



IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom



Modélisation de la concurrence

Fabien Dagnat

Concurrence – C2– janvier 2026

Le principe : modéliser

- ▶ Modèle : représentation simplifiée d'un système
- ▶ Pour concevoir et valider une conception
 - ▶ animer le modèle pour visualiser le comportement
 - ▶ vérifier mécaniquement des propriétés
- ▶ Un modèle, lequel ?
 - ▶ en UE cœur MAPD, des modèles en UML (diagrammes de séquence, d'état ou d'activité)
 - ▶ dans cette UE, des modèles
 - ▶ sous forme de LTS (*Labelled Transition System*) des automates
 - ▶ définis en FSP (*Finite State Processes*)
- ▶ Modèles formels pour vérifier mathématiquement des propriétés
 - ▶ dans cette UE, outil LTSA *model checking*

Modélisation de comportement

► Objectif de cette partie

- 1 construire des modèles dans le langage FSP
- 2 passer d'un modèle FSP à son automate (LTS) et vice et versa
- 3 composer des automates (LTS)
- 4 savoir décomposer un système puis construire son modèle par composition

► De la lecture éventuelle



Jeff Magee & Jeff Kramer,
Concurrency: State Models & Java Programs, 2nd ed,
Wiley, <http://www.doc.ic.ac.uk/~jnm/book>
Chapitre 1 à 8 principalement

- 1 Modéliser les processus
- 2 FSP, une syntaxe algébrique
- 3 Composition de processus

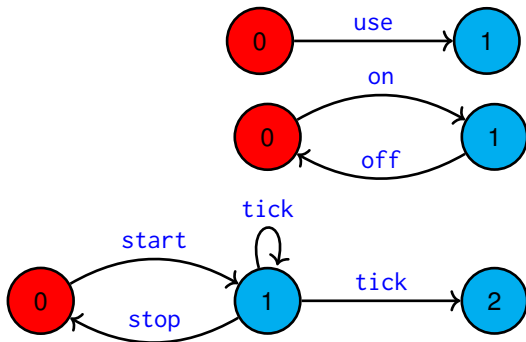
1 Modéliser les processus

2 FSP, une syntaxe algébrique

3 Composition de processus

Processus et leur comportement

- ▶ Suite d'actions avec choix et boucles \Rightarrow automate
 - ▶ on ignore le comportement interne non observable
- ▶ LTS (*Labelled Transition System*)
- ▶ Des exemples
 - ▶ Équipement à usage unique
 - ▶ Une lampe
 - ▶ Une horloge (\mathcal{H})

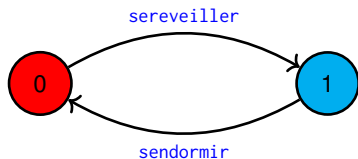


Modéliser la journée d'un élève

- ▶ Il faut trouver
 - ▶ les états (dont l'initial), les actions et les transitions
 - ▶ certains pensent mieux *état*, d'autres *action*
- ▶ Qu'est-ce qu'une action / un état ?
 - ▶ un *truc atomique* observable à l'extérieur du processus
 - ▶ \Rightarrow de nombreux modèles possibles

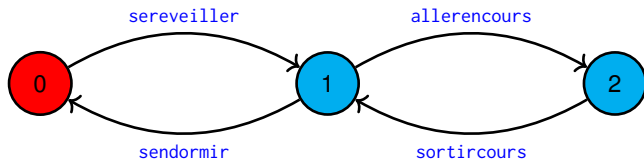
Modéliser la journée d'un élève

- ▶ Il faut trouver
 - ▶ les états (dont l'initial), les actions et les transitions
 - ▶ certains pensent mieux *état*, d'autres *action*
- ▶ Qu'est-ce qu'une action / un état ?
 - ▶ un *truc atomique* observable à l'extérieur du processus
 - ▶ \Rightarrow de nombreux modèles possibles



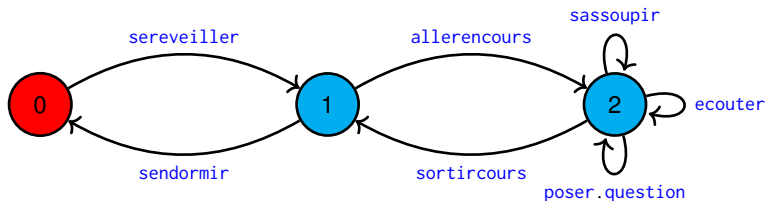
Modéliser la journée d'un élève

- ▶ Il faut trouver
 - ▶ les états (dont l'initial), les actions et les transitions
 - ▶ certains pensent mieux *état*, d'autres *action*
- ▶ Qu'est-ce qu'une action / un état ?
 - ▶ un *truc atomique* observable à l'extérieur du processus
 - ▶ \Rightarrow de nombreux modèles possibles



Modéliser la journée d'un élève

- ▶ Il faut trouver
 - ▶ les états (dont l'initial), les actions et les transitions
 - ▶ certains pensent mieux *état*, d'autres *action*
- ▶ Qu'est-ce qu'une action / un état ?
 - ▶ un *truc atomique* observable à l'extérieur du processus
 - ▶ \Rightarrow de nombreux modèles possibles



- ▶ Remarques : *événement* vs action ; entrée vs sortie

Qu'est-ce qu'un LTS ?

- ▶ Un LTS est un quadruplet (S, A, T, s_0) où
 - ▶ S est l'ensemble de ses états (ici, un état est un entier)
 - ▶ A est l'**alphabet**, l'ensemble de ses actions, une action est une suite de mots¹ séparés par des points
 - ▶ T est sa relation de transition, ses éléments sont des triplets (s_1, a, s_2) notés $s_1 \xrightarrow{a} s_2$, $T \subseteq S \times A \times S$
 - ▶ s_0 est son état initial, $s_0 \in S$
- ▶ Pour \mathcal{H}
 - ▶ $S = \{0, 1, 2\}$
 - ▶ $A = \{\text{start}, \text{tick}, \text{stop}\}$
 - ▶ $T = \{0 \xrightarrow{\text{start}} 1, 1 \xrightarrow{\text{stop}} 0, 1 \xrightarrow{\text{tick}} 1, 1 \xrightarrow{\text{tick}} 2\}$
 - ▶ $s_0 = 0$

¹. commence par une minuscule

Quelques notions utiles

► Les actions **déclenchables** : $\text{fireable} : S \rightarrow \mathcal{P}(A)$

► $\text{fireable}(s) = \left\{ a \in A \mid \exists s' \in S, s \xrightarrow{a} s' \in T \right\}$

► pour \mathcal{H}

► $\text{fireable}(0) = ?$

$\text{fireable}(1) = ?$

$\text{fireable}(2) = ?$

Quelques notions utiles

► Les actions **déclenchables** : $\text{fireable} : S \rightarrow \mathcal{P}(A)$

$$\text{fireable}(s) = \left\{ a \in A \mid \exists s' \in S, s \xrightarrow{a} s' \in T \right\}$$

► pour \mathcal{H}

$$\text{fireable}(0) = \{\text{start}\} \qquad \text{fireable}(1) = \{\text{tick}, \text{stop}\} \qquad \text{fireable}(2) = \{\}$$

Quelques notions utiles

► Les actions **déclenchables** : $\text{fireable} : S \rightarrow \mathcal{P}(A)$

$$\text{fireable}(s) = \{a \in A \mid \exists s' \in S, s \xrightarrow{a} s' \in T\}$$

► pour \mathcal{H}

$$\text{fireable}(0) = \{\text{start}\}$$

$$\text{fireable}(1) = \{\text{tick}, \text{stop}\}$$

$$\text{fireable}(2) = \{\}$$

► Un état **bloqué** : aucune action n'est déclenchable

$$\text{blocked}(s) = (\text{fireable}(s) = \{\})$$

► pour \mathcal{H}

$$\text{blocked}(0) = ?$$

$$\text{blocked}(1) = ?$$

$$\text{blocked}(2) = ?$$

Quelques notions utiles

► Les actions **déclenchables** : $\text{fireable} : S \rightarrow \mathcal{P}(A)$

$$\text{fireable}(s) = \{a \in A \mid \exists s' \in S, s \xrightarrow{a} s' \in T\}$$

► pour \mathcal{H}

$$\text{fireable}(0) = \{\text{start}\} \qquad \text{fireable}(1) = \{\text{tick}, \text{stop}\} \qquad \text{fireable}(2) = \{\}$$

► Un état **bloqué** : aucune action n'est déclenchable

$$\text{blocked}(s) = (\text{fireable}(s) = \{\})$$

► pour \mathcal{H}

$$\text{blocked}(0) = \text{false} \qquad \text{blocked}(1) = \text{false} \qquad \text{blocked}(2) = \text{true}$$

Quelques notions utiles II

► Les états suivants : $\text{next} : S \rightarrow \mathcal{P}(S)$

$$\text{next}(s_1) = \left\{ s_2 \in S \mid \exists a \in A, s_1 \xrightarrow{a} s_2 \in T \right\}$$

► pour \mathcal{H}

$$\text{next}(0) = ?$$

$$\text{next}(1) = ?$$

$$\text{next}(2) = ?$$

$$\text{blocked}(s) = (\text{next}(s) = \{\})$$

Quelques notions utiles II

► Les états suivants : $\text{next} : S \rightarrow \mathcal{P}(S)$

► $\text{next}(s_1) = \{s_2 \in S \mid \exists a \in A, s_1 \xrightarrow{a} s_2 \in T\}$

► pour \mathcal{H}

► $\text{next}(0) = \{1\}$

$\text{next}(1) = \{0, 1, 2\}$

$\text{next}(2) = \{\}$

► $\text{blocked}(s) = (\text{next}(s) = \{\})$

Quelques notions utiles II

► Les états suivants : $\text{next} : S \rightarrow \mathcal{P}(S)$

$$\text{next}(s_1) = \left\{ s_2 \in S \mid \exists a \in A, s_1 \xrightarrow{a} s_2 \in T \right\}$$

► pour \mathcal{H}

$$\text{next}(0) = \{1\}$$

$$\text{next}(1) = \{0, 1, 2\}$$

$$\text{next}(2) = \{\}$$

$$\text{blocked}(s) = (\text{next}(s) = \{\})$$

► Un état **puits** : bloqué ou toutes ses actions déclenchables bouclent

$$\text{sink}(s) = (\text{next}(s) \subseteq \{s\})$$

► pour \mathcal{H}

$$\text{sink}(0) = ?$$

$$\text{sink}(1) = ?$$

$$\text{sink}(2) = ?$$

Quelques notions utiles II

► Les états suivants : $\text{next} : S \rightarrow \mathcal{P}(S)$

► $\text{next}(s_1) = \{s_2 \in S \mid \exists a \in A, s_1 \xrightarrow{a} s_2 \in T\}$

► pour \mathcal{H}

► $\text{next}(0) = \{1\}$

$\text{next}(1) = \{0, 1, 2\}$

$\text{next}(2) = \{\}$

► $\text{blocked}(s) = (\text{next}(s) = \{\})$

► Un état **puits** : bloqué ou toutes ses actions déclenchables bouclent

► $\text{sink}(s) = (\text{next}(s) \subseteq \{s\})$

► pour \mathcal{H}

► $\text{sink}(0) = \text{false}$

$\text{sink}(1) = \text{false}$

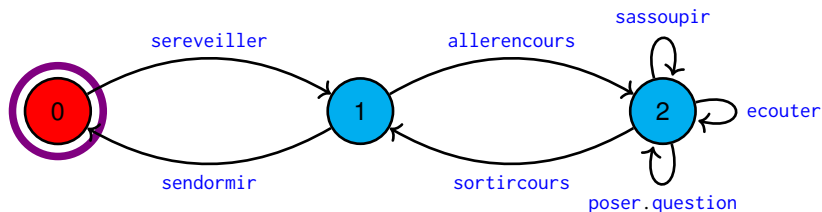
$\text{sink}(2) = \text{true}$

Quelle est la sémantique d'un LTS ?

- ▶ Comportement d'un LTS = ensemble de ses traces
- ▶ Une **trace** de (S, A, T, i) : suite d'actions $\langle a_n \rangle_{n \in \mathbb{N}}$ telle que
 - ▶ $N = \begin{cases} \mathbb{N} & (\text{trace infinie}) \\ \llbracket 0, k \rrbracket & \text{pour } k \in \mathbb{N} \quad (\text{trace finie}) \end{cases}$
 - ▶ il existe une suite d'états de S $\langle s_n \rangle_{n \in \mathbb{N}}$ avec $s_0 = i$ et $\forall n \in N, s_n \xrightarrow{a_n} s_{n+1} \in T$
- ▶ pour \mathcal{H}
 - ▶ sont des traces : $\langle \text{start} \rangle$, $\langle \text{start}, \text{stop} \rangle$, $\langle \text{start}, \text{tick} \rangle$ ou $\langle \text{start}, \text{tick}, \dots, \text{tick}, \text{stop} \rangle$
 - ▶ pas des traces : $\langle \text{tick} \rangle$, $\langle \text{start}, \text{to} \rangle$, $\langle \text{stop}, \text{start}, \text{tick} \rangle$

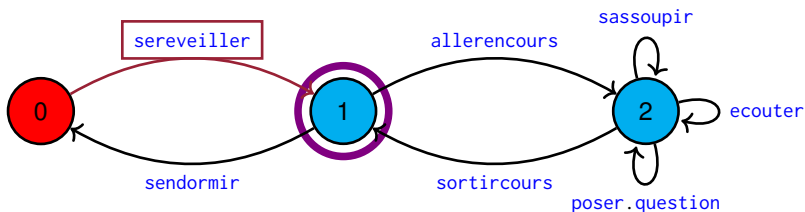
Animer un modèle

- ▶ Suivre une trace, c'est **animer** un modèle
- ▶ Une trace



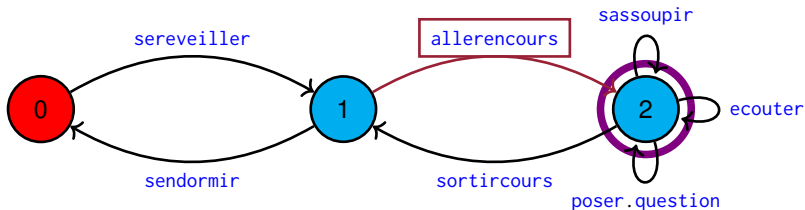
Animer un modèle

- ▶ Suivre une trace, c'est **animer** un modèle
- ▶ Une trace
 - ▶ **sereveiller**



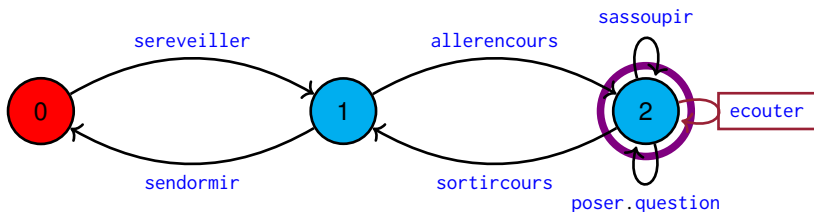
Animer un modèle

- ▶ Suivre une trace, c'est **animer** un modèle
- ▶ Une trace
 - ▶ **sereveiller**, **allerencours**



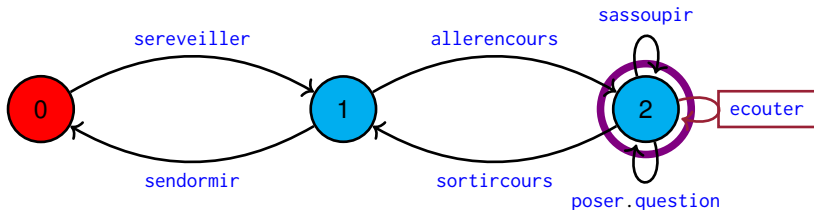
Animer un modèle

- ▶ Suivre une trace, c'est **animer** un modèle
- ▶ Une trace
 - ▶ **sereveiller**, **allerencours**, **ecouter**



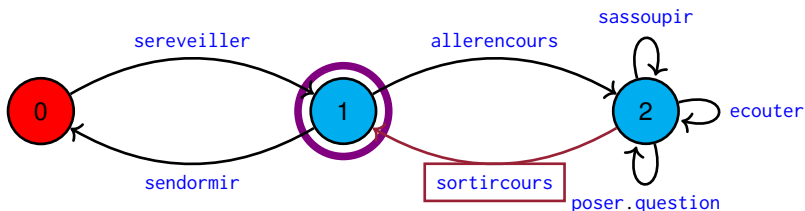
Animer un modèle

- ▶ Suivre une trace, c'est **animer** un modèle
- ▶ Une trace
 - ▶ **sereveiller**, **allerencours**, **ecouter**, **ecouter**



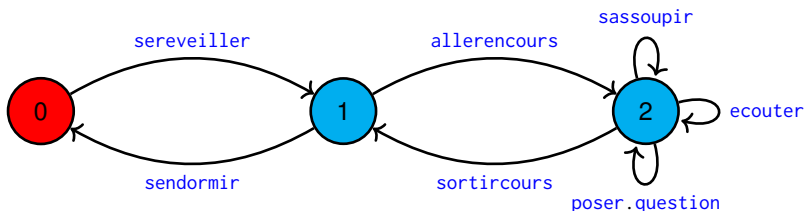
Animer un modèle

- ▶ Suivre une trace, c'est **animer** un modèle
- ▶ Une trace
 - ▶ **sereveiller**, **allerencours**, **ecouter**, **ecouter**,
sortircours

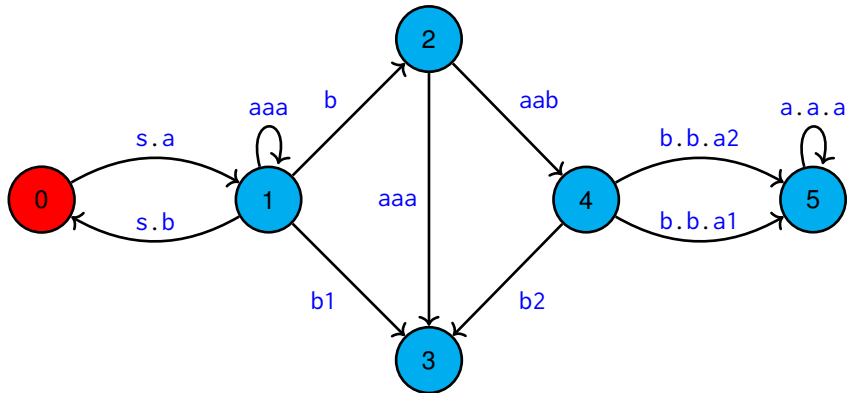


Animer un modèle

- ▶ Suivre une trace, c'est **animer** un modèle
- ▶ Une trace
 - ▶ sereveiller, allerencours, ecouter, ecouter
sortircours, ...



Un exercice



- Des traces ?
- Des états bloqués ou puits ?

Avancement

1 Modéliser les processus

2 FSP, une syntaxe algébrique

3 Composition de processus

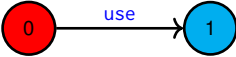
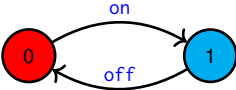
Et pour un processus plus complexe ?

- ▶ Si le processus contient
 - ▶ beaucoup d'états
 - ▶ beaucoup de transitions
 - ▶ un grand nombre de transitions de ou à partir d'un état
- ▶ La représentation graphique devient difficile à construire et à manipuler

⇒ FSP, un langage

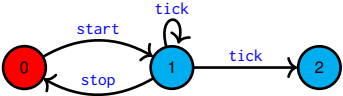
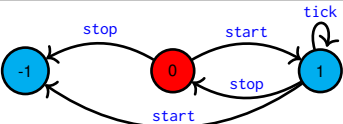
- ▶ pour décrire des processus
- ▶ est interprété comme un automate (LTS)
- ▶ équivalence entre un programme FSP et son LTS

Des premiers exemples de FSP I

FSP	LTS
$\text{ONESHOT} = (\text{use} \rightarrow \text{STOP}).$	 <pre>graph LR; 0((0)) -- use --> 1((1))</pre>
$\begin{aligned} \text{SYST} &= \text{OFF}, \\ \text{OFF} &= (\text{on} \rightarrow \text{ON}), \\ \text{ON} &= (\text{off} \rightarrow \text{OFF}). \end{aligned}$	 <pre>graph LR; 0((0)) -- on --> 1((1)); 1 -- off --> 0</pre>

- ▶ Processus en majuscule, seul nom public, définition terminée par ".", **STOP** état bloqué prédéfini
- ▶ Actions en minuscule et transition par " \rightarrow "
- ▶ Définitions *récurives* d'états en majuscule, séparées par ",", "

Des premiers exemples de FSP II

FSP	LTS
$H = (\text{start} \rightarrow ON),$ $ON = (\text{stop} \rightarrow H \mid \text{tick} \rightarrow ON \\ \mid \text{tick} \rightarrow STOP).$	
$H = (\text{start} \rightarrow ON \mid \text{stop} \rightarrow \text{ERROR}),$ $ON = (\text{stop} \rightarrow H \mid \text{tick} \rightarrow ON \\ \mid \text{start} \rightarrow K).$	

- ▶ Choix par "|"
- ▶ État prédéfini **ERROR** bloqué (-1 dans LTS)
- ▶ État non défini est égal à **ERROR**

Un exercice

- ▶ Communication non fiable
 - ▶ Modéliser un canal de communication qui accepte des actions **in**
 - ▶ Si une faute se produit, il ne génère pas de sortie, sinon il génère une action **out**

Un exercice

- ▶ Communication non fiable
 - ▶ Modéliser un canal de communication qui accepte des actions `in`
 - ▶ Si une faute se produit, il ne génère pas de sortie, sinon il génère une action `out`
- ▶ Utiliser le choix (**non déterminisme**)
 - ▶ `CHAN = (in -> CHAN | in -> OUT),`
`OUT = (out -> CHAN).`

ou de manière équivalente

- ▶ `CHAN = (in -> CHAN`
`| in -> out -> CHAN).`

Avancement

1 Modéliser les processus

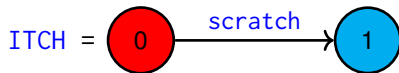
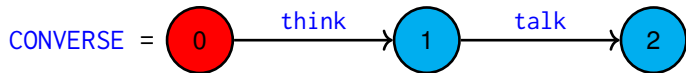
2 FSP, une syntaxe algébrique

3 Composition de processus

Composition parallèle

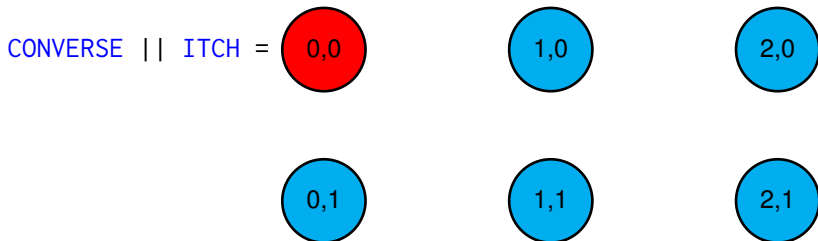
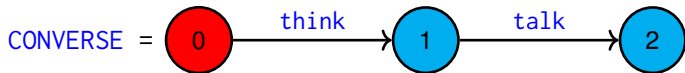
- ▶ $||$ **NOMCOMPOSITION** = $(P1 \ || \ \dots \ || \ Pn)$
 - ▶ commutatif et associatif,
 - ▶ élément neutre **STOP**, élément absorbant **ERROR**
- ▶ Sémantique intuitive (composition **asynchrone**)
 - ▶ exécution simultanée des actions de même nom
 - ▶ 2 actions indépendantes ne s'exécutent pas simultanément
 - ▶ tous les entrelacements possibles des actions indépendantes de tous les processus
- ▶ Sémantique formelle (LTS de la composition)
 - ▶ $S = S_1 \times S_2$ $A = A_1 \cup A_2$ $i = (i_1, i_2)$
 - ▶ $T = \{(s_1, s_2) \xrightarrow{a} (s'_1, s'_2) \mid s_1 \xrightarrow{a} s'_1 \wedge s_2 \xrightarrow{a} s'_2 \wedge a \in A_1 \cap A_2 \vee$
 $s_1 = s'_1 \wedge s_2 \xrightarrow{a} s'_2 \wedge a \notin A_1 \vee s_2 = s'_2 \wedge s_1 \xrightarrow{a} s'_1 \wedge a \notin A_2\}$
 - ▶ en général, on élimine les états non atteignables

Un premier exemple

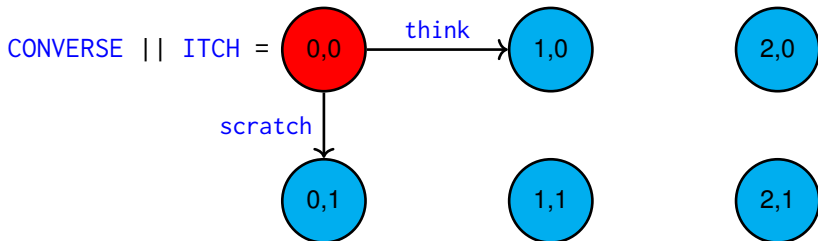
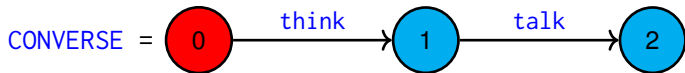


CONVERSE || ITCH =

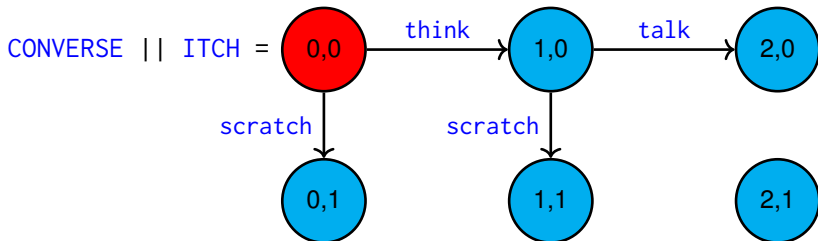
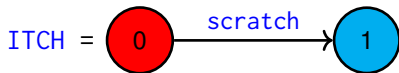
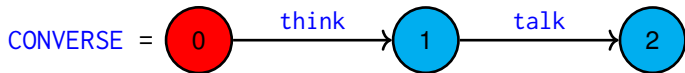
Un premier exemple



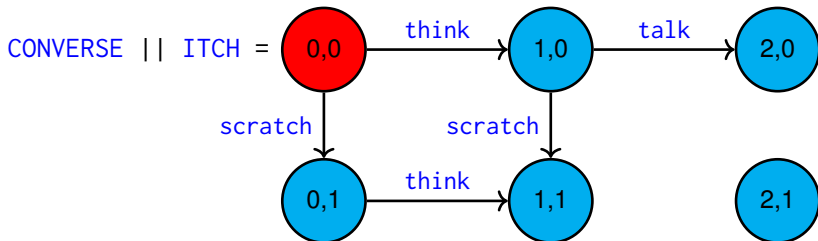
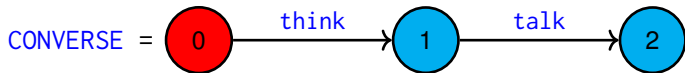
Un premier exemple



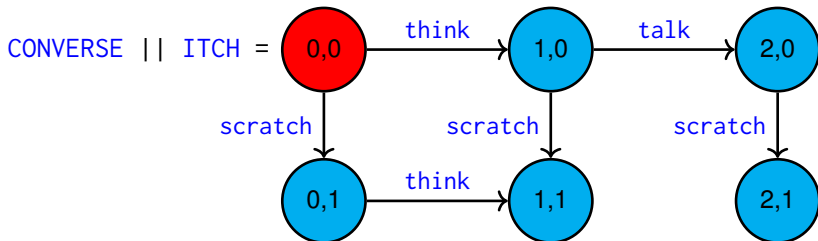
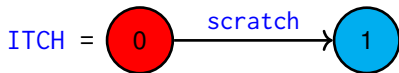
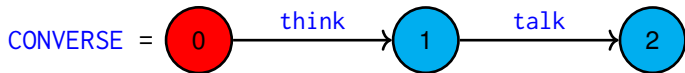
Un premier exemple



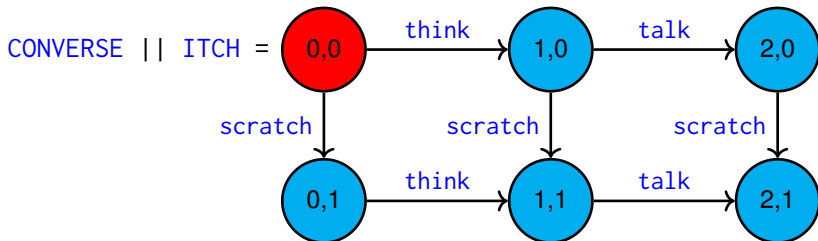
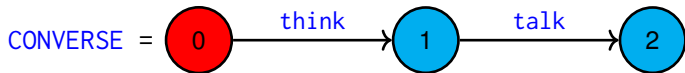
Un premier exemple



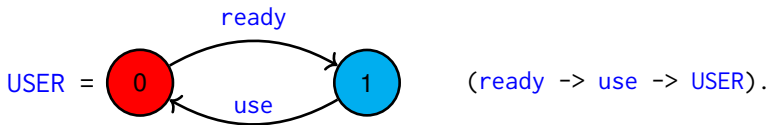
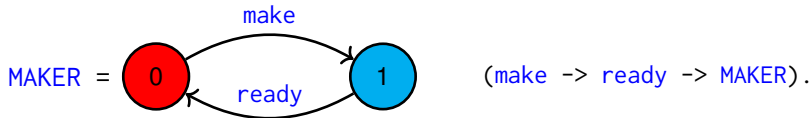
Un premier exemple



Un premier exemple

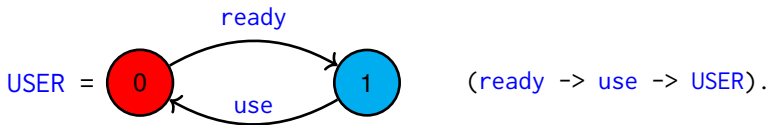
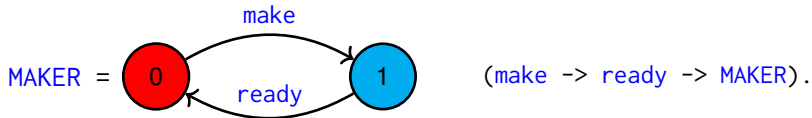


Un second exemple, action partagée



MAKER || USER =

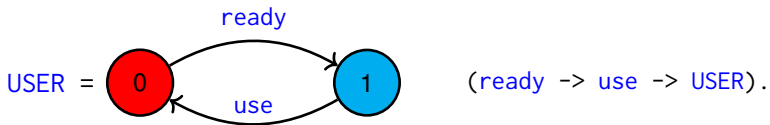
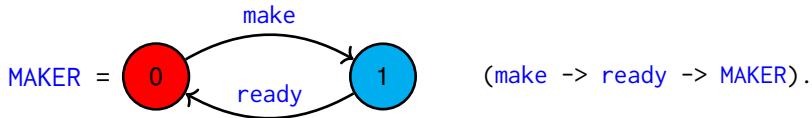
Un second exemple, action partagée



MAKER || USER =



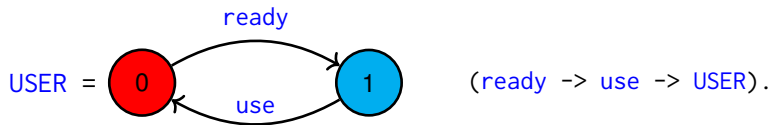
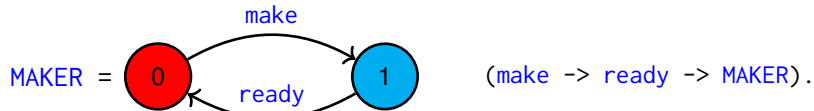
Un second exemple, action partagée



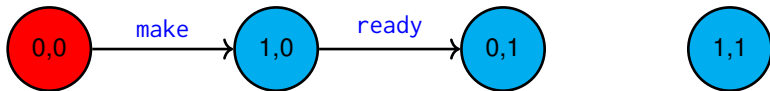
MAKER || USER =



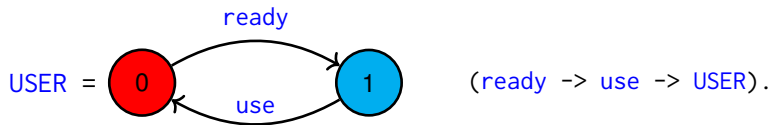
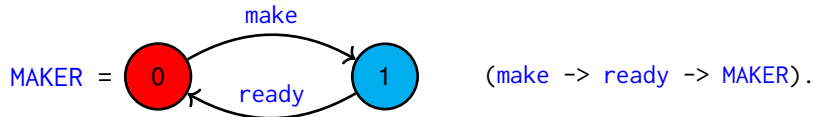
Un second exemple, action partagée



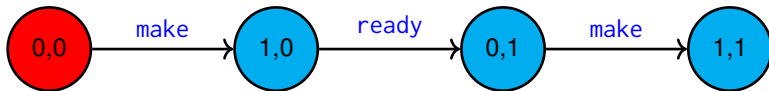
MAKER || USER =



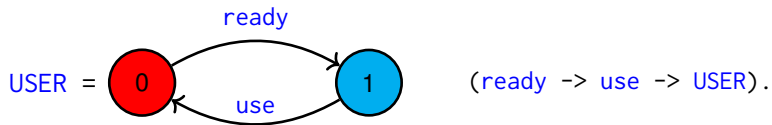
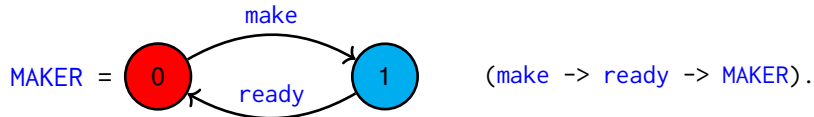
Un second exemple, action partagée



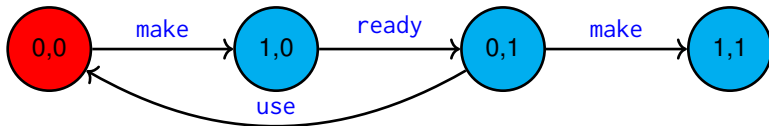
MAKER || USER =



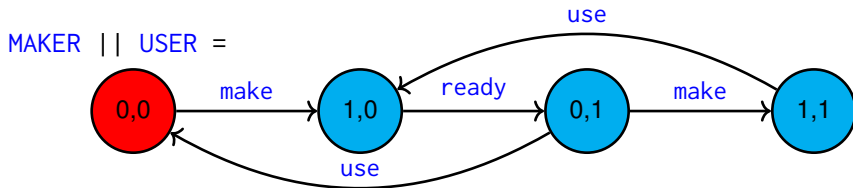
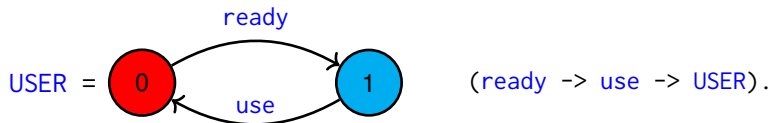
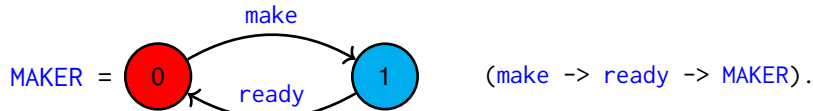
Un second exemple, action partagée



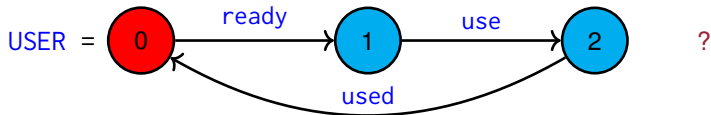
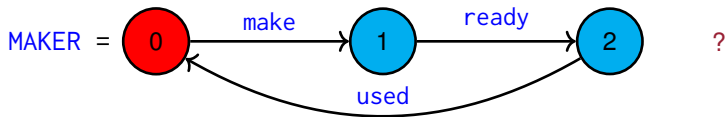
MAKER || USER =



Un second exemple, action partagée

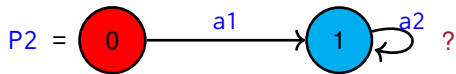
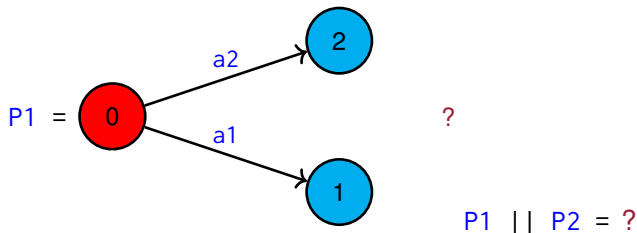


Un exercice, la poignée de main



MAKER || USER = ?

Composition et choix



- La synchronisation permet de sélectionner de l'extérieur une branche d'un choix

La notion d'instance

► Soit `SWITCH = (on -> off -> SWITCH)`.

► `SWITCH || SWITCH` \equiv ?

La notion d'instance

► Soit `SWITCH = (on -> off -> SWITCH)`.

► `SWITCH || SWITCH ≡ SWITCH`

La notion d'instance

- ▶ Soit `SWITCH = (on -> off -> SWITCH)`.
- ▶ `SWITCH || SWITCH ≡ SWITCH`
- ▶ Comment modéliser deux interrupteurs ?

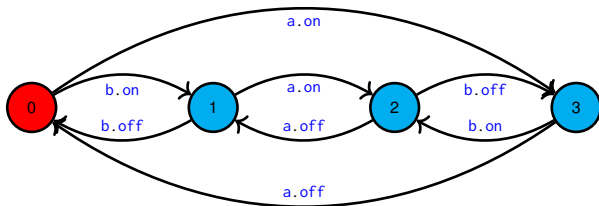
La notion d'instance

- ▶ Soit $\text{SWITCH} = (\text{on} \rightarrow \text{off} \rightarrow \text{SWITCH})$.
- ▶ $\text{SWITCH} \parallel \text{SWITCH} \equiv \text{SWITCH}$
- ▶ Comment modéliser deux interrupteurs ?
- ▶ Notion d'instance, $a : \text{SWITCH}$
 - ▶ Ajoute a en préfixe des actions de SWITCH
 - ▶ $\equiv \text{ASW}$ avec $\text{ASW} = (a.\text{on} \rightarrow a.\text{off} \rightarrow \text{ASW})$

La notion d'instance

- ▶ Soit $\text{SWITCH} = (\text{on} \rightarrow \text{off} \rightarrow \text{SWITCH})$.
- ▶ $\text{SWITCH} \parallel \text{SWITCH} \equiv \text{SWITCH}$
- ▶ Comment modéliser deux interrupteurs ?
- ▶ Notion d'instance, $a:\text{SWITCH}$
 - ▶ Ajoute a en préfixe des actions de SWITCH
 - ▶ $\equiv \text{ASW}$ avec $\text{ASW} = (a.\text{on} \rightarrow a.\text{off} \rightarrow \text{ASW})$

- ▶ LTS de $a:\text{SWITCH} \parallel b:\text{SWITCH}$?



Renommage

- ▶ On peut renommer une (préfixe d') action
 - ▶ $P/\{\text{remplaçant}, \text{remplacé}\}$
 - ▶ $(\text{call} \rightarrow P)/\{\text{request}/\text{call}\} \equiv (\text{request} \rightarrow P)$
 - ▶ $(\text{call.a} \rightarrow \text{call.b} \rightarrow P)/\{\text{request}/\text{call}\} \equiv (\text{request.a} \rightarrow \text{request.b} \rightarrow P)$
 - ▶ $(\text{call.a} \rightarrow a \rightarrow P)/\{b/a\} \equiv (\text{call.a} \rightarrow b \rightarrow P)$
 - ▶ $(\text{call.a} \rightarrow \text{call.b} \rightarrow P)/\{r/\text{call}, n/\text{call.a}\} \equiv (n \rightarrow r.b \rightarrow P)$
- ▶ Renommages multiples
 - ▶ $P/\{\{a, b\}/\{x, y\}\} \equiv P/\{a/x, a/y, b/x, b/y\}$
- ▶ Utile principalement pour composer des processus

Interaction avec de multiples instances

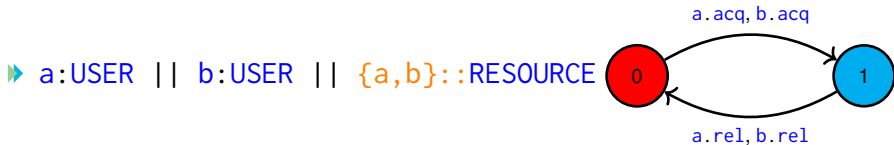
- ▶ Utilisateur : $USER = (acq \rightarrow use \rightarrow rel \rightarrow USER).$
- ▶ Ressource : $RESOURCE = (acq \rightarrow rel \rightarrow RESOURCE).$
- ▶ Comment avoir plusieurs utilisateurs pour une ressource ?
- ▶ $a:USER \parallel b:USER \parallel RESOURCE ?$

Interaction avec de multiples instances

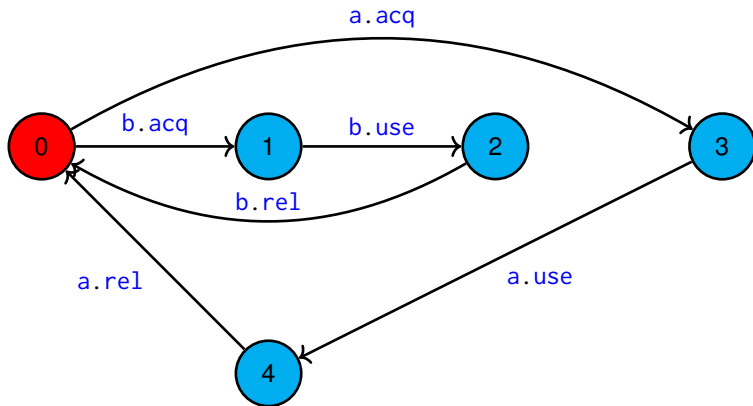
- ▶ Utilisateur : $USER = (acq \rightarrow use \rightarrow rel \rightarrow USER).$
- ▶ Ressource : $RESOURCE = (acq \rightarrow rel \rightarrow RESOURCE).$
- ▶ Comment avoir plusieurs utilisateurs pour une ressource ?
- ▶ $a:USER \parallel b:USER \parallel RESOURCE ?$
- ▶ Non, il faut du renommage

Interaction avec de multiples instances

- ▶ Utilisateur : $USER = (acq \rightarrow use \rightarrow rel \rightarrow USER).$
- ▶ Ressource : $RESOURCE = (acq \rightarrow rel \rightarrow RESOURCE).$
- ▶ Comment avoir plusieurs utilisateurs pour une ressource ?
- ▶ $a:USER \parallel b:USER \parallel RESOURCE$?
- ▶ Non, il faut du renommage



Résultat



- Comment le modèle assure-t-il que l'utilisateur qui acquiert la ressource est celui qui la libère ?

Conclusion

- ▶ Notion de LTS (automate)
- ▶ Syntaxe de base pour définir des processus (FSP)
 - ▶ $P ::= \text{STOP} \mid \text{ERROR} \mid X \mid (a \rightarrow P) \mid P_1 \mid P_2 \mid P_1 \parallel P_2$
- ▶ Sémantique de ce cœur
- ▶ Il faut pratiquer
 - ▶ Construire des modèles
 - ▶ Les valider en les animant (LTSA)