nearrow}_{\hat{w}w} + M^{\tau\nearrow}_{w\bar{r}} \theta M^{\tau\nearrow}_{\hat{w}\bar{r}}$$

In figure 14 we have:

- $M^{\tau\nearrow}_{\hat{w}w} = M^{\bar{\tau}\uparrow}_{\hat{w}e} = M^{\tau\nearrow}_{\hat{w}e} = M^{\tau\nearrow}_{\hat{w}u} - e_0$
  The difference between $\hat{w}$ and w at the moment of resuming τ (point a) is the same as the difference when w was created, point 1, that is at arrival of $\bar{\tau}$; by the property of monotony this difference is kept since both e and $\hat{w}$ were running, that is $\tau\nearrow$ at point a. Finally, by monotony this value is the same as the difference between the initial value for $\hat{w}$, that is $M^{\tau\nearrow}_{\hat{w}u}$ and e_0.
- $M^{\tau\nearrow}_{w\bar{r}} = M^{\bar{\tau}\uparrow}_{eu} = \alpha + e_0$ and
- $M^{\tau\nearrow}_{\hat{w}\bar{r}} = M^{\bar{\tau}\uparrow}_{\hat{w}u} = M^{\tau\nearrow}_{\hat{w}u} + \alpha$ adding e_0 gives:

$$\underbrace{M^{\tau\nearrow}_{\hat{w}u} + \alpha}_{M^{\tau\nearrow}_{\hat{w}\bar{r}}} = \underbrace{M^{\tau\nearrow}_{\hat{w}u} - e_0}_{M^{\tau\nearrow}_{\hat{w}w}} + \underbrace{\alpha + e_0}_{M^{\tau\nearrow}_{w\bar{r}}}$$

  hence proving

$$M^{\tau\nearrow}_{\hat{w}w} + M^{\tau\nearrow}_{w\bar{r}} = M^{\tau\nearrow}_{\hat{w}\bar{r}}$$

5.3.2 Case 2

Recall figure 12; we have also two possibilities for $\hat{\tau}$

1. $\pi_{\hat{\tau}} > \pi_\tau$, this situation is not possible under our scenario since we are considering τ preempted by $\bar{\tau}$ during its last execution.

2. $\pi_{\hat{\tau}} < \pi_\tau$, then $\hat{\tau}$ did not execute at all: its priority being smaller, it must wait at least for τ to finish; only $\hat{r}$ is running. Figure 15 shows this situation graphically;

$$M^{\tau\nearrow}_{\hat{r}w} + M^{\tau\nearrow}_{w\bar{r}} \theta M^{\tau\nearrow}_{\hat{r}\bar{r}}$$

- $M^{\tau\nearrow}_{\hat{r}w} = M^{\bar{\tau}\uparrow}_{\hat{r}w} = M^{\bar{\tau}\uparrow}_{ue} = -(\beta + e_0)$
- $M^{\tau\nearrow}_{w\bar{r}} = M^{\bar{\tau}\uparrow}_{eu} = \alpha + e_0$

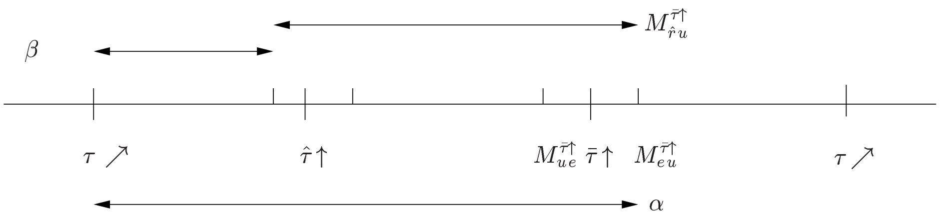

Figure 15: Case ② $\pi_{\hat{\tau}} < \pi_{\tau}$

- $M^{\tau\nearrow}_{\hat{r}\bar{r}} = M^{\bar{\tau}\uparrow}_{\hat{r}u}$
- In figure 15 we have:

$$\alpha - \beta = M^{\bar{\tau}\uparrow}_{\hat{r}u}$$

$$\underbrace{(\alpha + e_0)}_{M^{\tau\nearrow}_{w\bar{r}}} - \underbrace{(\beta + e_0)}_{M^{\tau\nearrow}_{\hat{r}w}} = M^{\bar{\tau}\nearrow}_{\hat{r}u}$$

hence proving

$$M^{\tau\nearrow}_{\hat{r}w} + M^{\tau\nearrow}_{w\bar{r}} = M^{\tau\nearrow}_{\hat{r}\bar{r}}$$

5.3.3 Case ③

Once again, we have two possibilities for $\hat{\tau}$

1. $\pi_{\hat{\tau}} > \pi_{\tau}$, in this case $\hat{\tau}$ did finish at $\tau^{\nearrow}$ and if $\hat{\tau}$ preempted a task, it should be one with higher priority than τ, so both tasks have finished by the moment $\tau^{\nearrow}$, (point ⓑ) and no clock $\hat{r}$ exists.
2. $\pi_{\hat{\tau}} < \pi_{\tau}$, then $\hat{\tau}$ did not execute at all, only clock $\hat{r}$ is running. Figure 16 shows the scenario.

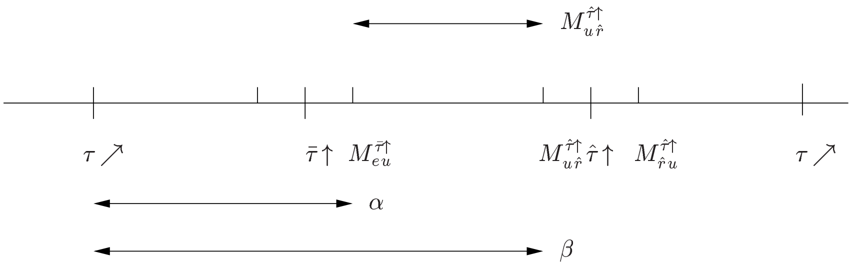

Figure 16: Case ③ $\pi_{\hat{\tau}} < \pi_{\tau}$

$$M^{\tau\nearrow}_{\hat{r}w} + M^{\tau\nearrow}_{w\bar{r}} \theta M^{\tau\nearrow}_{\hat{r}\bar{r}}$$

- $M^{\tau\nearrow}_{\hat{r}w} = M^{\hat{\tau}\uparrow}_{\hat{r}w} = M^{\hat{\tau}\uparrow}_{uw} = -(\beta + e_0)$
- $M^{\tau\nearrow}_{w\bar{r}} = M^{\hat{\tau}\uparrow}_{w\bar{r}} = M^{\bar{\tau}\uparrow}_{eu} = \alpha + e_0$
- $M^{\tau\nearrow}_{\hat{r}\bar{r}} = M^{\hat{\tau}\uparrow}_{u\bar{r}}$
- In figure 16 we have:

$$\alpha - \beta = M^{\hat{\tau}\uparrow}_{u\hat{r}}$$

$$\underbrace{(\alpha + e_0)}_{M^{\tau\nearrow}_{w\bar{r}}} \underbrace{-(\beta + e_0)}_{M^{\tau\nearrow}_{\hat{r}w}} = M_{u\hat{r}^{\hat{\tau}\uparrow}}$$

hence proving

$$M^{\tau\nearrow}_{w\bar{r}} + M^{\tau\nearrow}_{\hat{r}w} = M^{\tau\nearrow}_{\hat{r}\bar{r}}$$

Thus, we have proved that the reachability problem in our transition system $(\mathcal{S}, Q, \rightarrow)$ using a decrementation of the form $e = w - \bar{r}$ for preempted tasks, is solvable. Relationships among running clocks can be encoded using a DBM; we have proved that relationships involving stopped clocks when replaced by their differences do not give a difference constraint, but these difference constraints can be deduced from other difference constraints in M, thus they can be eliminated. The following theorem resumes our theory:

Theorem .3 *Given a task model as defined in 5.1 and a general scheduling policy, the reachability graph of the system can be symbolically characterized using predicates of the form $I \wedge M$ where I is a conjunction of equalities $e = w - \bar{r}$ and M is a* DBM*. Moroever, the reachability graph is finite.*

As a corollary: **the schedulability problem for this class of systems is decidable**.

5.4 Properties of the Model

We have shown that the relationships among release clocks for "free" tasks, that is tasks which have not preempted each other, can be implied by the sum of relationships between a preempted task, its already computed time and the corresponding release clocks. This is a very useful property because it reduces the amount of relationships in M. In fact, all those differences envolving released clocks of free tasks can be deduced from a base set of bounds, involving only released clocks from free tasks and w clock from a preempted task and hence matrix representation is also reduced.

The properties of continuity and monotony are exploted for our reachability analysis, implying that it is possible to construct the reachability graph.

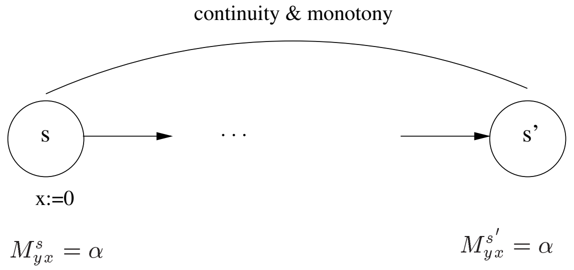

Figure 17: Nicety property

In general if we consider two states s and s' in a timed automaton, see figure 17, where clock x is reset in s and no reset or update operations are done in between, we can see that the difference is kept; this phenomena is due to the fact that both clocks show a monotonously increasing property (time passing) and also a continuity property (no update is done). Under this context, another clock z (not necessarily continuous) shows the property: $M_{zy} = M_{zx} + M_{xy}$ and hence no need to keep this difference, (intuitively it is as if the stopped time for z were "absorbed" by x and y, both running).

6 Final Recipe!

Now that we know the problem can be modelled as the transition system defined in section 2, we can sketch an operational approach of our system.

1. Given a RT problem, we can partition it into tasks characterized by timing constraints. If the problem is expressed in Java, we can use techniques such as [5, 8] to cut up the application into smaller tasks.
2. Each task is assigned a fixed or a dynamic priority which is used by the scheduler; naturally, we impose that at arrival of a task, a priority is known.
3. The scheduler keeps a queue Q of tasks, preempted or not, ordered by priority, being the task with highest priority on the head of Q.
4. As a new task τ_i arrives, an admittance test is performed to analyse if its execution leaves the system in a *safe* state, that is, a state where all tasks in Q, including τ_i, finish their executions before their respective deadlines and that no information of preemption is lost. We have given an admittance test for EDF. +
5. If τ_i is accepted:
   - it preempts the currently executing task τ_c if $\pi_i > \pi_c$; we update the information of preemption marking that $p_c := i$ and also setting clock $w_c := e_c$; τ_i joins the queue Q as the new head and τ_c is behind it.
   - it does not preempt τ_c if $\pi_i \leq \pi_c$; in this case, τ_i joins the queue Q somewhere according to its priority.
6. When τ_c finishes, the scheduler can eliminate it from Q but its release clock is kept if $\exists p_j = c$ for some task $\tau_j \in Q$; otherwise all clocks can be eliminated.
7. When a task τ_i resumes execution, its already executed time can be recovered from the difference $e_i := w_i - r_{p_i}$ if $p_i \neq 0$; otherwise $e_i := 0$.

References

[1] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Are timed automata updatable? In *Proceedings of the 12th Int. Conf. on Computer Aided Verification*, pages 464–479, Chicago, USA, July 2000. 1

[2] Franck Cassez and Kim Larsen. The impressive power of stopwatches. *Lecture Notes in Computer Science*, 1877:138+, 2000. 1

[3] Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Schedulability anaysis using two clocks. In *ETAPS 2003*, 2003. 1

[4] Elena Fersman, Paul Pettersson, and Wang Yi. Timed automata with asynchronous processes: Schedulability and decidability. In *ETAPS 2002*, 2002. 1

[5] D. Garbervetsky, C. Nakhli, S. Yovine, and H. Zorgati. Program instrumentation and run-time analysis of scoped memory in java. In *Proceeding of International Workshop on Runtime Verification. ETAPS 2004*, Barcelona. Spain, April 2004. 1

[6] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 373–382, 1995. 1

[7] Y. Kesten, A. Pnueli, J. Sifakis, and S.Yovine. Integration graphs: A class of decidable hybrid systems. *LNCS. Sepecial Edition on Hybrid Systems*, 736:179–208, 1993. 1

[8] Christos Kloukinas, Chaker Nakhli, and Sergio Yovine. A methodology and tool support for generating scheduled native code for real-time java applications. In *Proceedings of the Third International Conference on Embedded Software (EMSOFT 2003)*, pages 274–289. Lecture Notes in Computer Science-2855, Springer Verlag, october 2003. 1

[9] C.L. Liu and James Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20:46–61, January 1973. 4

[10] Jennifer McManis and Pravin Varaiya. Suspension automata: a decidable class of hybrid automata. In *Proc.6th International Conference on Computer Aided Verification, CAV'94, Stanford, California, USA*, volume 818, pages 105–117. Springer-Verlag, 1994. 1