Algorithmique I

Aline Goeminne

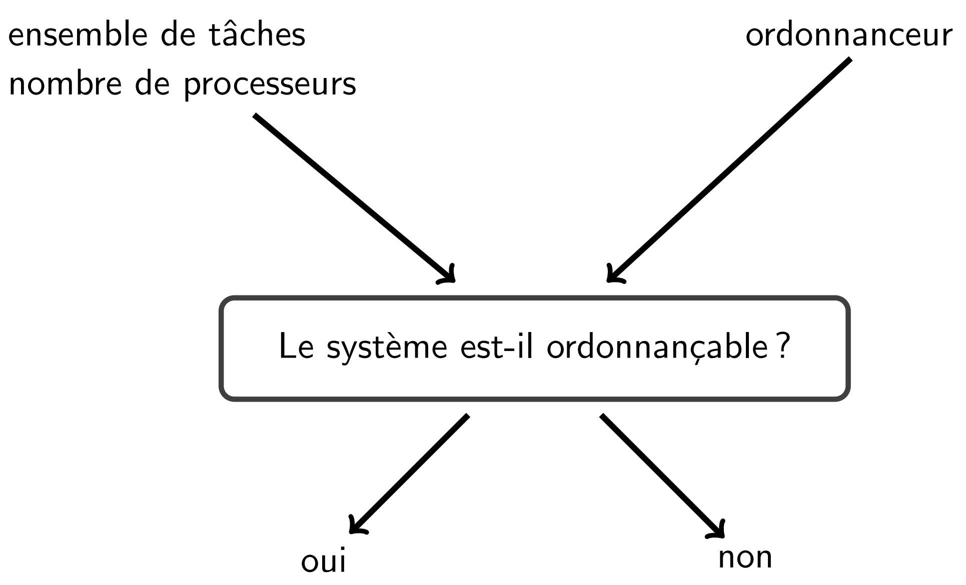
Chapitre VI : Problèmes d'accessibilité pour résoudre un problème d'ordonnancement de tâches

2025 – 2026

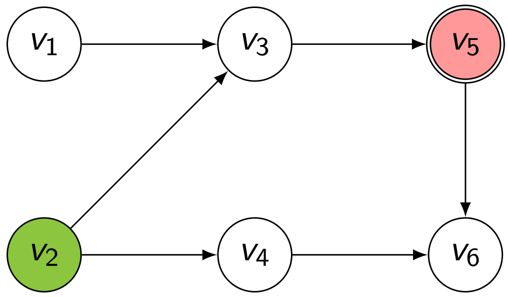
Table des matières

Introduction

Problème d'ordonnancement

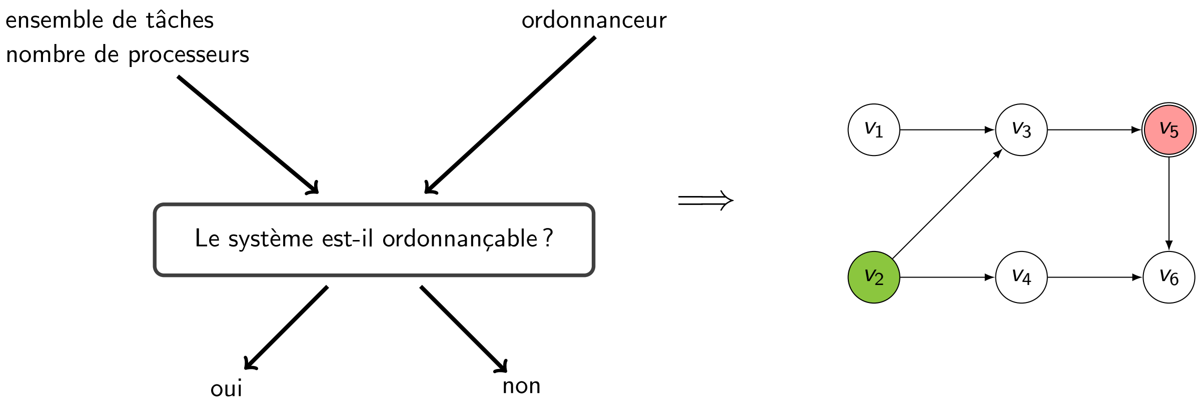


Problème d'accessibilité



- un ensemble de sommets/états : $\{v_1, v_2, v_3, v_4, v_5, v_6\}$;
- un ensemble d'arcs/transitions ;
- v_2 : état initial ;
- v_5 : état cible.

Ordonnancement $\Longrightarrow$ accessibilité

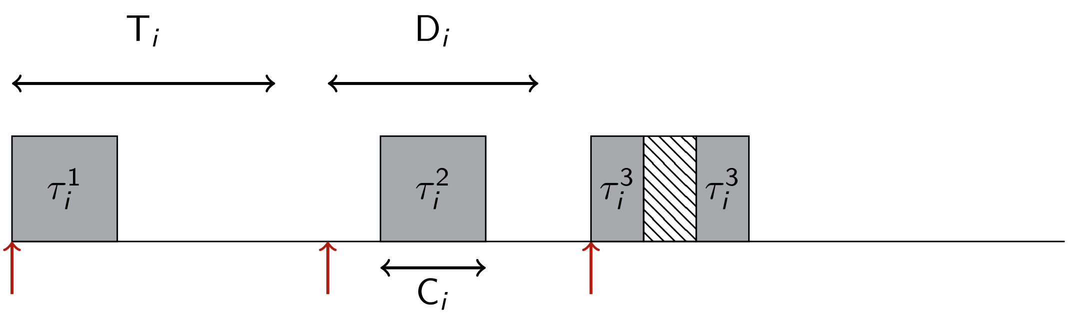


Problème d'ordonnancement

Définition

Un **système temps réel** à tâches **sporadiques**, **préemptives** avec des **deadlines** arbitraires sur des processeurs identiques est caractérisé par les éléments suivants :

- $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ un ensemble de n tâches ;
- pour tout $\tau_i \in \mathcal{T}$, $\mathrm{T}_i > 0$ est le temps minimal entre deux arrivées d'une tâche τ_i ;
- pour tout $\tau_i \in \mathcal{T}$, $\mathrm{D}_i > 0$ est la deadline relative ;
- pour tout $\tau_i \in \mathcal{T}$, $\mathrm{C}_i > 0$ est le temps d'exécution ;



Définition

Un **système temps réel** à tâches **sporadiques**, **préemptives** avec des **deadlines** arbitraires sur des processeurs identiques est caractérisé par les éléments suivants :

- $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ un ensemble de n tâches ;
- pour tout $\tau_i \in \mathcal{T}$, $\mathrm{T}_i > 0$ est le temps minimal entre deux arrivées d'une tâche τ_i ;
- pour tout $\tau_i \in \mathcal{T}$, $\mathrm{D}_i > 0$ est la deadline relative ;
- pour tout $\tau_i \in \mathcal{T}$, $\mathrm{C}_i > 0$ est le temps d'exécution ;

Hypothèses :

- temps discret ;
- travaux non parallélisables mais possibilité de migrer de processeur ;
- la préemption et la migration non coûteux en temps ;
- pour tout $1 \leq i \leq n$, $\mathrm{T}_i \geq \mathrm{D}_i$.

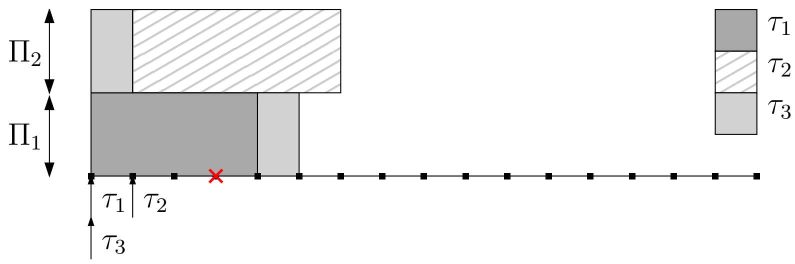
Exemple

Soit un **système** muni de 2 processeurs et de 3 tâches :

τ_i	T_i	D_i	C_i
τ_1	6	6	4
τ_2	6	5	5
τ_3	7	3	2

Ordonnanceur (intuition) : place sur les processeurs les tâches actives de plus haute importance (dont les indices sont les plus petits).

Exemple d'exécution du système

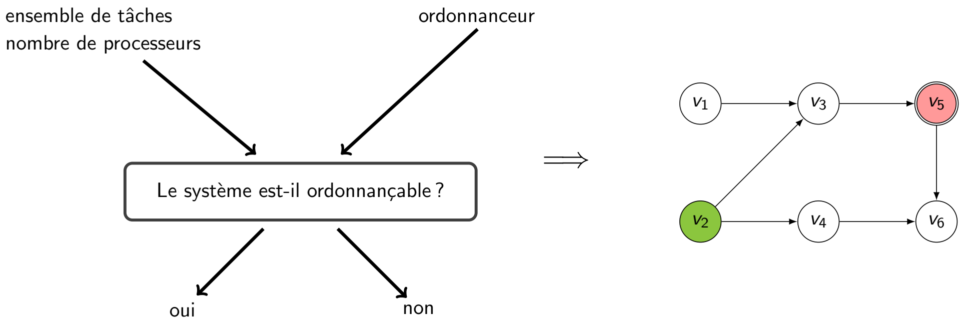


$\rightsquigarrow$ la tâche τ_3 rate sa deadline.
$\rightsquigarrow$ le système n'est pas ordonnançable avec cet ordonnanceur.

Problème : Etant donné un **système** muni de m processeurs et de n tâches ainsi qu'un **ordonnanceur**, existe-t-il une exécution possible du système telle qu'une tâche rate sa deadline ?

- si oui, le système n'est pas ordonnançable avec cet ordonnanceur.
- ce problème est PSPACE-complet.

Supposons dans un premier temps que l'on sache comment passer de ce problème d'ordonnancement à un problème d'accessibilité dans un graphe. (Voir Section 4 pour plus de détails)



- Le graphe obtenu peut être extrêmement grand.
- Les états du graphe peuvent être dotés d'une sémantique particulière qui permet d'améliorer l'algorithme d'accessibilité "naïf".

Problème d'accessibilité

Définitions et algorithme naïf

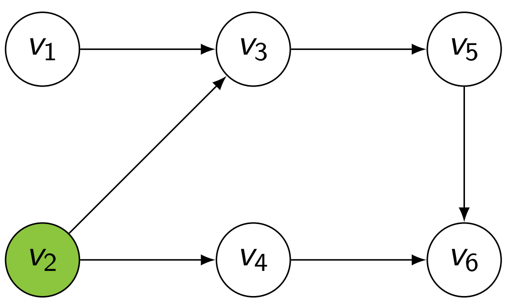
Définition

On note $G = (V, E)$ un graphe avec V un ensemble de sommets/états et $E \subseteq V \times V$ un ensemble d'arcs/transitions.

Soit G un graphe, un **chemin** (fini) dans G est une suite (fini) de sommets de G, $\pi = \pi_1 \dots \pi_k$ telle que pour tout $0 \leq \ell < k$, $(\pi_\ell, \pi_{\ell+1}) \in E$.

Soit G un graphe et $I \subseteq V$ un sous-ensemble de **sommets initiaux**, l'ensemble des **sommets accessibles** depuis I, noté $\mathrm{Reach}(G, I)$, est défini par :

$$\mathrm{Reach}(G, I) = \{v \in V \mid \exists \pi = \pi_1 \dots \pi_\ell \text{ tq } \pi_1 \in I \land \pi_\ell = v\}$$



- $I = \{v_2\}$
- $\mathrm{Reach}(G, I) = \{v_2, v_3, v_4, v_5, v_6\}$

Définition

Soit G un graphe, I un sous-ensemble de sommets initiaux et T un sous-ensemble de **sommets cibles**,
le problème d'accessibilité demande s'il est possible d'atteindre un sommet de T depuis un sommet de I,

$$\mathrm{Reach}(\mathrm{G}, \mathrm{I}) \cap \mathrm{T} \neq \emptyset \ ?$$

Soit G un graphe, soit $v \in \mathrm{V}$ et soit $\mathrm{S} \subseteq \mathrm{V}$:

- $\mathrm{Succ}(v) = \{v' \in \mathrm{V} \mid (v, v') \in \mathrm{E}\}$ est l'ensemble des **successeurs** de v ;
- $\mathrm{Succ}(\mathrm{S}) = \bigcup_{v \in \mathrm{S}} \mathrm{Succ}(v)$

Algorithme "naïf"

Algorithme 1

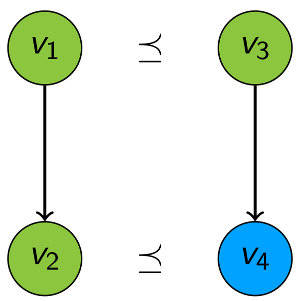
$i \leftarrow 0$
$\mathrm{R}_0 \leftarrow \mathrm{I}$
répéter
    $\mathrm{R}_{i-1} \leftarrow \mathrm{R}_i$
    $i \leftarrow i + 1$
    $\mathrm{R}_i \leftarrow \mathrm{R}_{i-1} \cup \mathrm{Succ}(\mathrm{R}_{i-1})$
    si $\mathrm{R}_i \cap \mathrm{T} \neq \emptyset$ **alors**
        retourner Accessible
jusqu'à $\mathrm{R}_i = \mathrm{R}_{i-1}$
retourner Non accessible

$\rightsquigarrow$ algorithme polynomial en la taille du graphe.

Relations de simulation

Définition

Soit G un graphe, une **relation de simulation**[1] sur G est un préordre[2] $\preceq \subseteq V \times V$ tel que : pour tout v_1, v_2 et $v_3 \in V$ tels que $(v_1, v_2) \in E$ et $v_1 \preceq v_3$, il **existe** $v_4 \in V$ tel que $(v_3, v_4) \in E$ et $v_2 \preceq v_4$.

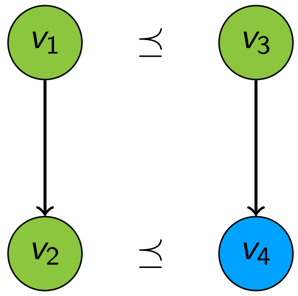


1. Rappel : $\preceq$ est un préordre si (1) pour tout $v \in V$, $v \preceq v$ (réflexivité) et (2) pour tout $v_1, v_2, v_3 \in V$, si $v_1 \preceq v_2$ et $v_2 \preceq v_3$, alors $v_1 \preceq v_3$ (transitivité).

2. La notion de relation de simulation peut-être définie de manière plus générale dans le cadre des systèmes de transitions.

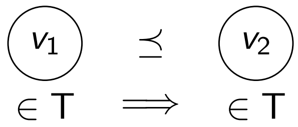
Définition

Soit G un graphe, une **relation de simulation**[1] sur G est un préordre[2] $\preceq \subseteq V \times V$ tel que : pour tout v_1, v_2 et $v_3 \in V$ tels que $(v_1, v_2) \in E$ et $v_1 \preceq v_3$, il **existe** $v_4 \in V$ tel que $(v_3, v_4) \in E$ et $v_2 \preceq v_4$.



Rem : Si $v_1 \preceq v_3$, on dit que v_3 **simule** v_1 ou que v_1 est **simulé par** v_3.

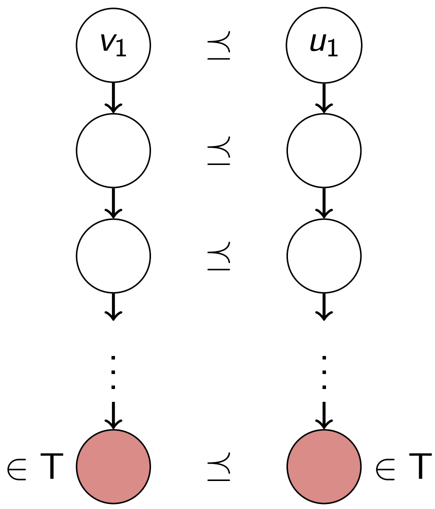
Soit $\preceq \subseteq V \times V$ une relation de simulation et soit $T \subseteq V$, on dit que $\preceq$ est **compatible** avec T si pour tout $v_1, v_2 \in V$ tels que $v_1 \preceq v_2$, si $v_1 \in T$, alors $v_2 \in T$.



(Dans la suite on supposera $\preceq$ compatible avec T)

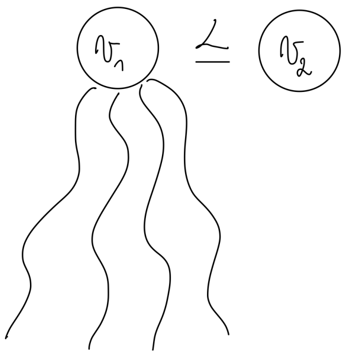
Conséquences

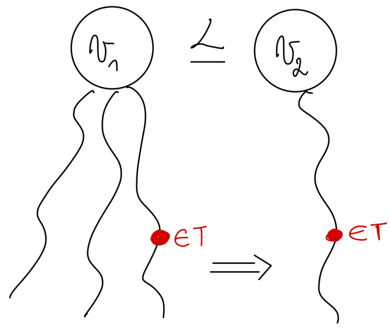
Si on peut atteindre T depuis v_1 et si $v_1 \preceq u_1$, alors on peut atteindre T depuis u_1.



Conséquences

Lorsque v_1 et v_2, tels que $v_1 \preceq v_2$, ont été calculés à une étape de l'algorithme d'accessibilité, nous ne devons **plus calculer les successeurs de v_1.**





Aucun chemin depuis v_1 n'atteint T. On peut ne pas explorer ses successeurs.

Il existe un chemin depuis v_1 qui atteint T. Donc il existe un chemin depuis v_2 qui atteint T.
On peut se contenter d'explorer les successeurs de v_2.

Algorithme amélioré

Algorithme amélioré

Étant donnés G un graphe, $\preceq$ une relation de simulation sur G et $S \subseteq V$ un sous-ensemble de sommets,

$$\mathrm{Max}^{\preceq}(S) = \{v \in S \mid \forall v' \in S, (v \preceq v' \implies v = v')\}.$$

- **Intuitivement :** $\mathrm{Max}^{\preceq}(S)$ est obtenu depuis S en enlevant tous les sommets qui sont simulés par un autre sommet de S.
- Les éléments de $\mathrm{Max}^{\preceq}(S)$ ne sont pas comparables selon $\preceq$.
- Ces ensembles d'éléments incomparables sont appelés des **antichaînes**.

Algorithme amélioré

Algorithme 2

$i \leftarrow 0$
$\tilde{R}_0 \leftarrow \mathrm{Max}^{\preceq}(I)$
répéter
    $\tilde{R}_{i-1} \leftarrow \tilde{R}_i$
    $i \leftarrow i + 1$
    $\tilde{R}_i \leftarrow \tilde{R}_{i-1} \cup \mathrm{Succ}(\tilde{R}_{i-1})$
    $\tilde{R}_i \leftarrow \mathrm{Max}^{\preceq}(\tilde{R}_i)$
    si $\tilde{R}_i \cap T \neq \emptyset$ **alors**
        retourner Accessible
jusqu'à $\tilde{R}_i = \tilde{R}_{i-1}$
retourner Non accessible

Algorithme amélioré

Lemme 1. Pour tout $S \subseteq V$, $\text{Max}^{\preceq}(\text{Succ}(\text{Max}^{\preceq}(S))) = \text{Max}^{\preceq}(\text{Succ}(S))$.

Lemme 2. Pour tout $S_1, S_2 \subseteq V$,

$$\text{Max}^{\preceq}(S_1 \cup S_2) = \text{Max}^{\preceq}(\text{Max}^{\preceq}(S_1) \cup \text{Max}^{\preceq}(S_2)).$$

Lemme 3. Etant donnés G un graphe, I un ensemble de sommets initiaux, T un ensemble de sommets cibles et $\preceq$ une relation de simulation sur G, posons $R_0, R_1, \ldots$ et $\tilde{R}_0, \tilde{R}_1, \ldots$ qui dénotent respectivement la séquence d'ensemble calculée par l'Algorithme 1 et l'Algorithme 2. Pour tout $i \geq 0$, $\tilde{R}_i = \text{Max}^{\preceq}(R_i)$.

Algorithme amélioré

Lemme 3. Etant donnés G un graphe, I un ensemble de sommets initiaux, T un ensemble de sommets cibles et $\preceq$ une relation de simulation sur G, posons $R_0, R_1, \ldots$ et $\tilde{R}_0, \tilde{R}_1, \ldots$ qui dénotent respectivement la séquence d'ensemble calculée par l'Algorithme 1 et l'Algorithme 2. Pour tout $i \geq 0$, $\tilde{R}_i = \text{Max}^{\preceq}(R_i)$.

Preuve : Montrons le par récurrence sur i.

- Cas de base : si $i = 0$, $\tilde{R}_0 = \text{Max}^{\preceq}(I)$ et $R_0 = I \rightsquigarrow$ OK.
- (HR) $\tilde{R}_k = \text{Max}^{\preceq}(R_k)$, montrons que la propriété est vraie pour $i = k+1$.

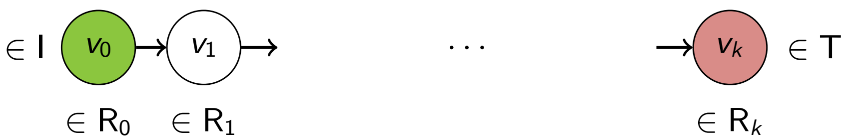
$$\begin{aligned}
\tilde{R}_{k+1} &= \text{Max}^{\preceq}\left(\tilde{R}_k \cup \text{Succ}(\tilde{R}_k)\right) && \text{(Par def.)}\\
&= \text{Max}^{\preceq}\left(\text{Max}^{\preceq}(\tilde{R}) \cup \text{Max}^{\preceq}(\text{Succ}(\tilde{R}_k))\right) && \text{(Par Lem. 2)}\\
&= \text{Max}^{\preceq}\left(\text{Max}^{\preceq}(\text{Max}^{\preceq}(R_k)) \cup \text{Max}^{\preceq}(\text{Succ}(\text{Max}^{\preceq}(R_k)))\right) && \text{(Par HR)}\\
&= \text{Max}^{\preceq}\left(\text{Max}^{\preceq}(R_k) \cup \text{Max}^{\preceq}(\text{Succ}(R_k))\right) && \text{(Par Lem. 1)}\\
&= \text{Max}^{\preceq}\left(R_k \cup \text{Succ}(R_k)\right) && \text{(Par Lem. 2)}\\
&= \text{Max}^{\preceq}(R_{k+1}) && \text{(Par def.)} \quad \square
\end{aligned}$$

Théorème 4. Etant donnés G, I, T et $\preceq$, l'Algorithme 2 termine toujours et retourne "Accessible" ssi T est accessible depuis I.

La preuve se base sur une comparaison entre la séquence $\mathrm{R}_0, \mathrm{R}_1, \ldots$ calculée par l'Algo. 1 (correct et qui termine) et la séquence $\tilde{\mathrm{R}}_0, \tilde{\mathrm{R}}_1, \ldots$ calculée par l'Algo. 2.

La preuve se base sur une comparaison entre la séquence $\mathrm{R}_0, \mathrm{R}_1, \ldots$ calculée par l'Algo. 1 (correct et qui termine) et la séquence $\tilde{\mathrm{R}}_0, \tilde{\mathrm{R}}_1, \ldots$ calculée par l'Algo. 2.

I) Supposons que T **est accessible** dans G en k étapes et pas moins.



I.1) Supposons que Algo 2 s'arrête en $\ell < k$ étapes.

- Soit $\tilde{\mathrm{R}}_\ell \cap \mathrm{T} \neq \emptyset$, mais $\tilde{R}_\ell \subseteq \mathrm{R}_\ell$, donc $\mathrm{R}_\ell \cap \mathrm{T} \neq \emptyset$ et Algo 1 devrait s'arrêter en $< k$ étapes. CONTRAD.
- Soit $\tilde{R}_\ell = \tilde{R}_{\ell-1}$. On a alors que $\mathrm{Max}^{\preceq}(\mathrm{R}_\ell) = \mathrm{Max}^{\preceq}(\mathrm{R}_{\ell-1})$ (par Lem. 3) mais $\mathrm{R}_\ell \neq \mathrm{R}_{\ell-1}$ (sinon Algo 1 s'arrêterait en $< k$ étapes). En particulier, tous les éléments de R_ℓ sont simulés par un élément de $\mathrm{R}_{\ell-1}$.

$$\forall u \in \mathrm{R}_\ell,\ \exists u' \in \mathrm{R}_{\ell-1}\ u \preceq u' \qquad (\star)$$

  - Si il existe $u' \in \mathrm{R}_{\ell-1}$ tel que $v_k \preceq u'$, alors comme $v_k \in \mathrm{T}$, on a aussi $u' \in \mathrm{T}$ et donc $\mathrm{R}_{\ell-1} \cap \mathrm{T} \neq \emptyset$. CONTRAD

Aline Goeminne Algorithmique I – Chapitre VI 25

$\in \mathrm{I}$ $v_0 \to v_1 \to \cdots \to v_k$ $\in \mathrm{T}$

$\in \mathrm{R}_0$ $\in \mathrm{R}_1$ $\in \mathrm{R}_k$

$$\forall u \in \mathrm{R}_\ell,\ \exists u' \in \mathrm{R}_{\ell-1}\ u \preceq u' \qquad (\star)$$

- Sinon, soit $0 \leq m < k$, le plus petit indice[3] tel que

$$\exists u' \in \mathrm{R}_{\ell-1},\ v_m \preceq u' \text{ et } \neg(\exists u'' \in \mathrm{R}_{\ell-1},\ v_{m+1} \preceq v'') \qquad (\star\star)$$

Puisque nous sommes dans la situation suivante :

$$\begin{array}{ccccc} v_m & \preceq & u' & & \\ \downarrow & & \downarrow & & \\ v_{m+1} & \preceq & x & \preceq & y \end{array}$$

Il existe $x \in \mathrm{V}$, tel que $x \in \mathrm{R}_\ell$ (car $u' \in \mathrm{R}_{\ell-1}$), $(u', x) \in \mathrm{E}$ et $v_{m+1} \preceq x$.
Comme $x \in \mathrm{R}_\ell$, il existe $y \in \mathrm{R}_{\ell-1}$ tel que $x \preceq y$ par $(\star)$.
CONTRAD avec $(\star\star)$.

3. Un tel indice existe grâce à $(\star)$ $(v_\ell \in \mathrm{R}_\ell)$ et le point précédent.

$\in \mathrm{I}$ $v_0 \to v_1 \to \cdots \to v_k$ $\in \mathrm{T}$

$\in \mathrm{R}_0$ $\in \mathrm{R}_1$ $\in \mathrm{R}_k$

I.2) Algo 2 s'arrête à l'étape k.

- $v_k \in \mathrm{T}$ et $v_k \in \mathrm{R}_k$ (par hypothèse) $\implies$ $\mathrm{R}_k \cap \mathrm{T} \neq \emptyset$.
- $\tilde{R}_k = \mathrm{Max}^{\preceq}(\mathrm{R}_k)$ (par Lem. 3) donc il existe $v' \in \tilde{R}_k$ tel que $v_k \preceq v'$.
- Comme $v_k \in \mathrm{T}$, $v' \in \mathrm{T}$ et donc $\tilde{R}_k \cap \mathrm{T} \neq \emptyset$
- Algo 2 s'arrête à l'étape k et retourne "Accessible".

II) Supposons que T **n'est pas accessible** dans G.

- Pour tout $i \geq 0$, $\mathrm{R}_i \cap \mathrm{T} = \emptyset$;
- Comme $\tilde{R}_i \subseteq \mathrm{R}_i$, $\tilde{\mathrm{R}}_i \cap \mathrm{T} = \emptyset$ (pour tout $i \geq 0$).
- Algo 2 ne s'arrêtera pas en retournant "Accessible".

Il reste à prouver que la boucle s'arrête.

- Comme T n'est pas accessible, il existe k tel que $\mathrm{R}_k = \mathrm{R}_{k-1}$.
- Dans ce cas $\mathrm{Max}^{\preceq}(\mathrm{R}_k) = \mathrm{Max}^{\preceq}(\mathrm{R}_{k-1})$.
- Donc, $\tilde{R}_k = \tilde{R}_{k-1}$ et Algo 2 s'arrête en retournant "Non accessible".

De l'ordonnancement à l'accessibilité

Définition (rappel)

Un **système temps réel** à tâches **sporadiques**, **préemptives** avec des **deadlines** arbitraires sur des processeurs identiques est caractérisé par les éléments suivants :

- $\mathcal{T} = \{\tau_1, \tau_2, \dots, \tau_n\}$ un ensemble de n tâches ;
- pour tout $\tau_i \in \mathcal{T}$, $\mathsf{T}_i > 0$ est le temps minimal entre deux arrivées d'une tâche τ_i ;
- pour tout $\tau_i \in \mathcal{T}$, $\mathsf{D}_i > 0$ est la deadline relative ;
- pour tout $\tau_i \in \mathcal{T}$, $\mathsf{C}_i > 0$ est le temps d'exécution ;

Hypothèses :

- temps discret ;
- travaux non parallélisables mais possibilité de migrer de processeur ;
- la préemption et la migration non coûteux en temps ;
- pour tout $1 \leq i \leq n$, $\mathsf{T}_i \geq \mathsf{D}_i$.

Questions :

- Comment définir les états/sommets ?
- Comment définir les transitions/arcs entre ces états ?
- Quels états choisir comme états initiaux ?
- Quels états choisir comme états cibles ?

Etats du système

Etats du système

Afin de modéliser l'exécution du système, les seules informations nécessaires à conserver à chaque instant sont :

- pour chaque tâche $\tau_i \in \mathcal{T}$, $\text{nat}(\tau_i)$ [a] est le plus petit laps de temps possible avant l'arrivée d'une prochaine tâche τ_i ;
- pour chaque tâche $\tau_i \in \mathcal{T}$, $\text{rct}(\tau_i)$ [b] est le temps d'exécution restant pour la tâche τ_i en cours de traitement.

a. nat pour (earliest) next arrival time
b. rct pour remaining processing time

Soit $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ un ensemble de tâches, un **état du système** de $\mathcal{T}$ est un uplet $\mathsf{S} = \langle(\text{rct}_\mathsf{S}(\tau_i), \text{nat}_\mathsf{S}(\tau_i))_{1 \leq i \leq n}\rangle$ avec :

- $\text{rct}_\mathsf{S} : \mathcal{T} \to \{0, 1, \ldots, \mathsf{C}_{max}\}$ est une fonction telle que $\mathsf{C}_{max} = \max_i \mathsf{C}_i$;
- $\text{nat}_\mathsf{S} : \mathcal{T} \to \{0, 1, \ldots, \mathsf{T}_{max}\}$ est une fonction avec $\mathsf{T}_{max} = \max_i \mathsf{T}_i$.

L'ensemble des états du système est noté $\text{States}(\mathcal{T})$.

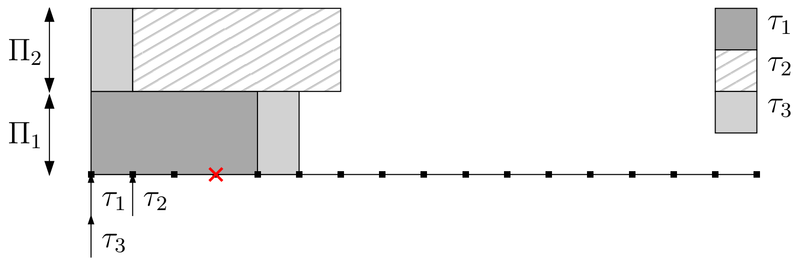
Etats du système

Exemple

Soit un **système** muni de 2 processeurs et de 3 tâches :

τ_i	T_i	D_i	C_i
τ_1	6	6	4
τ_2	6	5	5
τ_3	7	3	2

Exemple d'exécution du système



L'état qui correspond à la croix rouge est donné par :

	τ_1	τ_2	τ_3
rct	1	3	1
nat	3	4	4

$\rightsquigarrow \langle(1, 3), (3, 4), (1, 4)\rangle$

Tâches éligibles, actives, en attente

L'ensemble des tâches **éligibles** dans un état S est donné par :

$$\mathsf{Eligible}(S) = \{\tau_i \in \mathcal{T} \mid \mathsf{nat}_S(\tau_i) = 0 \wedge \mathsf{rct}_S(\tau_i) = 0\}.$$

L'ensemble des tâches **actives** dans un état S est donné par :

$$\mathsf{Active}(S) = \{\tau_i \in \mathcal{T} \mid \mathsf{rct}_S(\tau_i) > 0\}.$$

L'ensemble des tâches **en attente** dans un état S est donné par :

$$\mathsf{Idle}(S) = \mathcal{T} \setminus \mathsf{Active}(S).$$

Laxité d'une tâche et mauvais états

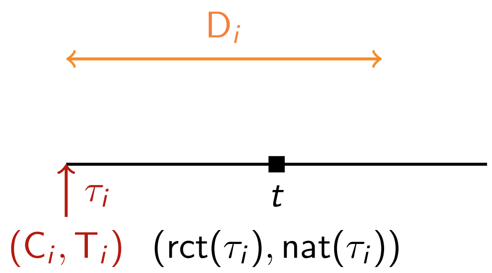
Comment caractériser qu'une tâche râte sa deadline ?

La **laxité** de la tâche τ_i dans un état S est donné par :

$$\mathsf{Laxity}_S(\tau_i) = \mathsf{nat}_S(\tau_i) - (T_i - D_i) - \mathsf{rct}_S(\tau_i).$$

L'ensemble des **mauvais états/états défaillants** est défini par :

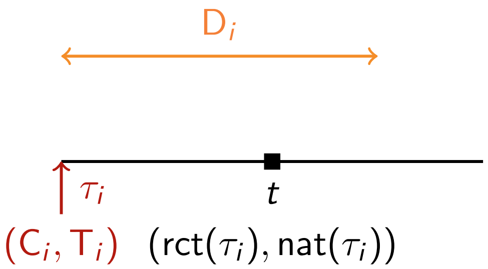
$$\mathsf{Fail}_{\mathcal{T}} = \{S \in \mathsf{States}(\mathcal{T}) \mid \text{ il existe } \tau_i \in \mathsf{Active}(S), \mathsf{Laxity}_S(\tau_i) < 0\}$$



1. $t = T_i - \mathsf{nat}(\tau_i)$
2. temps restant → deadline : $D_i - t$
3. temps d'exécution encore nécessaire : $\mathsf{rct}(\tau_i)$
4. Condition à respecter : (2) $\geq$ (3)

L'ensemble des **mauvais états/états défaillants** est défini par :

$$\text{Fail}_{\mathcal{T}} = \{S \in \text{States}(\mathcal{T}) \mid \text{ il existe } \tau_i \in \text{Active}(S), \text{Laxity}_S(\tau_i) < 0\}$$

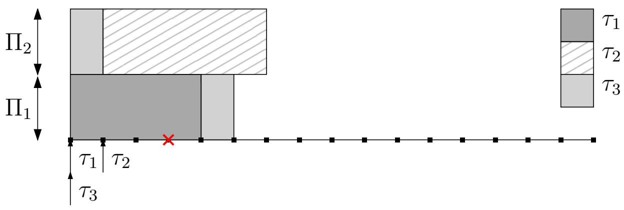


1. $t = T_i - \text{nat}(\tau_i)$
2. temps restant → deadline : $D_i - t$
3. temps d'exécution encore nécessaire : $\text{rct}(\tau_i)$
4. Condition à respecter : (2) ≥ (3)

$$\begin{aligned} (2) &\geq (3) \\ D_i - t &\geq \text{rct}(\tau_i) \\ D_i - (T_i - \text{nat}(\tau_i)) &\geq \text{rct}(\tau_i) \\ \text{nat}(\tau_i) - (T_i - D_i) - \text{rct}(\tau_i) &\geq 0 \\ \text{Laxity}_S(\tau_i) &\geq 0 \end{aligned}$$

Intuition

τ_i	T_i	D_i	C_i
τ_1	6	6	4
τ_2	6	5	5
τ_3	7	3	2



$\langle(0,0),(0,0),(0,0)\rangle$

$(\tau_1 \wedge \tau_3$ arrivent$)$ ↓

$\langle\underline{(4,6)},(0,0),\underline{(2,7)}\rangle$ L(τ_3)=1

(1 unité de temps s'écoule) ↓

$\langle(3,5),(0,0),(1,6)\rangle$ L(τ_3)=1

$(\tau_2$ arrive$)$ ↓

$\langle\underline{(3,5)},\underline{(5,6)},(1,6)\rangle$

(1 unité de temps s'écoule) ↓

$\langle\underline{(2,4)},\underline{(4,5)},(1,5)\rangle$ L(τ_3)=0

(1 unité de temps s'écoule) ↓

$\langle(1,3),(3,4),(1,4)\rangle$ $L(\tau_3) = -1$

Transitions du système

Ordonnanceur

Un **ordonnanceur** pour $\mathcal{T}$ sur m processeurs est une fonction Run : States($\mathcal{T}$) $\longrightarrow \mathcal{P}(\mathcal{T})$ telle que, pour tout S :

- $\mathrm{Run(S)} \subseteq \mathrm{Active(S)}$;
- $0 \leq |\mathrm{Run(S)}| \leq m$.

Exemple d'ordonnanceur :

Soit S un état du système et $\ell = \min\{m, |\mathrm{Active(S)}|\}$, l'ordonnanceur GFP est une fonction $\mathrm{Run_{GFP}}$ telle que : si $\mathrm{Run_{GFP}} = \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_\ell}\}$, alors pour tout $1 \leq j \leq \ell$ et pour tout $\tau_k \in \mathrm{Active(S)} \setminus \mathrm{Run_{GFP}}$, on a $k > i_j$.

Les tâches sont **classées par ordre de priorité** (celles d'indice le plus petit sont les plus prioritaires) et les tâches actives les plus prioritaires sont ordonnancées en premier.

Transitions (2 types de transitions)

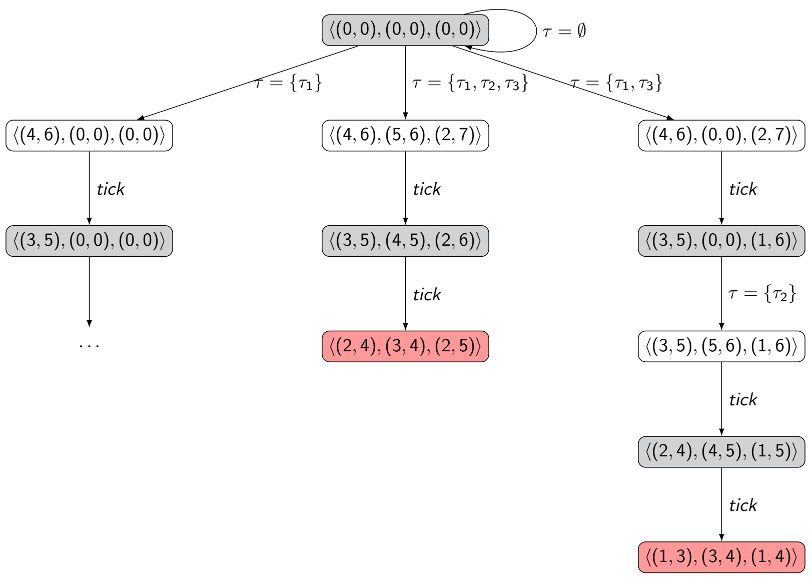
1. **Transitions de requêtes** : si on est dans un état S, un ensemble de tâches $\tau \subseteq$ Eligible(S) éligibles arrivent, si $\tau_i \in \tau$, $\text{rct}(\tau_i) = \text{C}_i$ et $\text{nat}(\tau_i) = \text{T}_i$.
2. **transitions de tick d'horloge** : modélisation de l'écoulement d'une unité de temps ; si on est dans un état S,
   - décrémenter les rct de toutes les tâches de Run(S) (toutes les tâches ordonnancées)
   - décrémenter les nat de toutes les tâches du système.

(Pour plus de formalisme se référer à [GGL13])

[GGL13] : Multiprocessor schedulability of arbitrary deadline sporadic tasks : complexity and antichain algorithm, G. Geeraerts, J. Goosens and M. Lindström, Real-Time Syst 2013.

Exemple

τ_i	T_i	D_i	C_i
τ_1	6	6	4
τ_2	6	5	5
τ_3	7	3	2



Retour au problème d'accessibilité

Soient $\mathcal{T}$ un ensemble de tâches et Run un ordonnanceur sur m processeurs, on définit :

- $\mathsf{G} = (\mathsf{States}(\mathcal{T}), \mathsf{E}')$ avec E' les transitions de requêtes et de tick d'horloge comme définies précédemment.
- $\mathsf{I} = \{\langle(0,0), \ldots, (0,0)\rangle\}$
- $\mathsf{T} = \mathsf{Fail}_{\mathcal{T}}$.

alors on a

$$\mathsf{Reach}(\mathsf{G}, \mathsf{I}) \cap \mathsf{T} \neq \emptyset$$

ssi

il existe une exécution du système telle qu'une tâche dépasse sa deadline (le système n'est pas ordonnançable)

Relation de simulation

Relation de simulation

Soit $\mathcal{T}$ un ensemble de tâches, le préordre *idle tasks* $\preceq_{\mathrm{id}} \subseteq \mathrm{States}(\mathcal{T}) \times \mathrm{States}(\mathcal{T})$ est tel que pour tout $\mathrm{S}_1, \mathrm{S}_2 \in \mathrm{States}(\mathcal{T})$, $\mathrm{S}_1 \preceq_{\mathrm{id}} \mathrm{S}_2$ si et seulement si :

1. $\mathrm{rct}_{\mathrm{S}_1} = \mathrm{rct}_{\mathrm{S}_2}$;
2. Pour tout $\tau_i \in \mathcal{T}$ tel que $\mathrm{rct}_{\mathrm{S}_1}(\tau_i) = 0$: $\mathrm{nat}_{\mathrm{S}_1}(\tau_i) \geq \mathrm{nat}_{\mathrm{S}_2}(\tau_i)$;
3. Pour tout $\tau_i \in \mathcal{T}$ tel que $\mathrm{rct}_{\mathrm{S}_1}(\tau_i) > 0$: $\mathrm{nat}_{\mathrm{S}_1}(\tau_i) = \mathrm{nat}_{\mathrm{S}_2}(\tau_i)$.

Intuitivement : S_2 simule S_1 si

1. les deux états coïncident sur leurs tâches actives (*i.e.*, les tâches τ_i telles que $\mathrm{rct}_{\mathrm{S}_1}(\tau_i) > 0$)
2. la valeur de nat de chaque tâche en attente est plus petite que dans S_2 que S_1.

Exemple :
Soient $\mathrm{S}_1 = \langle(0,3),(2,3)\rangle$, $S_2 = \langle(0,1),(2,3)\rangle$ et $S_3 = \langle(0,1),(3,4)\rangle$.

- $\mathrm{S}_1 \preceq_{\mathrm{id}} \mathrm{S}_2$
- S_2 et S_3 non comparables.

Relation de simulation

Soit $\mathcal{T}$ un ensemble de tâches, le préordre *idle tasks* $\preceq_{\mathrm{id}} \subseteq \mathrm{States}(\mathcal{T}) \times \mathrm{States}(\mathcal{T})$ est tel que pour tout $\mathrm{S}_1, \mathrm{S}_2 \in \mathrm{States}(\mathcal{T})$, $\mathrm{S}_1 \preceq_{\mathrm{id}} \mathrm{S}_2$ si et seulement si :

1. $\mathrm{rct}_{\mathrm{S}_1} = \mathrm{rct}_{\mathrm{S}_2}$;
2. Pour tout $\tau_i \in \mathcal{T}$ tel que $\mathrm{rct}_{\mathrm{S}_1}(\tau_i) = 0$: $\mathrm{nat}_{\mathrm{S}_1}(\tau_i) \geq \mathrm{nat}_{\mathrm{S}_2}(\tau_i)$;
3. Pour tout $\tau_i \in \mathcal{T}$ tel que $\mathrm{rct}_{\mathrm{S}_1}(\tau_i) > 0$: $\mathrm{nat}_{\mathrm{S}_1}(\tau_i) = \mathrm{nat}_{\mathrm{S}_2}(\tau_i)$.

Si $\mathrm{S}_1 \preceq_{\mathrm{id}} \mathrm{S}_2$, alors $\mathrm{Active}(\mathrm{S}_1) = \mathrm{Active}(\mathrm{S}_2)$.

Vrai car $\mathrm{rct}_{\mathrm{S}_1} = \mathrm{rct}_{\mathrm{S}_2}$.

Relation de simulation

Soit $\mathcal{T}$ un ensemble de tâches, le préordre *idle tasks* $\preceq_{\text{id}} \subseteq \text{States}(\mathcal{T}) \times \text{States}(\mathcal{T})$ est tel que pour tout $S_1, S_2 \in \text{States}(\mathcal{T})$, $S_1 \preceq_{\text{id}} S_2$ si et seulement si :

1. $\text{rct}_{S_1} = \text{rct}_{S_2}$;
2. Pour tout $\tau_i \in \mathcal{T}$ tel que $\text{rct}_{S_1}(\tau_i) = 0$: $\text{nat}_{S_1}(\tau_i) \geq \text{nat}_{S_2}(\tau_i)$;
3. Pour tout $\tau_i \in \mathcal{T}$ tel que $\text{rct}_{S_1}(\tau_i) > 0$: $\text{nat}_{S_1}(\tau_i) = \text{nat}_{S_2}(\tau_i)$.

$\preceq_{\text{id}}$ est compatible avec $\text{Fail}_{\mathcal{T}}$.

Preuve.

- Supposons qu'on ait $S_1 \preceq_{\text{id}} S_2$ avec $S_1 \in \text{Fail}_{\mathcal{T}}$.
- Il existe $\tau_i \in \text{Active}(S_1)$ telle que $\text{Laxity}_{S_1}(\tau_i) = \text{nat}_{S_1}(\tau_i) - (T_i - D_i) - \text{rct}_{S_1}(\tau_i) < 0$.
- Par le résultat précédent $\tau_i \in \text{Active}(S_2)$ et par définition de $\preceq_{\text{id}}$, comme $\text{rct}_{S_1}(\tau_i) > 0$, on a $\text{rct}_{S_1}(\tau_i) = \text{rct}_{S_2}(\tau_i)$ et $\text{nat}_{S_1}(\tau_i) = \text{nat}_{S_2}(\tau_i)$.
- Donc $\text{Laxity}_{S_1}(\tau_i) = \text{Laxity}_{S_2}(\tau_i) < 0$ et $S_2 \in \text{Fail}_{\mathcal{T}}$.

Relation de simulation

Soit $\mathcal{T}$ un ensemble de tâches, le préordre *idle tasks* $\preceq_{\text{id}} \subseteq \text{States}(\mathcal{T}) \times \text{States}(\mathcal{T})$ est tel que pour tout $S_1, S_2 \in \text{States}(\mathcal{T})$, $S_1 \preceq_{\text{id}} S_2$ si et seulement si :

1. $\text{rct}_{S_1} = \text{rct}_{S_2}$;
2. Pour tout $\tau_i \in \mathcal{T}$ tel que $\text{rct}_{S_1}(\tau_i) = 0$: $\text{nat}_{S_1}(\tau_i) \geq \text{nat}_{S_2}(\tau_i)$;
3. Pour tout $\tau_i \in \mathcal{T}$ tel que $\text{rct}_{S_1}(\tau_i) > 0$: $\text{nat}_{S_1}(\tau_i) = \text{nat}_{S_2}(\tau_i)$.

Si $S_1 \preceq_{\text{id}} S_2$, alors $\text{Eligible}(S_1) \subseteq \text{Eligible}(S_2)$.

Preuve.

- soit $\tau_i \in \text{Eligible}(S_1)$, alors $\text{nat}_{S_1}(\tau_i) = 0$ et $\text{rct}_{S_1}(\tau_i) = 0$.
- par définition de $\preceq_{\text{id}}$, $\text{rct}_{S_2}(\tau_i) = \text{rct}_{S_1}(\tau_i) = 0$ et $0 = \text{nat}_{S_1}(\tau_i) \geq \text{nat}_{S_2}(\tau_i) \geq 0$
- $\tau_i \in \text{Eligible}(S_2)$

Relation de simulation

Théorème 5. Soit $\mathcal{T}$ un ensemble de tâches et soit Run un ordonnanceur sans mémoire pour $\mathcal{T}$ sur m processeurs. Alors $\preceq_{\mathsf{id}}$ est une relation de simulation sur $\mathsf{G} = (\mathsf{States}(\mathcal{T}), \mathsf{E}')$ compatible avec $\mathsf{Fail}_{\mathcal{T}}$.

Run est sans mémoire si pour tout $\mathsf{S}_1, \mathsf{S}_2 \in \mathsf{States}(\mathcal{T})$ avec $\mathsf{Active}(\mathsf{S}_1) = \mathsf{Active}(\mathsf{S}_2)$,

$$\forall \tau_i \in \mathsf{Active}(\mathsf{S}_1),\ \mathsf{nat}_{\mathsf{S}_1}(\tau_i) = \mathsf{nat}_{\mathsf{S}_2}(\tau_i) \wedge \mathsf{rct}_{\mathsf{S}_1}(\tau_i) = \mathsf{rct}_{\mathsf{S}_2}(\tau_i)$$
$$\implies \mathsf{Run}(\mathsf{S}_1) = \mathsf{Run}(\mathsf{S}_2).$$

Pour aller plus loin

- Papier de référence : [GGL13] : Multiprocessor schedulability of arbitrary deadline sporadic tasks : complexity and antichain algorithm, G. Geeraerts, J. Goosens and M. Lindström, Real-Time Syst 2013.
- Les relations de simulation, bisimulation sont utilisées en model checking : Principles of model checking, Christel Baier and Joost-Pieter Katoen, MIT Press 2008.