Cours ASPD
Modélisation des systèmes répartis
Telecom Nancy 2A IL

Dominique Méry
Telecom Nancy
Université de Lorraine

Année universitaire 2024-2025
2 janvier 2026(3:16pm)

Sommaire

Section Courante

Systèmes de transition

Système de transition

Un système de transition $\mathcal{TS}$ est une structure $(\mathcal{C}, \mathcal{I}, \longrightarrow)$ où

- $\mathcal{C}$: un (fini ou infini) ensemble de configurations
- $\longrightarrow$: une relation binaire sur $\mathcal{C}$
- $\mathcal{I}$: un sous-ensemble de $\mathcal{C}$ constituant les configurations initiales.

Système de transition étiquettée

Un système de transition étiquettée $\mathcal{LTS}$ est une structure $(\mathcal{C}, \mathcal{I}, \mathcal{E}, \longrightarrow)$ où

- $\mathcal{C}$: un (fini ou infini) ensemble de configurations
- $\mathcal{I}$: un sous-ensemble de $\mathcal{C}$ constituant les configurations initiales.
- $\mathcal{E}$: un ensemble d'événements
- $\longrightarrow$: une partie de $\mathcal{C} \times \mathcal{E} \times \mathcal{C}$

Exécution d'un système de transition

Soit un système de transition $\mathcal{ST}$ défini par $(\mathcal{C}, \mathcal{I}, \longrightarrow)$.

Exécution d'un système de transition

Soit un système de transition $\mathcal{ST}$ défini par $(\mathcal{C}, \mathcal{I}, \longrightarrow)$.

- Une configuration terminale $t \in \mathcal{C}$ est une configuration telle que, pour toute configuration $c \in \mathcal{C}$, $(t, c) \notin \longrightarrow$.

Exécution d'un système de transition

Soit un système de transition $\mathcal{ST}$ défini par $(\mathcal{C}, \mathcal{I}, \longrightarrow)$.

- Une configuration terminale $t \in \mathcal{C}$ est une configuration telle que, pour toute configuration $c \in \mathcal{C}$, $(t, c) \notin \longrightarrow$.
- TERM$[\mathcal{ST}]$ est l'ensemble des configurations terminales du système de transition $\mathcal{ST}$.

Exécution d'un système de transition

Soit un système de transition $\mathcal{ST}$ défini par $(\mathcal{C}, \mathcal{I}, \longrightarrow)$.

- Une configuration terminale $t \in \mathcal{C}$ est une configuration telle que, pour toute configuration $c \in \mathcal{C}$, $(t, c) \notin \longrightarrow$.
- TERM$[\mathcal{ST}]$ est l'ensemble des configurations terminales du système de transition $\mathcal{ST}$.
- Une exécution de $\mathcal{ST}$ est une trace maximale σ sur $\mathcal{C}$ satisfaisant les conditions suivantes :
  - $\sigma \in \mathbb{N} \nrightarrow \mathcal{C}$
  - soit il existe une valeur $n \in \mathbb{N}$ telle que $dom(\sigma) = 0..n-1$ et sa longueur est n, soit $dom(\sigma) = \mathbb{N}$ et sa longueur est infinie.
  - Quand l'exécution est finie et de longueur n, alors $\sigma(n-1) \in$ TERM$[\mathcal{ST}]$.

Exécution d'un système de transition

Soit un système de transition $\mathcal{ST}$ défini par $(\mathcal{C}, \mathcal{I}, \longrightarrow)$.

- Une configuration terminale $t \in \mathcal{C}$ est une configuration telle que, pour toute configuration $c \in \mathcal{C}$, $(t, c) \notin \longrightarrow$.
- TERM$[\mathcal{ST}]$ est l'ensemble des configurations terminales du système de transition $\mathcal{ST}$.
- Une exécution de $\mathcal{ST}$ est une trace maximale σ sur $\mathcal{C}$ satisfaisant les conditions suivantes :
  - $\sigma \in \mathbb{N} \nrightarrow \mathcal{C}$
  - soit il existe une valeur $n \in \mathbb{N}$ telle que $dom(\sigma) = 0..n-1$ et sa longueur est n, soit $dom(\sigma) = \mathbb{N}$ et sa longueur est infinie.
  - Quand l'exécution est finie et de longueur n, alors $\sigma(n-1) \in$ TERM$[\mathcal{ST}]$.
- Une configuration $c \in \mathcal{C}$ est accessible, s'il existe une exécution σ telle qu'il existe $i \in dom(\sigma)$ tel que $\sigma(i) = c$ $((c \in ran(\sigma))$

Exécution d'un système de transition

Soit un système de transition $\mathcal{ST}$ défini par $(\mathcal{C}, \mathcal{I}, \longrightarrow)$.

- Une configuration terminale $t \in \mathcal{C}$ est une configuration telle que, pour toute configuration $c \in \mathcal{C}$, $(t, c) \notin \longrightarrow$.
- TERM$[\mathcal{ST}]$ est l'ensemble des configurations terminales du système de transition $\mathcal{ST}$.
- Une exécution de $\mathcal{ST}$ est une trace maximale σ sur $\mathcal{C}$ satisfaisant les conditions suivantes :
  - $\sigma \in \mathbb{N} \nrightarrow \mathcal{C}$
  - soit il existe une valeur $n \in \mathbb{N}$ telle que $dom(\sigma) = 0..n-1$ et sa longueur est n, soit $dom(\sigma) = \mathbb{N}$ et sa longueur est infinie.
  - Quand l'exécution est finie et de longueur n, alors $\sigma(n-1) \in$ TERM$[\mathcal{ST}]$.
- Une configuration $c \in \mathcal{C}$ est accessible, s'il existe une exécution σ telle qu'il existe $i \in dom(\sigma)$ tel que $\sigma(i) = c$ $((c \in ran(\sigma))$
- REACHABLE$[\mathcal{ST}]$ est l'ensemble des configurations accessibles du système de transition $\mathcal{ST}$.

Algorithme local

Algorithme local

Un algorithme local $\mathcal{LA}$ d'un processus P est une structure $(\mathcal{LC}, \mathcal{LI}, \longrightarrow_i, \longrightarrow_s, \longrightarrow_r, \mathcal{M})$ telle que :

- $\mathcal{LC}$: un ensemble de configurations
- $\mathcal{LI}$: un sous-ensemble de $\mathcal{LC}$ constituant les configurations initiales.
- $\mathcal{M}$: un ensemble de messages
- $\longrightarrow_i$: une partie de $\mathcal{LC} \times \mathcal{LC}$
- $\longrightarrow_s$: une partie de $\mathcal{LC} \times \mathcal{M} \times \mathcal{LC}$
- $\longrightarrow_r$: une partie d'une partie de $\mathcal{LC} \times \mathcal{M} \times \mathcal{LC}$
- $\longrightarrow_P \stackrel{def}{=} \Longrightarrow_i \cup \Longrightarrow_r \cup \Longrightarrow_s$

Algorithme local

Algorithme local

Un algorithme local $\mathcal{LA}$ d'un processus P est une structure $(\mathcal{LC}, \mathcal{LI}, \longrightarrow_i, \longrightarrow_s, \longrightarrow_r, \mathcal{M})$ telle que :

- $\mathcal{LC}$: un ensemble de configurations
- $\mathcal{LI}$: un sous-ensemble de $\mathcal{LC}$ constituant les configurations initiales.
- $\mathcal{M}$: un ensemble de messages
- $\longrightarrow_i$: une partie de $\mathcal{LC} \times \mathcal{LC}$
- $\longrightarrow_s$: une partie de $\mathcal{LC} \times \mathcal{M} \times \mathcal{LC}$
- $\longrightarrow_r$: une partie d'une partie de $\mathcal{LC} \times \mathcal{M} \times \mathcal{LC}$
- $\longrightarrow_P \stackrel{def}{=} \Longrightarrow_i \cup \Longrightarrow_r \cup \Longrightarrow_s$

- $\mathcal{M}$ est l'ensemble des messages qui sont échangés par les processus.
- Un message est utilisé une seule fois
- $\mathcal{M}$ pourrait être un multi-ensemble.

Transition locale

Soient lc, m et lc', m' deux configurations de $\mathcal{LA}$.

1. $lc, m \Longrightarrow_P lc', m' \stackrel{def}{=} (lc \longrightarrow_i lc') \wedge m = m'$
2. $lc, m \Longrightarrow_P lc', m' \stackrel{def}{=} \exists\, mes \in \mathcal{M} : \left\{ \begin{array}{l} ((lc, mes, lc') \in \longrightarrow_s) \\ \wedge\ m' = m \cup \{mes\} \end{array} \right.$
3. $lc, m \Longrightarrow_P lc', m' \stackrel{def}{=} \exists\, mes \in \mathcal{M} : \left\{ \begin{array}{l} ((lc, mes, lc') \in \longrightarrow_r) \\ \wedge\ m = m' \cup \{mes\} \end{array} \right.$

Algorithme réparti

Algorithme réparti

Un algorithme réparti $\mathcal{DA}$ pour un ensemble fini de processus $\{P_1,\ldots,P_n\}$ est un ensemble fini d'algorithmes locaux $\{\mathcal{LA}_1,\ldots,\mathcal{LA}_n\}$ où $\mathcal{LC}_i$ est un algorithme local pour le processus P_i, $i \in 1..n$.

Un algorithme réparti $\mathcal{DA}$ pour un ensemble fini de processus $\{P_1,\ldots,P_n\}$ est associé à une structure de transition construite à partir des systèmes de transition des algorithmes locaux :

- $\mathcal{C} = \mathcal{LC}_1 \times \ldots \times \mathcal{LC}_n \times \mathcal{M}$: un ensemble des configurations constituée des configurations locales et des messages possibles.
- $\mathcal{I} == \mathcal{LI}_1 \times \ldots \times \mathcal{LI}_n \times \mathcal{M}$: un sous-ensemble de $\mathcal{C}$ constituant les configurations initiales.
- $\mathcal{M}$: un ensemble de messages
- $\longrightarrow \stackrel{def}{=} \longrightarrow_{P_1} \cup \ldots \cup \longrightarrow_{P_n}$:

Modèle asynchrone

Modèle asynchrone

soient C et C' deux configurations de $\mathcal{C}$:

Modèle asynchrone

soient C et C' deux configurations de $\mathcal{C}$:

1. local : $C \longrightarrow C'$: il existe P de $\{P_1, \ldots, P_n\}$ avec $P = P_i$ tel que $\forall j \in 1..N : j \neq i : C_j = C'_j$ et $C_i \longrightarrow_P C'$

Modèle asynchrone

soient C et C' deux configurations de $\mathcal{C}$:

1. local : $C \longrightarrow C'$: il existe P de $\{P_1, \ldots, P_n\}$ avec $P = P_i$ tel que $\forall j \in 1..N : j \neq i : C_j = C'_j$ et $C_i \longrightarrow_P C'$
2. sending : $C \longrightarrow C'$: il existe P de $\{P_1, \ldots, P_n\}$ avec $P = P_i$ tel que $\forall j \in 1..N : C_j = C'_j$ et $M_2 = M_1 \cup \{m\}$ où $m \in \mathcal{M}$ et $(C_i, m, C'_i) \in \longrightarrow_{P\,s} C'$

Modèle asynchrone

soient C et C' deux configurations de $\mathcal{C}$:

1. local : $C \longrightarrow C'$: il existe P de $\{P_1, \ldots, P_n\}$ avec $P = P_i$ tel que $\forall j \in 1..N : j \neq i : C_j = C'_j$ et $C_i \longrightarrow_P C'$
2. sending : $C \longrightarrow C'$: il existe P de $\{P_1, \ldots, P_n\}$ avec $P = P_i$ tel que $\forall j \in 1..N : C_j = C'_j$ et $M_2 = M_1 \cup \{m\}$ où $m \in \mathcal{M}$ et $(C_i, m, C'_i) \in \longrightarrow_{P\,s} C'$
3. receiving : $C \longrightarrow C'$: il existe P de $\{P_1, \ldots, P_n\}$ avec $P = P_i$ tel que $\forall j \in 1..N : C_j = C'_j$ et $M_1 = M_2 \cup \{m\}$ où $m \in \mathcal{M}$ et $(C_i, m, C'_i) \in \longrightarrow_{P\,r} C'$

Propriétés des algorithmes répartis

Propriétés des algorithmes répartis

- *safety* : toutes les configurations d'un algorithme réparti vérifie une propriété A

Propriétés des algorithmes répartis

- *safety* : toutes les configurations d'un algorithme réparti vérifie une propriété A
- *équité faible* : toute transition activable ou observable à partir d'une configuration donnée finit par être observée

Propriétés des algorithmes répartis

- *safety* : toutes les configurations d'un algorithme réparti vérifie une propriété A
- *équité faible* : toute transition activable ou observable à partir d'une configuration donnée finit par être observée
- *équité forte* : toute transition infirniment souvent activable ou observable à partir d'une configuration donnée finit par être observée

Propriétés des algorithmes répartis

- *safety* : toutes les configurations d'un algorithme réparti vérifie une propriété A
- *équité faible* : toute transition activable ou observable à partir d'une configuration donnée finit par être observée
- *équité forte* : toute transition infirniment souvent activable ou observable à partir d'une configuration donnée finit par être observée
- $P \leadsto Q$: A partir de toute configuration satisfaisant une propritét P, l'algorithme réparti attenindra fatalement une configuration satisfaisant Q.

Exemples de propriété de sûreté

- Exclusion mutuelle : soit une ressource R partagée par un ensemble de processus $\{\mathrm{P}_1, \ldots, \mathrm{P}_n\}$. R est utilisée par au plus un processus de $\{\mathrm{P}_1, \ldots, \mathrm{P}_n\}$.
- La ressource R est utilisée par au plus un processus P ddu système réparti.
- Absence de blocage : soit les processus $\{\mathrm{P}_1, \ldots, \mathrm{P}_n\}$. Aucun des processus n'est bloqué c'est à dire que tout processus peut toujours exécuté une action sauf s'il est terminé.
- Correction Partielle : étant donné un processus de calcul caractérisé par un ensemble d'actions ou d'événements. Si les variables satisfont une précondition $\mathrm{PRE}(x)$, alors si le processus termine, les variables satisfont $\mathrm{POST}(x)$.
- Une propriété de sûreté exprime que rien de mauvais ne peut arriver !

Exemples de propriétés générales

- Chaque fois que le système entre dans une configuration instable, il finira par retrouver un état stable au bout d'un temps fini.
- Le processus P envoie infiniment souvent des messages au processys Q.
- Tout demande est servie
- Les messages sont *toujours* reçus dans l'ordre d'envoi

Observation d'un système réparti

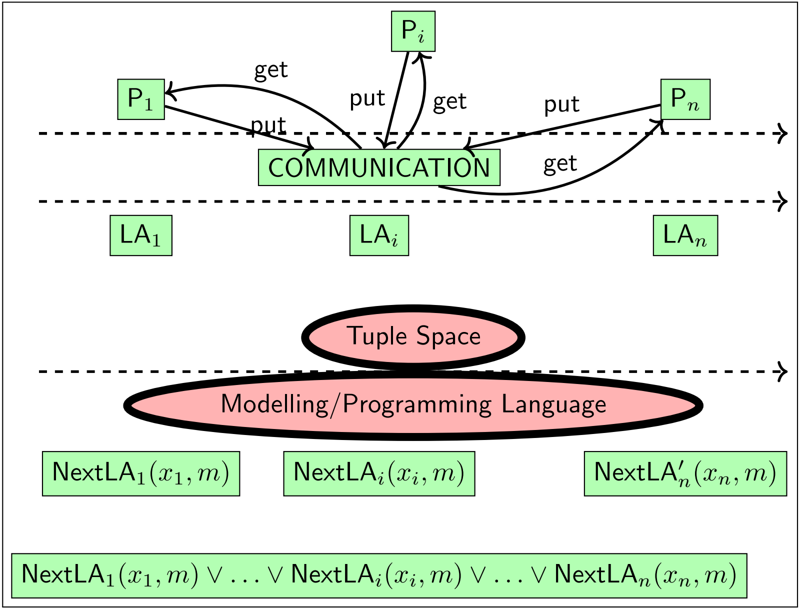
- $u_0 \xrightarrow{\mathsf{e}_0} u_1 \xrightarrow{\mathsf{e}_1} \ldots \xrightarrow{\mathsf{e}_{i-1}} u_i \xrightarrow{\mathsf{e}_i} u_{i+1} \xrightarrow{\mathsf{e}_{i+1}} \ldots$
- e_0 ou e_1 ou $\ldots$ ou e_{i-1} ou e_i ou e_{i+1} ou $\ldots$
- $\mathsf{e} \in \{\mathsf{e}_0, \mathsf{e}_1, \ldots, \mathsf{e}_{i-1}, \mathsf{e}_i, \mathsf{e}_{i+1}, \ldots\}$
- $\mathsf{e} \in E$: E est l'ensemble fini des actions ou des événements observés sur le système modifiant l'état courant.
- $u_0 \xrightarrow{\mathsf{g}} \ldots \xrightarrow{\mathsf{f}} u \xrightarrow{\mathsf{e}} u' \xrightarrow{\mathsf{g}} \ldots$
- Chaque événement modélise la transformation d'une liste de variables d'états appelées *frame* et notée u :

$$\mathbf{if}\ cond(u)\ \mathbf{then}\ u := f(u)\ \mathbf{fi}$$

Non-déterminisme et entrelacement

Les événements de E sont observés les uns à la suite des autres en veillant à ce qu'un événement est observé quand sa *garde* est vraie. On peut ajouter une hypothèse d'équité sur la trace produite.

Distributed System as a collection of local algorithms



Section Courante

]

Modèle relationnel d'un système

Un modèle relationnel $\mathcal{MS}$ pour un système $\mathcal{S}$ est une structure

$$(Th(s,c), x, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$$

où

- $Th(s,c)$ est une théorie définissant les ensembles, les constantes et les propriétés statiques de ces éléments.
- x est une liste de variables flexibles.
- $\textsc{Vals}$ est un ensemble de valeurs possibles pour x.
- $\{r_0, \ldots, r_n\}$ est un ensemble fini de relations reliant les valeurs avant x et les valeurs après x'.
- $\textsc{Init}(x)$ définit l'ensemble des valeurs initiales de x.
- la relation r_0 est la relation $Id[\textsc{Vals}]$, identité sur $\textsc{Vals}$.

Definition

Soit $(Th(s,c), x, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel d'un système $\mathcal{S}$. La relation $\textsc{Next}$ associée à ce modèle est définie par la disjonction des relations r_i :

$\textsc{Next} \stackrel{def}{=} r_0 \vee \ldots \vee r_n$

i

pour une variable x, nous définissons les valeurs suivantes :

- x est la valeur courante de la variable x.
- x' est la valeur suivante de la variable x.
- x_0 ou $\underline{x}$ sont la valeur initiale de la variable x.
- $\overline{x}$ est la valeur finale de la variable X, quand cette notion a du sens.

Exemples de systèmes de transition

- Une grammaire $(\mathrm{N},\mathrm{T},\mathrm{P},\mathrm{S})$ permet de construire un système de transition sur l'ensemble des configurations $(\mathrm{N}\cup\mathrm{T})^{\star}$.
- Une machine de Turing $(\mathrm{Q},\Sigma,\Gamma,\mathrm{B},\delta)$ permet de construire un système de transition sur l'ensemble des configurations $(\Sigma\cup\Gamma)^{\star}$.
- Un réseau de Petri
- Un programme

Section Courante

]

Section Courante

]

Modélisation avec une relation

- Le système est modélisé par
  - une liste de variables flexibles x et une condition initiale notée $Init(x)$
  - une relation de transition modélisant le passage des variabables flexibles de l'état courant à l'état suivant $Next(x, x')$
  - un invariant inductif noté $I(x)$
  - une liste de propriétés de sûreté

Etat courant

Définir un protocole simple avec TLA$^+$

- Envoi de messages
- Modélisation par des ensembles de messages envoyés ou reçus.

$$
\begin{array}{l}
\mathsf{sending}(agent, message, bgent) \stackrel{def}{=} \\
\quad \wedge agent \in AGENTS \\
\quad \wedge bgent \in AGENTS \\
\quad \wedge message \in MESSAGES \\
\quad \wedge << agent, message, bgent >> \notin sent \\
\quad \wedge << agent, message, bgent >> \notin got \\
\quad \wedge sent' = sent \cup << agent, message, bgent >> \\
\quad \wedge got' = got
\end{array}
$$

Définir un protocole simple avec TLA$^+$

- Réception de messages
- Modélisation par des ensembles de messages envoyés ou reçus.

$$
\begin{array}{l}
\mathsf{receiving}(agent, message, bgent) \stackrel{def}{=} \\
\quad \wedge agent \in AGENTS \\
\quad \wedge bgent \in AGENTS \\
\quad \wedge message \in MESSAGES \\
\quad \wedge << agent, message, bgent >> \in sent \\
\quad \wedge << agent, message, bgent >> \notin got \\
\quad \wedge got' = got \cup << agent, message, bgent >> \\
\quad \wedge sent' = sent
\end{array}
$$

Définir un protocole simple avec TLA$^+$

- Définir le système
- Donner des propriétés de sûreté

$$Init \stackrel{def}{=} sent = \varnothing \land got = \varnothing$$

$$\begin{array}{l} Next \stackrel{def}{=} \\ \quad \exists agent, bgent \in AGENTS : \\ \quad \exists message \in MESSAGES : \\ \qquad \lor \quad \textsf{sending}(agent, message, bgent) \\ \qquad \lor \quad \textsf{receiving}(agent, message, bgent) \end{array}$$

. . . et les messages se perdent parfois. . .

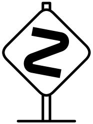
- Le système de gestion des communications peut être non fiable et perdre des messages.
- loosing(agent,message,bgent) modélise la perte d'un message.

$$
\begin{array}{l}
\mathsf{loosing}(agent, message, bgent) \stackrel{def}{=} \\
\quad \wedge agent \in AGENTS \\
\quad \wedge bgent \in AGENTS \\
\quad \wedge message \in MESSAGES \\
\quad \wedge << agent, message, bgent >> \in sent \\
\quad \wedge << agent, message, bgent >> \notin got \\
\quad \wedge got' = got - << agent, message, bgent >> \\
\quad \wedge sent' = sent
\end{array}
$$

Définir un protocole simple avec pertes

$$
\begin{array}{l}
Init \stackrel{def}{=} sent = \varnothing \wedge got = \varnothing \\
\\
Next \stackrel{def}{=} \\
\quad \exists agent, bgent \in AGENTS : \\
\quad \exists message \in MESSAGES : \\
\qquad \begin{array}{ll}
\vee & \mathsf{sending}(agent, message, bgent) \\
\vee & \mathsf{receiving}(agent, message, bgent) \\
\vee & \mathsf{loosing}(agent, message, bgent)
\end{array}
\end{array}
$$

- sûreté *tout message reçu est envoyé* $got \subseteq sent$

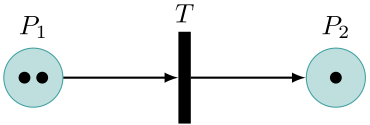


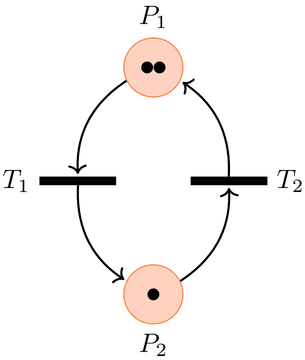
Il est possible que $got = \varnothing$

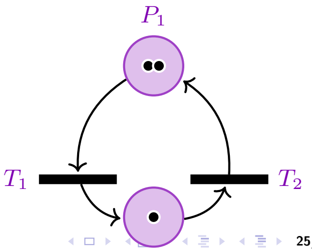
Etat courant

Exemples de réseaux de Petri

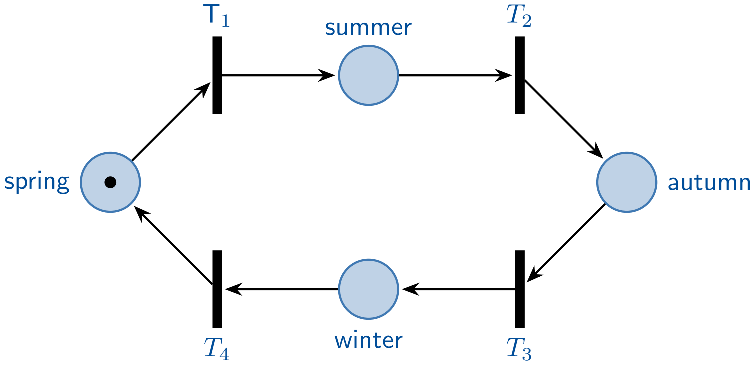
- Graphes bipartis
- Places
- Transitions
- Capacité des places
- Consommation/production des jetons



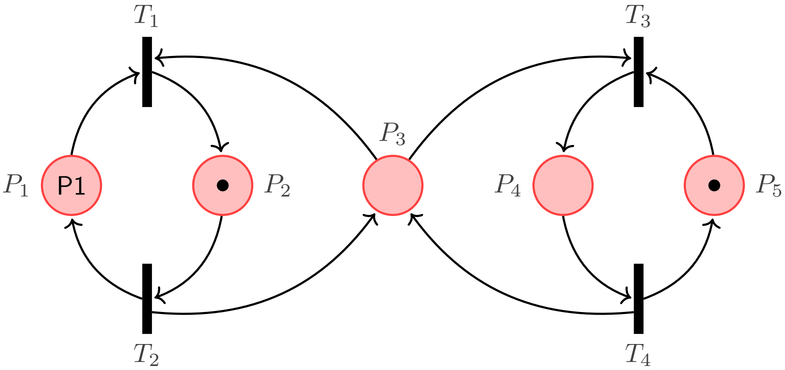




Les quatre saisons . . .



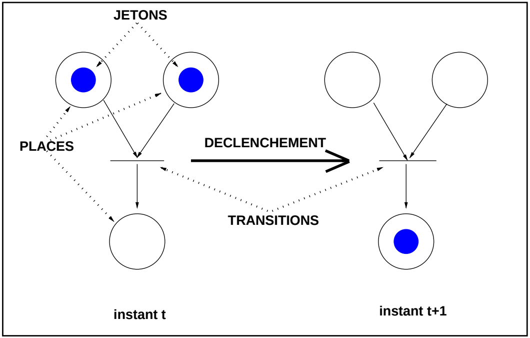
Synchronisation de deux processus concurrents



Réseaux de Petri

- Un réseau de Petri est un graphe dirigé biparti ayant des jetons constituant la marquage.
- Le réseau est caractérisé par son marquage qui évolue au cours de l'exécution des transitions
- Le déclenchement ou l'activation des transitions est fonction de conditions de ressources sur les places avant la transition et après la transition.

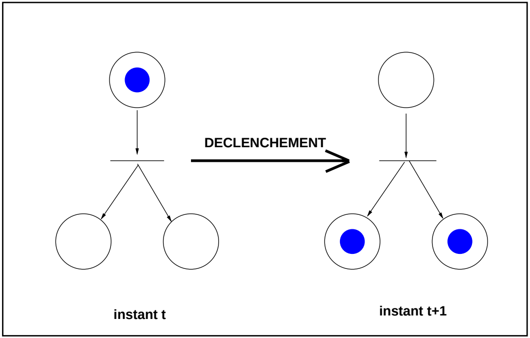
Réseaux de Petri



Réseaux de Petri

- Les transitions peuvent soit consommer des jetons (synchronisation) soit produire de jetons (activités concurrentes) :
- Les ressources sont modélisées par les jetons présents t il peut y avoir une limitation de la capacité des places.

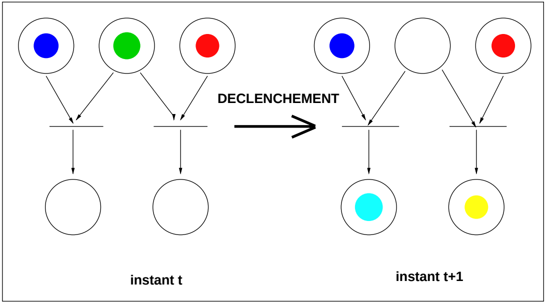
Réseaux de Petri



Réseaux de Petri

- Le partage d'une ressource est modélisé par le partage d'un jeton requis pour l'une ou l'autre des transitions possibles c'est-à-dire activable.
- Le jeton vert est consommé par l'une ou l'autre des deux transitions possibles.

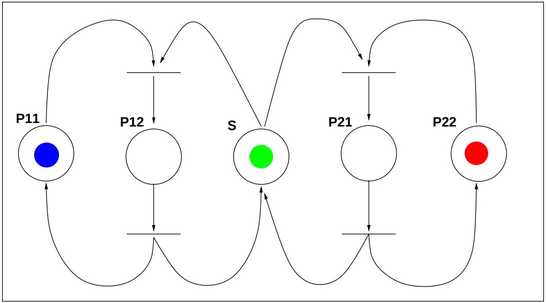
Réseaux de Petri



Réseaux de Petri

- La synchronisation de processus est réalisée par une place S qui est partagée par deux processus P1 et P2 :
- La propriété d'exclusion mutuelle est garantie par l'utilisation exclusive du jeton de la place S par les processus P1 et P2.

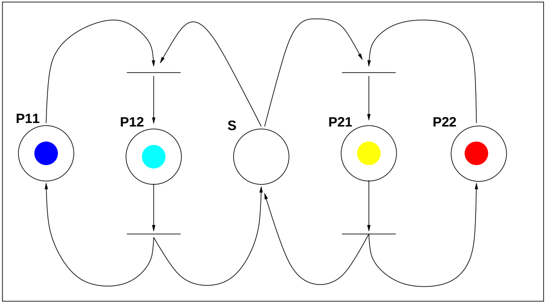
Réseaux de Petri



Réseaux de Petri

- Le déclenchement de l'une des deux transitions est possible quand le jeton vert est en place mais une seule est activée.
- Les réseaux de Petri (1962) ont été créés par **Carl Adam Petri** (avec un C et pas un K) et ont été largement utilisés par la communauté informatique et automatique.
- Des extensions ont été proposées notamment en colorant les jetons ou en ajoutant des probablités aux transitions.

Réseaux de Petri



Réseaux de Petri

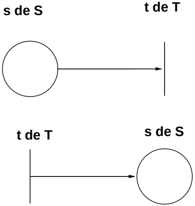
Un réseau de Petri est un uple R=(S,T,F,K,M,W)

- S est l'ensemble (fini) des places.
- T est l'ensemble (fini) des transitions.
- $S \cap T = \varnothing$
- F est la relation du flôt d'exécution :
  $F \subseteq S \times T \cup T \times S$
- K représente la capacité de chaque place :
  $K \in S \rightarrow Nat \cup \{\omega\}$

Réseaux de Petri

- M représente le initial marquage chaque place :
  M $\in$ S $\rightarrow$ Nat et vérifie la condition $\forall$ s $\in$ S : M(s) $\leq$ K(s).
- W représente le poids de chaque arc :
  W $\in$ F $\rightarrow$ Nat
- relation entre la représentation graphique et la définition textuelle :
- un marquage M pour R est une fonction de S dans Nat
  M $\in$ S $\rightarrow$ Nat
  et respectant la condition $\forall\ s \in\ S : M(s)\ \leq\ K(s)$.

Réseaux de Petri



(s,t)

(t,s)

Réseaux de Petri

- une transition t de T est activable à partir de M un marquage de R si
  1. $\forall$ s $\in$ { s' $\in$ S — (s',t) $\in$ F } :
     M(s) $\geq$ W(s,t).
  2. $\forall$ s $\in$ { s' $\in$ S — (t,s') $\in$ F } :
     M(s) $\leq$ K(s) - W(s,t).
- Pour chaque transition t de T, Pre(t) est l'ensemble des places conduisant à t et Post(t) est l'ensemble des places pointées par un lien depuis t :
  Pre(t) = { s' $\in$ S : (s',t) $\in$ F }
  Post(t) = { s' $\in$ S : (t,s') $\in$ F }

Réseaux de Petri

- Soit une transition t de T activable à partir de M un marquage de R :
  1. $\forall$ s $\in$ { s' $\in$ S — (s',t) $\in$ F } :
     M(s) $\geq$ W(s,t).
  2. $\forall$ s $\in$ { s' $\in$ S — (t,s') $\in$ F } :
     M(s) $\leq$ K(s) - W(s,t).
- un nouveau marquage M' est défini à partir de M par : ' $\forall$ s $\in$ S,

$$M'(s) = \begin{cases} \mathrm{M(s)}-\mathrm{W(s,t)}, \text{ si } s \in \textsc{Pre}(t) \text{ - } \textsc{Post}(t) \\ \mathrm{M(s)}+\mathrm{W(t,s)}, \text{ si } s \in \textsc{Post}(t) \text{ - } \textsc{Pre}(t) \\ \mathrm{M(s)}-\mathrm{W(s,t)}+\mathrm{W(t,s)}, \text{ si } s \in \textsc{Pre}(t) \cap \textsc{Post}(t) \\ \mathrm{M(s)}, \text{ sinon} \end{cases}$$

Réseaux de Petri

- Une relation de transition sur l'ensemble des marquages possibles modélise l'activité du réseau :
$\mathrm{M}_0 \xrightarrow{\mathrm{T}_0} \mathrm{M}_1 \xrightarrow{\mathrm{T}_1} \mathrm{M}_2 \xrightarrow{\mathrm{T}_2} \mathrm{M}_3 \xrightarrow{\mathrm{T}_3} \mathrm{M}_4 \xrightarrow{\mathrm{T}_4} \ldots \mathrm{M}_\mathrm{I} \xrightarrow{\mathrm{T}_\mathrm{I}} \mathrm{M}_{\mathrm{I}+1} \xrightarrow{\mathrm{T}_{\mathrm{I}+1}} \ldots$
- Un réseau est bloqué, si aucune de ses transitions n'est activable.
- Un réseau est non bloqué en permanence ou vif, si initialement et pour tout marquage atteint au cours du calcul, au moins une transition est activable.

Réseaux de Petri

Invariant de réseau de Petri

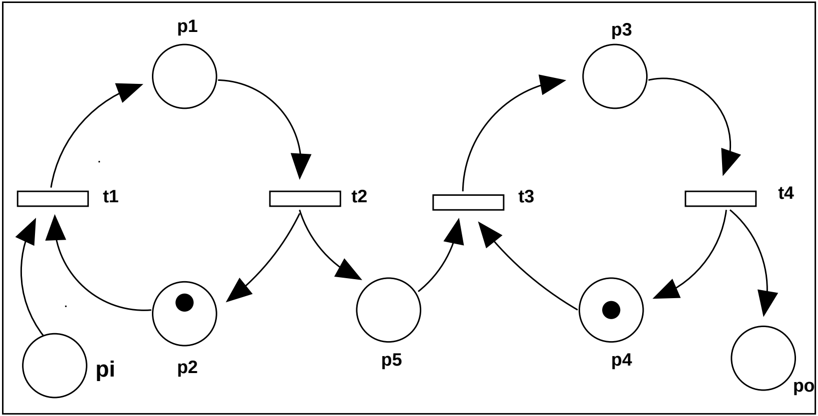
Un invariant de marquage pour un réseau de Petri est un expression de la forme suivante :
$\exists\ p_1, \ldots p_n \in P : \exists q_1, \ldots q_n, C \in \mathbb{Z} :$
$\forall M \in \mathcal{M} : q_1 M(p_1) + q_n M(p_n) = C$

- Les réseaux de Petri sont aussi représentés à l'aide de matrices pour leur flôt et cela définit une algèbre sur les réseaux de Petri : $M_{\mathrm{K}} = M_{\mathrm{I}} + W.S$ est l'équation fondamentale permettant de définir la relation de transition.
- Les réseaux de Petri permettent d'exprimer des contraintes de synchronisation

Réseaux de Petri

- Le modèle est aussi puissant que les machines de Turing
- Le modèle permet de modéliser les activités concurrentes et non déterministes.
- Le Graphcet est une forme proche des réseaux de Petri et est utilisé pour la modélisation des systèmes.
- La notion sous-jacente est celle des systèmes de transition discrets.

Exemple d'un réseau de Petri



Modélisation de petri10

EXTENDS $Naturals, TLC$
CONSTANTS $Places, N, Q, B$
VARIABLES M

t1 $\stackrel{def}{=}$

$\wedge M["p2"] = 1 \wedge M["pi"] \geq 1 \wedge M["p1"] = 0$
$\wedge M' = [[[M EXCEPT !["p1"] = 1] EXCEPT !["pi"] = M["pi"]-1] EXC$

t2 $\stackrel{def}{=}$

$\wedge M["p1"] = 1 \wedge M["p5"] \leq B$
$\wedge M' = [[[M EXCEPT !["p1"] = 0] EXCEPT !["p5"] = M["p5"]+1] EXC$

t3 $\stackrel{def}{=}$

$\wedge M["p5"] \geq 1 \wedge M["p3"] = 0$
$\wedge M' = [[[M EXCEPT !["p3"] = 1] EXCEPT !["p5"] = M["p5"]-1] EXC$

t4 $\stackrel{def}{=}$

$\wedge M["p3"] = 1 \wedge M["p4"] = 0 \wedge M["po"] < Q$
$\wedge M' = [[[M EXCEPT !["p3"] = 0] EXCEPT !["po"] = M["po"]+1] EXC$

Modélisation de petri10

$$Init1 \stackrel{def}{=} M = [p \in Places| - > IF\ p\ \in\ "p4", "p2" THEN\ 1\ ELSE\ IF\ p$$

$$Init \stackrel{def}{=} Init1$$
$$Next \stackrel{def}{=} t1 \vee t2 \vee t3 \vee t4$$
$$Petri \stackrel{def}{=} Init \wedge \Box[Next]_{<M>}$$
$$TypeInvariant \stackrel{def}{=} \forall p \in Places : M[p] \geq 0$$
$$Inv1 \stackrel{def}{=} M["pi"] + M["p5"] + M["po"] + M["p1"] + M["p3"] = N$$
$$Inv2 \stackrel{def}{=} M["po"] \# Q$$
$$Inv3 \stackrel{def}{=} M["p5"] \# 1$$
$$Inv \stackrel{def}{=} TypeInvariant$$

Choix de modélisation

- Donner le ≪ quoi ≫ : spécification de ce que fait le protocole

Choix de modélisation

- Donner le ≪ quoi ≫ : spécification de ce que fait le protocole
  - envoi d'un message m par un processus P à un processus Q

Choix de modélisation

- Donner le ≪ quoi ≫ : spécification de ce que fait le protocole
  - envoi d'un message m par un processus P à un processus Q
  - décomposition en plusieurs phases

Choix de modélisation

- Donner le « quoi » : spécification de ce que fait le protocole
  - envoi d'un message m par un processus P à un processus Q
  - décomposition en plusieurs phases
- Donner le « comment » : simulation du protocole par des événements et des phases des couches plus basses

Section Courante

]

Définir un module en TLA^{+}

- Définir les données : chaines, nombres, ensembles, fonctions
- Définir les actions : relatioin entre des variables non primées et primées
- Définir le système : donner ses conditions initiales et la relation de transition
- Définir les propriétés : sûreté, non-blocage, accessibilité

Le langage TLA$^+$

1. L'entité de structuration syntaxique est le MODULE dont le nom *name* est utilisé comme identificatuer du fichier en ajoutant le suffixe *.tla*
2. Un module peut étendre d'autres modules par la directive EXTENDS indiquant que toiut ce qui est dans ces modules est utilisable dans le module courant
3. Un module peut déclarer des constantes par la directive CONSTANTS et ces constantes sont instanciées dans un modèle.
4. Un module peut déclarer des variables dites flexibles par la directive VARIABLES et chaque variable X a deux références possibles x valeur courante et x' valeur suivante
5. Un module peur définir une entité en indiquant son nom *name* et une expression *expr* comportant des éléments déjà définis :
   `name == expr`

Conventions pour l'outil TLC

- Toute action est écrite sous la forme suivante :

$$\boxed{\begin{array}{l} name \stackrel{def}{=} \\ \qquad \wedge\ G(x,y,u) \\ \qquad \wedge\ u' = f(u) \\ \qquad \wedge\ y' = y \end{array}}$$

- y est une variable qui n'est pas modifiée
- f est une fonction calculable ou codable
- x est une coinstante

Un exemple simple et complet

```
-------------------- MODULE pgcd -----------------
EXTENDS Naturals,TLC
CONSTANTS a,b
VARIABLES  x,y
-----------------------------------
Init == x=a /\ y=b
-----------------------------------
a1 == x > y /\ x'=x-y /\ y'=y
a2 == x < y /\ y'=y-x /\ x'=x
over == x=y /\ x'=x /\ y'=y
------------------------------------
Next == a1 \/ a2 \/ over
------------------------------------
test == x # y
prop == x \geq 0
prop2 == x+y \leq a+b
=============================================
```

Section Courante

]

Processes

```
—— MODULE module_name ——
\* TLA+ code

(* --algorithm algorithm_name
variables global_variables

process p_name = ident
variables local_variables
begin
  \* pluscal code
end process

process p_group \in  set
variables local_variables
begin
  \* pluscal code
end process

end algorithm; *)
```

Macros and Procedures

```
macro Name(var1, ...)
begin
 \* something to write
end macro;

procedure Name(arg1, ...)
variables var1 = ... \* not  \in, only =
begin
  Label:
  \* something
  return;
end procedure;
```

Exemples

```
process pro = "test"
begin
  print<<"test">>;
end process

process (pro \in 0..8)
begin
  print<<"test",self>>;
end process
```

Processus en PlusCal

- A multiprocess algorithm contains one or more processes.
- A process begins in one of two ways :
  - defining a set of processes : `process ( ProcName` $\in$ `IdSet )`
  - defining one process with an identifier `process ( ProcName = Id )`
- `self` designates the current process
- Communication using shared variables defined as global variables
- Communication using sequences introduced by the `EXTENDS Sequences` and usong operation sover sequences as `Head`, `Append` and `Tail`

processus R

```
—algorithm ex_process {
  variables
    input = <<>>, output = <<>>,
    msgChan = <<>>, ackChan = <<>>,
    newChan = <<>>;
\* defining macros
  process (Sender = "S")
  {

  }; \* end Sender process block
  process (Receiver = "R")
  {

  }; \* end Receiver process block

} \* end algorithm
\sec
```

Synchronisation des processus

How to do

`await (expression)` :

- A step containing the statement `await expr` can be executed only when the value of the Boolean expression expr is TRUE.
- Although it usually appears at the beginning of a step, an await statement can appear anywhere within the step.
- `await` can be used as well as `when`

MODULE pluscaltut2

```
EXTENDS Integers, Sequences, TLC, FiniteSets

(*
--algorithm Tut2 {
variables x = 0;

process (one = 1)

variables temp
{

 A:
        temp := x + 1;


        x := temp;

};


process (two = 2)

variables temp
{
  CC:
        temp := x + 1;

        x:= temp;
};
}
end algorithm;

*)
\* BEGIN TRANSLATION (chksum(pcal) = "b54fa406" /\ chksum(tla) = "e84b4125")
\* Process variable temp of process one at line 10 col 11 changed to temp_
CONSTANT defaultInitValue
VARIABLES x, pc, temp_, temp

vars == << x, pc, temp_, temp >>

ProcSet == {1} \cup {2}

Init == (* Global variables *)
        /\ x = 0
```

```
---------------------------- MODULE pluscaltut3 ----------------------------
EXTENDS Integers, Sequences, TLC, FiniteSets
(*
--algorithm Tut3 {
variables x = 0;

process (one = 1)
{
  A:
    x := x + 1;
  B:
    await  x = 1;
  C:
    print <<"x=",x>>;
};

process (two = 2)
{
  D:
    await x = 1;
  E:
    assert x = 1;
  F:
    x := x - 2;
```

Gestion des communications

How to do

Using macros for defining sending and receiving primitives over sequences.

```
--algorithm ex_process {
  variables
    input = <<>>, output = <<>>,
    msgChan = <<>>, ackChan = <<>>,
    newChan = <<>>;
  macro Send(m, chan) {
    chan := Append(chan, m);
  }
  macro Recv(v, chan) {
    await chan # <<>>;
    v := Head(chan);
    chan := Tail(chan);
  }

* Processes S and R

} \* end algorithm
```

Definir les processus S et R

```
—algorithm ex_process {
  variables
    input = <<>>, output = <<>>,
    msgChan = <<>>, ackChan = <<>>,
    newChan = <<>>;
\* defining macros
  process (Sender = "S")
  variables msg;
  {
  sending:  Send("Hello", msgChan);
  printing:    print <<"Sender", input>>;
  }; \* end Sender process block
  process (Receiver = "R")
  {
  waiting: Recv(msg, msgChan);
  adding: output := Append(output, msg);
  printing:  print <<"Receiver", output>>;
  }; \* end Receiver process block
} \* end algorithm
```

```
---------------------------- MODULE pluscaltut1 ----------------------------
EXTENDS Integers, Sequences, TLC, FiniteSets
(*
--wf
--algorithm Tut1 {
 variables x = 0;


 process (one = 1)
 {
   A: assert x \in {0,1};
      x := x - 1;
   B: assert x \in {-1,0} ;
      x := x * 3;
    BB: assert x \in {-3,-2,0,1};
 };

 process (two = 2)
 {
   C: assert x \in {-3,-2,-1,0,1};
      x := x + 1;
   D:
      assert x \in {-2,-1,0,1,2};
 };
```

Autres instructions

- `with` : `with (id` $\in$ `S) body` is executed by executing the (possibly compound) statement body with identifier id equal to a nondeterministically chosen element of S.
- `either`
- `call`
- `return`
- `goto`

Section Courante

]

- Importance de l'abstraction
- Raffiner la vue des modèles
- Intégration du temps
- Intégration des probabilités