

CM2 La synchronisation

Lundi 20.09.2021

La synchronisation est indispensable pour la coordination entre plusieurs composantes qui s'exécute en parallèle.

On veut le max de parallélisation entre des taches indépendantes, mais, il y a des différents processus qui doit avoir l'accès a des ressources communes, et c'est alors qu'il y aura des conflits.

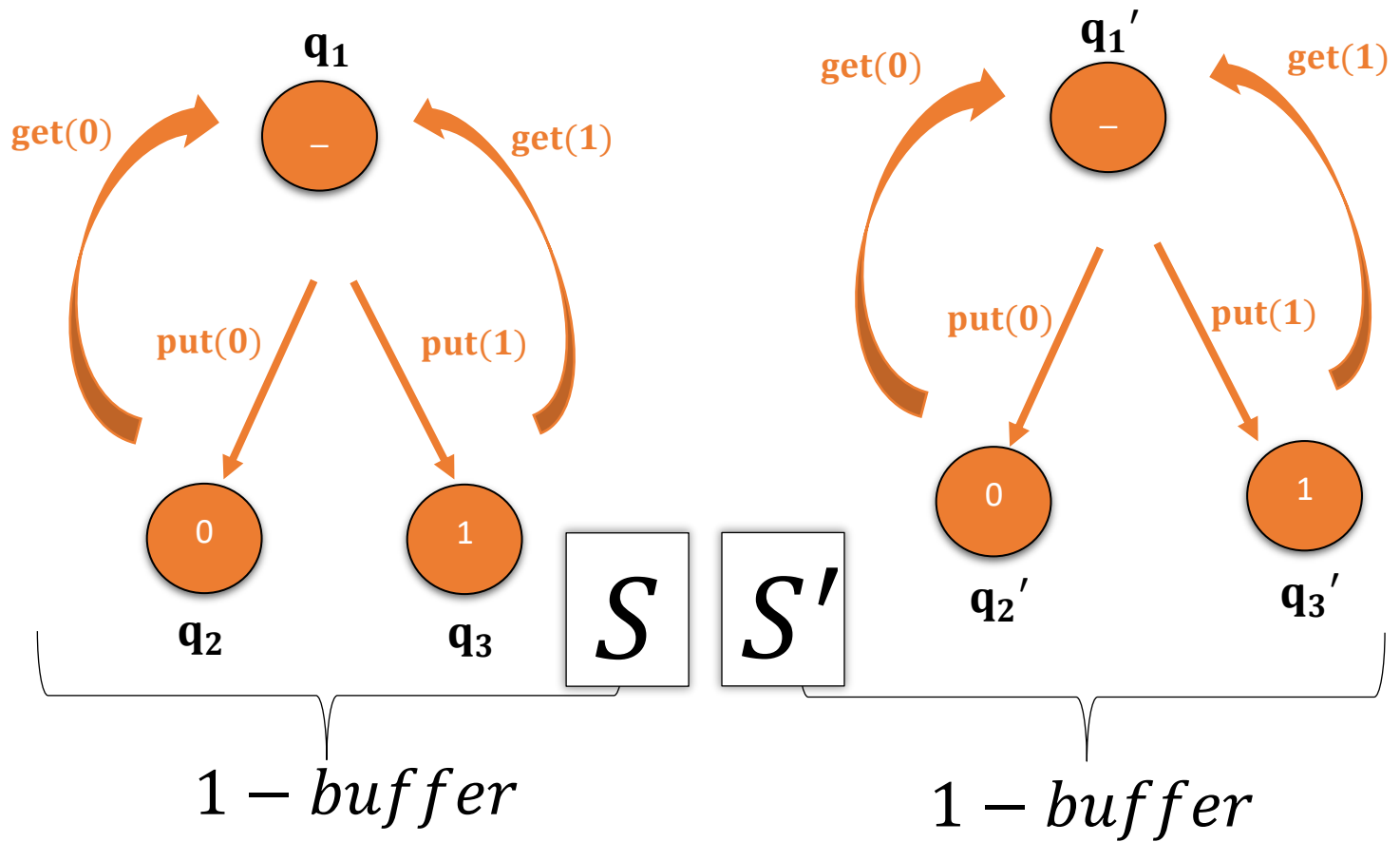
Exemple de la semaine dernière

Exemple avec de la concurrence, mais pas dans le sens qu'on imagine :

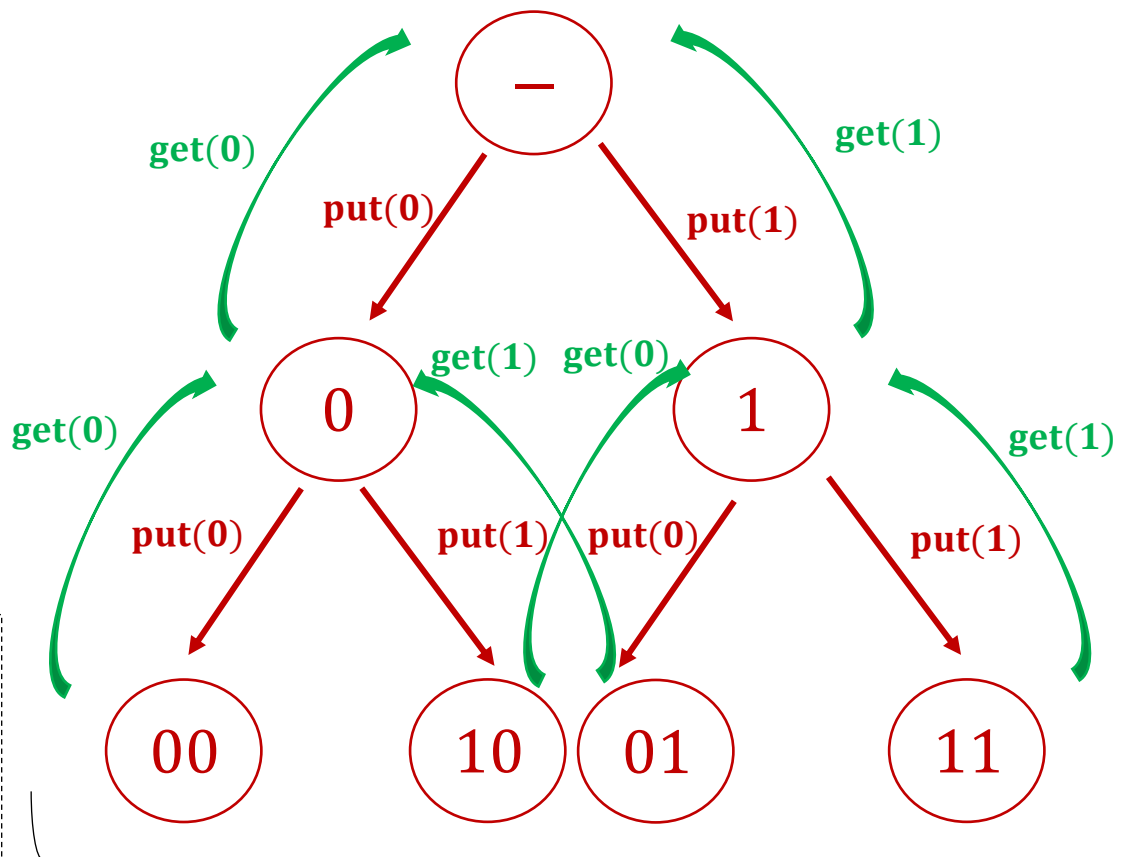
On a 1 buffer a 2 places qu'on voulait crée à partir de 2 buffers a 1 place. Si on les pose ensemble sans aucune coordination entre eux, il n'y a pas d'ordre FIFO comme on voulait.

On va reprendre l'exemple de la semaine dernier et ensuite on va formaliser de manière plus précise ces concepts de synchronisation.

*On a des systèmes a 1 place, il faut préciser ce qui rentre et ce qui sort.
Voici donc 2 copies de ce système qu'on va mettre en parallèle :*



Séquence d'actions possibles :

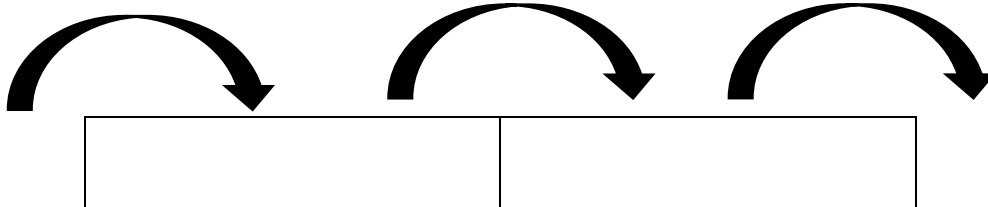


On a des
“get” que
quand cela
respecte
la
politique
FIFO

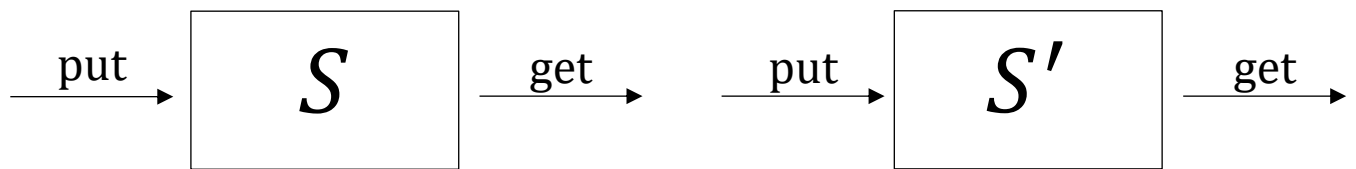
2 – buffer avec une politique $FIFO_2$

Mtn, on veut mettre notre 2 machines a 1 buffer en parallèle de manière qu'on obtient le diagramme ci-dessus. C'est là qu'on introduit l'idée de la synchronisation.

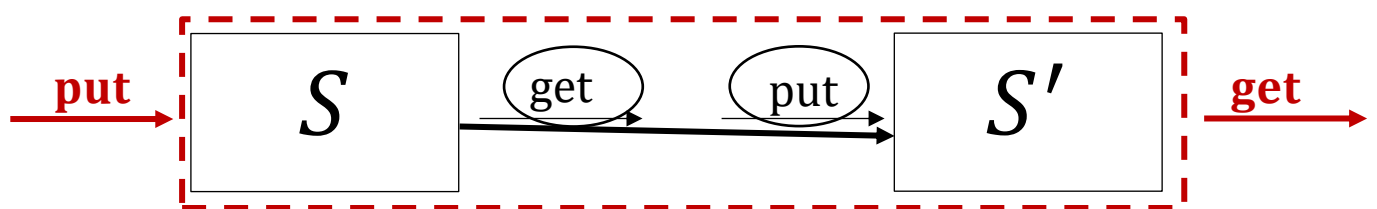
Dans une politique FIFO, on va décaler à chaque fois les valeurs :



Ce décalage nécessite une communication entre les 2 machines :



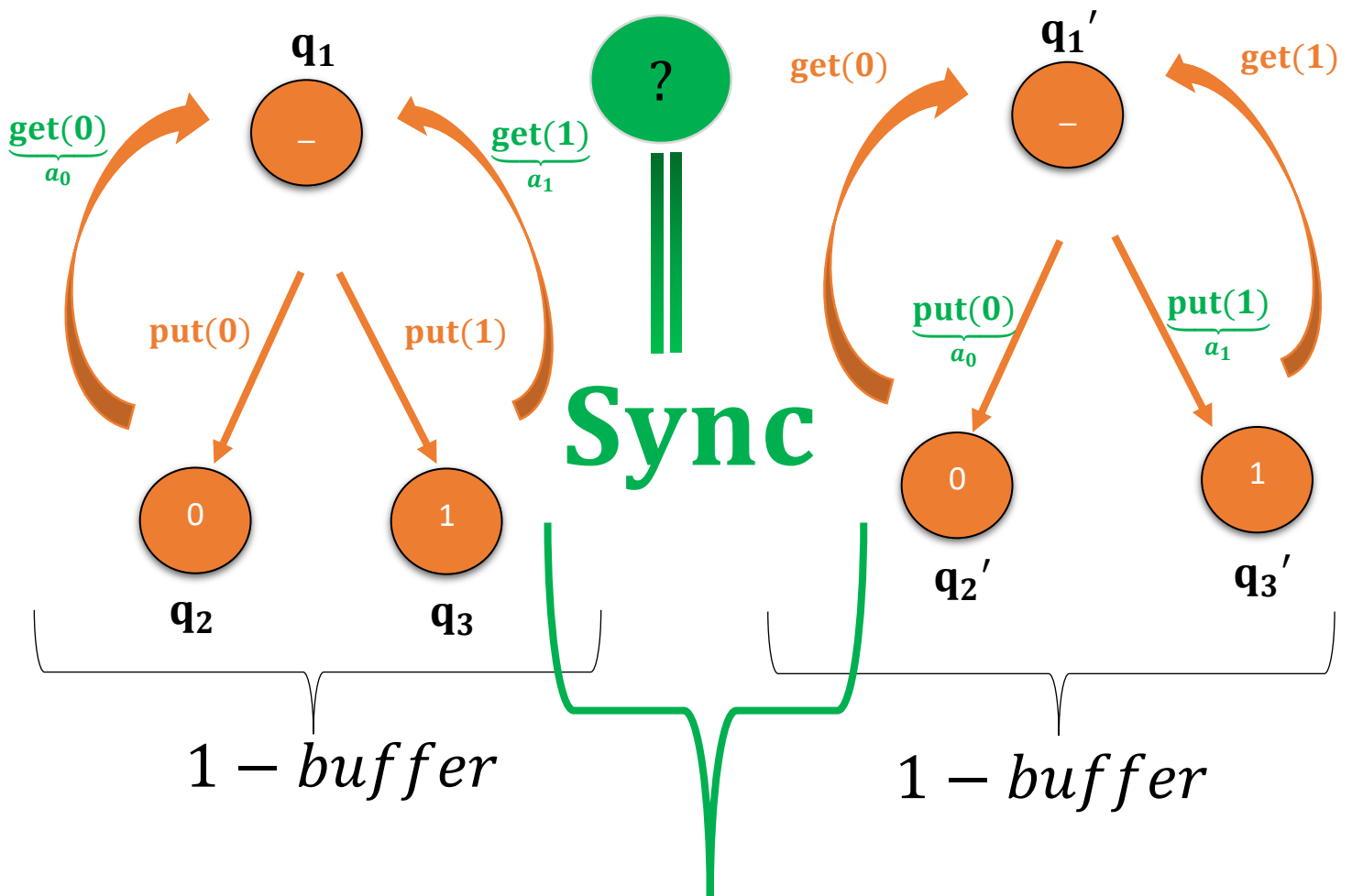
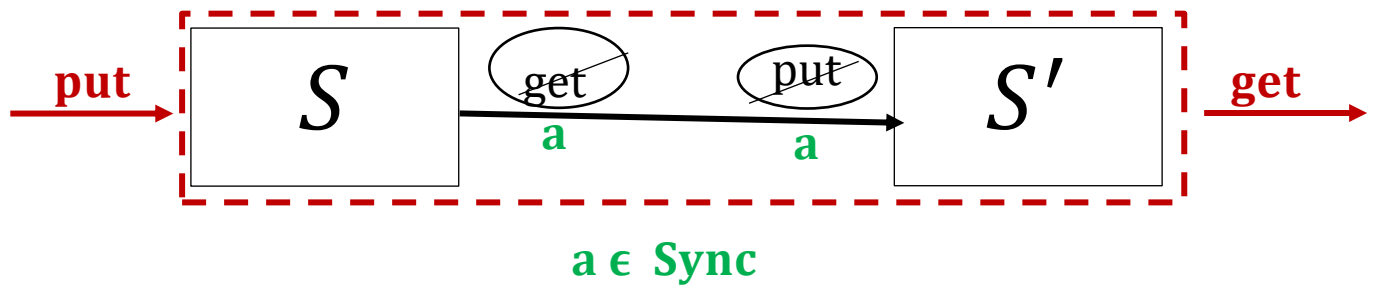
Ce qu'on va faire :



Les actions en rouge :

Actions indépendante avec l'extérieur.

Il va falloir crée un canal qui va créer la porte de sortie du buffer1 avec l'entrée au buffer2



Les actions de synchronizations $\{a(0), a(1)\}$

get(1) se passe en même temps que put(1)

Composition parallèle

Etats Actions Relations de transitions

$$S = (Q, A, \delta) \quad S' = (Q', A', \delta')$$

$$\delta \subseteq Q \times A \times Q$$

$$\delta' \subseteq Q' \times A \times Q'$$

$$\text{Sync} \subseteq A$$

Inclus

L'ensemble des actions est le même

L'ensemble des actions de synchro

$$S \parallel S' = (Q \times Q', A, \delta_{||})$$

Sync

$Q_{||}$

$$\delta_{||} \subseteq Q_{||} \times A \times Q_{||} \quad \text{telle que:}$$

Action a

$$q_1 \xrightarrow{a}_{\delta} q_2 \quad \text{et } a \notin \text{Sync}$$

Je fais une action "a" de q_1 vers q_2 ,
cette action est asynchrone.

$$\forall q_1' \in Q', \quad (q_1, q_1') \xrightarrow{a}_{\delta_{||}} (q_2, q_1')$$

Non connotation :


$$q_1 \xrightarrow{a}_{\delta} q_2 \text{ pour } (q_1, a, q_2) \notin \delta$$

$c \cdot - \hat{a} - d \cdot$: quel que soit l'état de l'autre machine, la machine 1 et la machine 2 vont bouger, mais, en effet, c'est que la machine 1 qui change d'état, pas la machine 2 :

$$(q_1, q_1') \dots (q_2, q_1')$$

L'état global a changé, mais qu'une composante a changé -> action asynchrone.

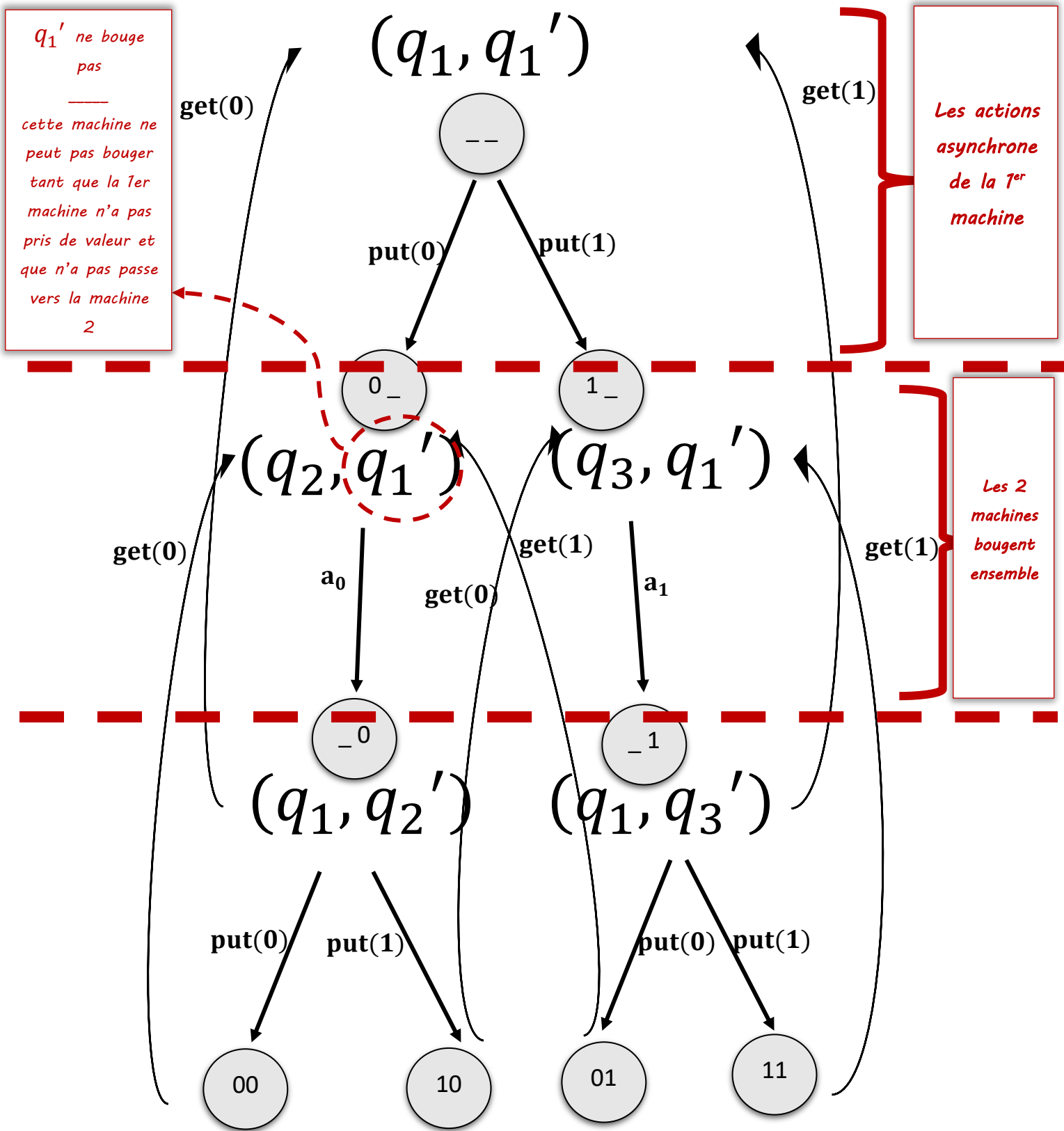
<i>Cas 1</i>	<i>Cas 2</i>	<i>Cas 3</i>
$\begin{array}{c} a \\ q_1 \rightarrow q_2 \text{ et } a \notin \text{Sync} \\ \delta \end{array}$	$\begin{array}{c} a \\ q_1' \rightarrow q_2' \text{ et } a \notin \text{Sync} \\ \delta \end{array}$	$\begin{array}{c} a \qquad a \\ q_1 \rightarrow q_2 \text{ et } q_1' \rightarrow q_2' \\ \delta \qquad \delta \\ \text{et } a \in \text{Sync} \end{array}$
$\forall q_1' \in Q',$ $\begin{array}{c} a \\ (q_1, q_1') \rightarrow (q_2, q_1') \\ \delta_{ } \end{array}$ <p><i>Action asynchrone de S</i></p>	$\forall q_1 \in Q,$ $\begin{array}{c} a \\ (q_1, q_1') \rightarrow (q_1, q_2') \\ \delta_{ } \end{array}$ <p><i>Action asynchrone de S'</i></p>	$\begin{array}{c} a \\ (q_1, q_2') \rightarrow (q_2, q_2') \\ \delta_{ } \end{array}$ <p><i>Action synchrone de S et S' (ensemble)</i></p>



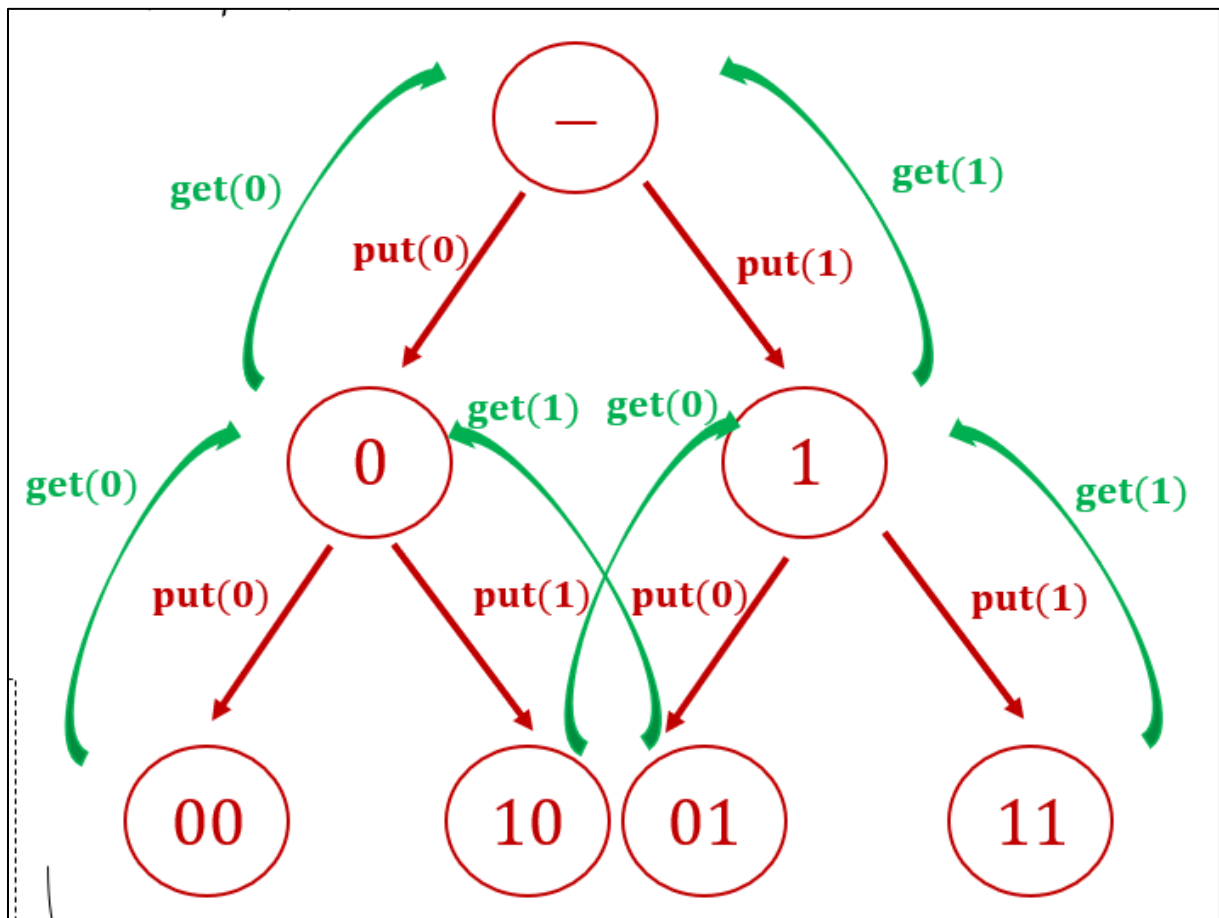
*Les 2 composantes ont
changé simultanément*

Donc,

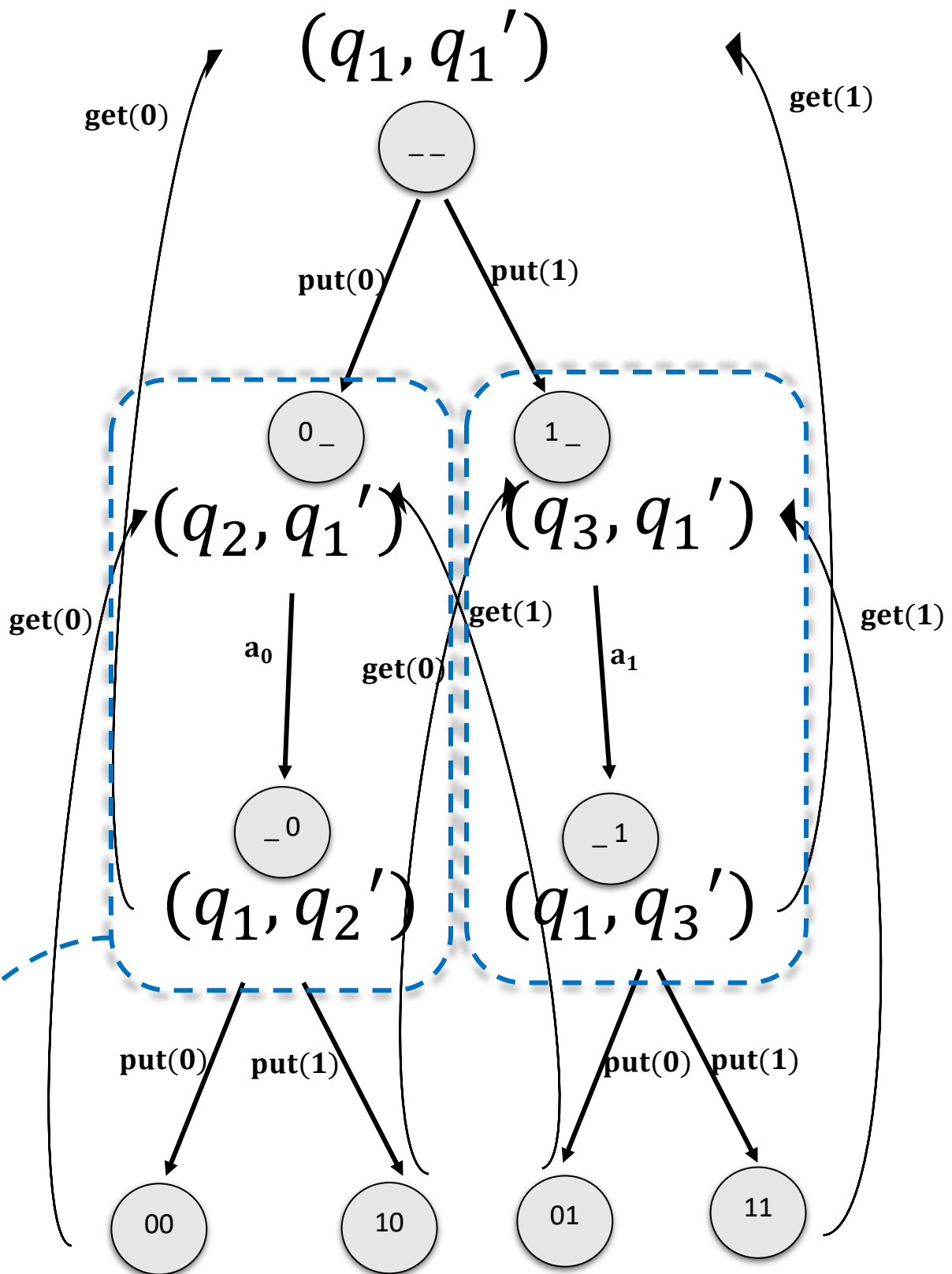
c'est ça la définition formelle comment on peut bouger : soit on est asynchrone et chaque un peut bouger seul, soit on est synchrone et alors on peut manger que ensemble (cas général, pas spécial a notre exemple).



Ce diagramme est pareil que le diagramme suivant ?



Oui, si on unit les états alors ces 2 machines sont équivalentes :



On change d'état mais ce n'est pas visible de l'extérieur

Notion d'équivalences pour faire des comparaisons

Il y a différentes notions d'équivalences qu'on peut avoir pour comparer des systèmes.

On va définir l'équivalence tel qu'elle est dans notre exemple (on regarde que ce qu'est visible à l'extérieur).

Notion 1 : équivalence de traces

Si j'ai un system de transition S :

$$S = (Q, A, \delta)$$

Etant donnée un état q , $q \in Q$, une séquence d'exécution de S à partir de q (une séquence d'alternance état-action) :

$q_0 a_0 q_1 a_1 q_0 \dots etc$

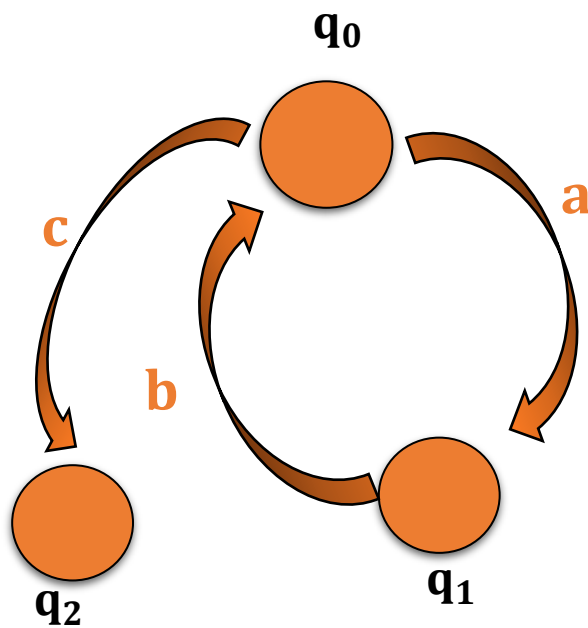
$$tq \left\{ \begin{array}{l} a_0 = q \\ q_i \in Q, \forall i \\ a_i \in A, \forall i \\ (q_i, a_i, q_{i+2}) \in \delta \forall i \end{array} \right.$$

2 cas :

(1) *Infinie*

(2) Finie de la forme $q_0 \dots q_n$
a

avec $q_n \nrightarrow (\nexists a, \nexists q, q_n \rightarrow_\delta q)$



(1) *Infinie* : $q_0 a q_1 b q_0 a q_1 b q_0 \dots$

(2) Finie de la forme : $q_0 a q_1 b q_0 c q_2$

On veut observer la trace de l'exécution

Pour une séquence d'exécution, la trace est la séquence des actions :

$$a_0 a_1 a_2 \dots$$

Cette trace peut être finie ou infinie selon si la séquence d'exécution est finie ou infinie.

Traces(q, S) : ensemble de traces de S à partir de q

Equivalence des traces

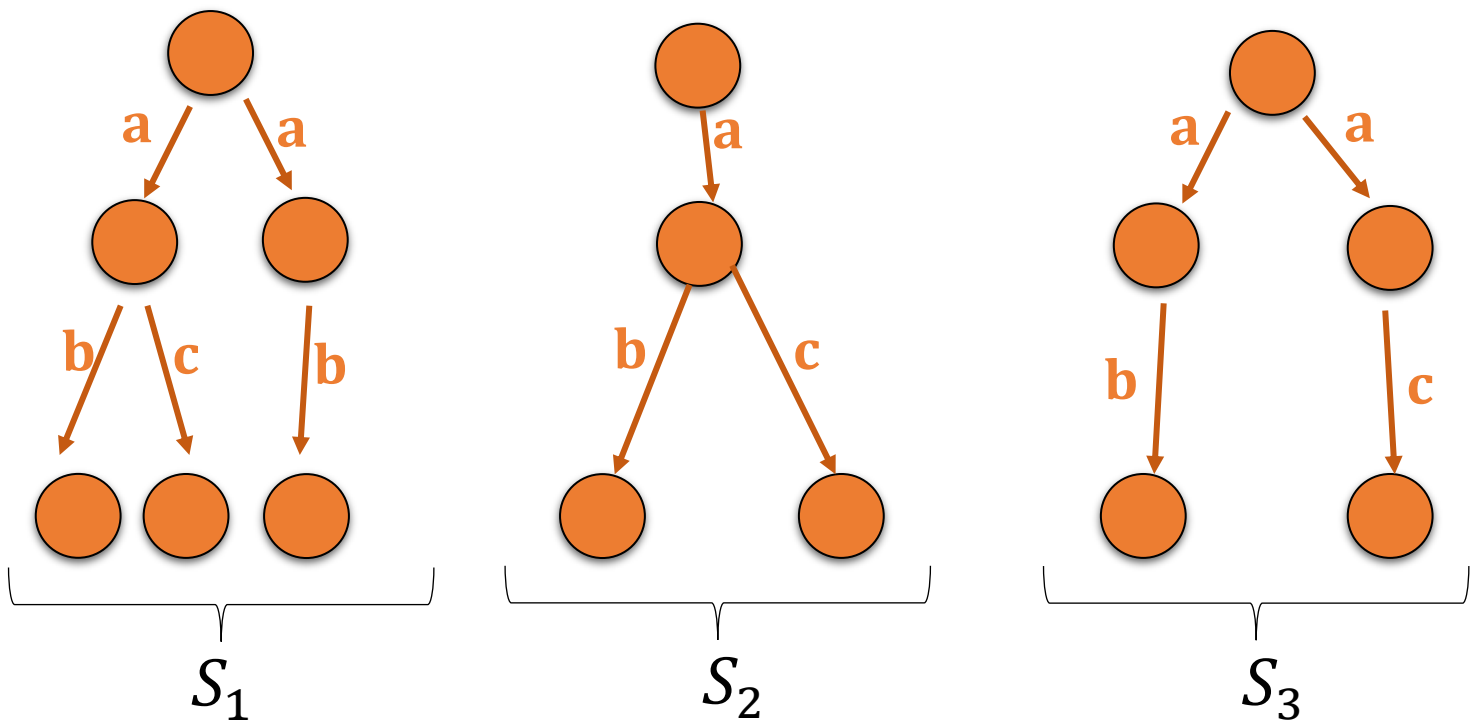
2 états sont équivalents s'ils ont le même ensemble de traces

$$q \rightarrow_t q' \text{ SSi } \text{Traces}(q, S) = \text{Traces}(q', S)$$

Exemple

On va prendre 3 systèmes.

Ils sont pareil ou pas ?



Selon la définition, la question est : les ensembles de traces des exécutions sont les mêmes ou pas ?

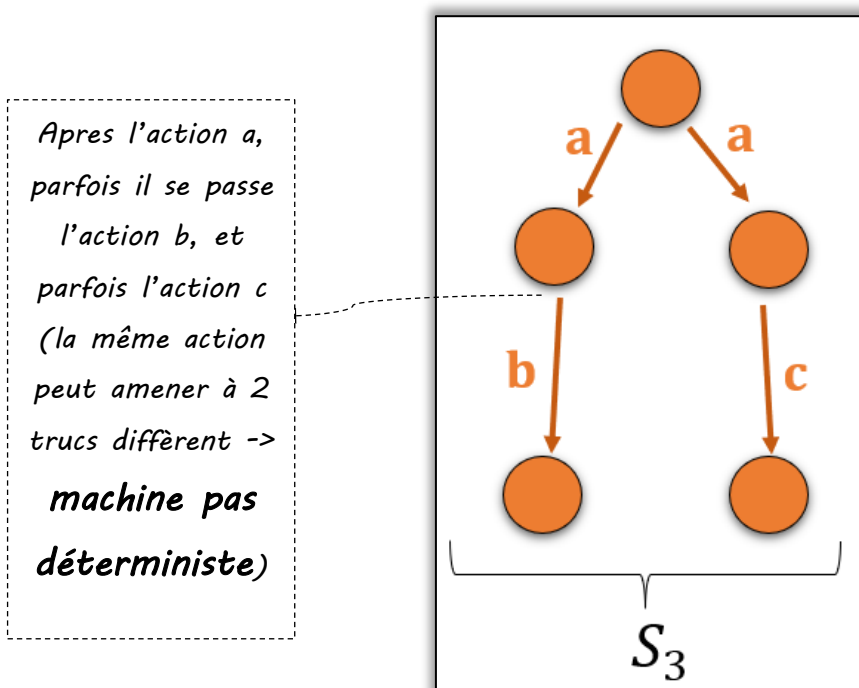
Réponse : oui, les ensembles de traces des exécutions sont les mêmes

Programmes non-déterministes

Si j'ai une machine, par exemple : une machine à café, avec les actions suivantes :

- (a) J'introduis de l'argent dans la machine
- (b) ..
- (c) Je récupère le café

Donc, si on nous décrit cette machine avec $S1$ / $S2$ / $S3$, les 3 systèmes sont pareil ou pas ?



Donc, avec la concurrence, on peut obtenir un programme non-déterministe, ce n'est pas quelque chose de rare car on n'arrive pas toujours à maîtriser tout ce qui se passe.

Quand on a 2 machines qui s'exécutent en parallèle, on ne peut pas savoir si l'action X va se passer avant l'action Y ou l'inverse.

Donc, en concurrence il y a du non-déterminisme, mais, pour certaines applications, dans le cas de certaines propriétés, la capacité d'interagir à tout moment est important, est alors regarder que les traces ne suffisent pas.

Notion 2 : relation de bisimulation

La relation $R : R \subseteq Q \times Q$

La relation R est une relation de dissimulation *SSI* :

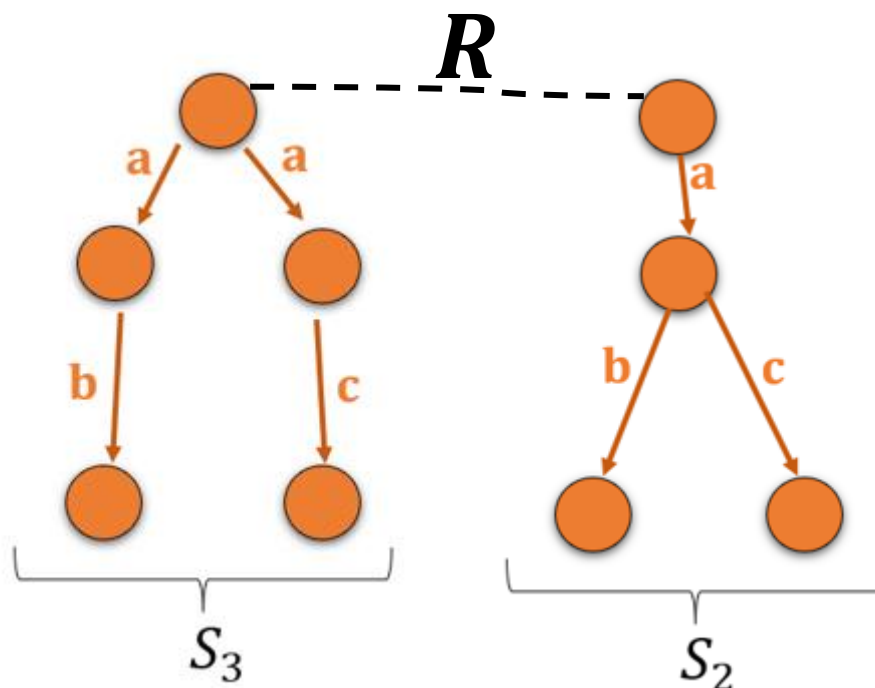
➤ R est symétrique (X en relation avec Y , et Y en relation avec X)

➤ $\forall q_1, q_2 \quad q_1 R q_2 \text{ ssi } \forall a \in A, \forall q_1' \in Q$

$$\begin{array}{c} a \\ q_1 \xrightarrow{\delta} q_1' \end{array} \Rightarrow \begin{array}{c} a \\ \exists q_2' \quad q_2 \xrightarrow{\delta} q_2' \end{array} \text{ et } q_1' R q_2'$$

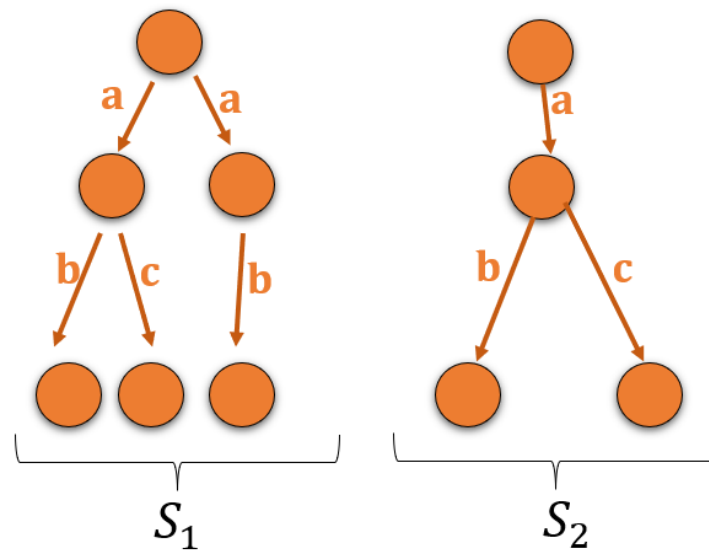
Exemple

Il y a une relation de bisimulations qui relie l'état initial de S_3 avec l'état initial de S_2 ?



Réponse : ce n'est pas possible car si ils sont en relation on peut pas satisfaire les contraintes (il avances pas de la même manière, si S_3 va a « b », S_2 va a « c »).

S_1 et S_2 sont bisimilaire ? Non !



Supposons que les états initiaux sont bisimilaire,

-> Si S_2 commence à jouer, S_1 peut répondre.

Mais il faut que ça soit symétrique, que peu importe qui commence, ils pourront avancer de manière équivalente, faut que chaque action de l'un permette à l'autre d'arriver à l'équivalence.

Cette relation est donc plus forte qu'une relation de traces.

$$q \rightarrow_{\delta} q' \text{ ssi } \exists R \text{ une relation de bisimulation tq } qRq'$$

- Les traces : trop faible comme relation.
- La bisimulation : très forte.
- Existe-t-elle une relation plus faible ?

Relation de simulation (uni-directionnelle)

Notion 3 : relation de simulation

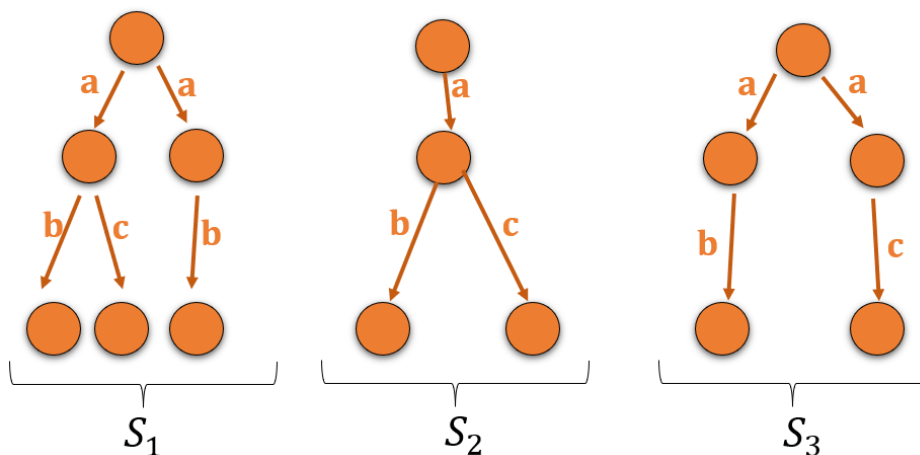
$R \subseteq Q \times Q$ est une relation de simulation ssi

➤ $\forall q_1, q_2 \quad q_1 R q_2 \text{ ssi } \forall a \in A, \forall q_1' \in Q$

$\begin{matrix} a \\ q_1 \end{matrix} \xrightarrow{\delta} \begin{matrix} a \\ q_1' \end{matrix} \Rightarrow \exists q_2' \quad \begin{matrix} a \\ q_2 \end{matrix} \xrightarrow{\delta} q_2' \text{ et } q_2' \quad \begin{matrix} a \\ q_1' \end{matrix} R q_2'$

Def : q' simule q

$q' \sqsubseteq q$ ssi $\exists R$ une relation de simulation $q R q'$



$\begin{matrix} ? \\ S_3 \sqsubseteq_S S_2 : \text{oui} \end{matrix} \quad \begin{matrix} ? \\ S_3 \sqsubseteq_S S_2 : \text{non} \end{matrix}$

Donc, on a la simulation dans un sens, pas l'autre.

$$S_1 \stackrel{?}{\sqsubseteq} S_2 \quad ?$$

A chaque fois que S_2 fait quelque chose, S_1 peut répondre ?

Donc S_1 peut simuler S_2 !

S_2 peut simuler S_1 ? oui.

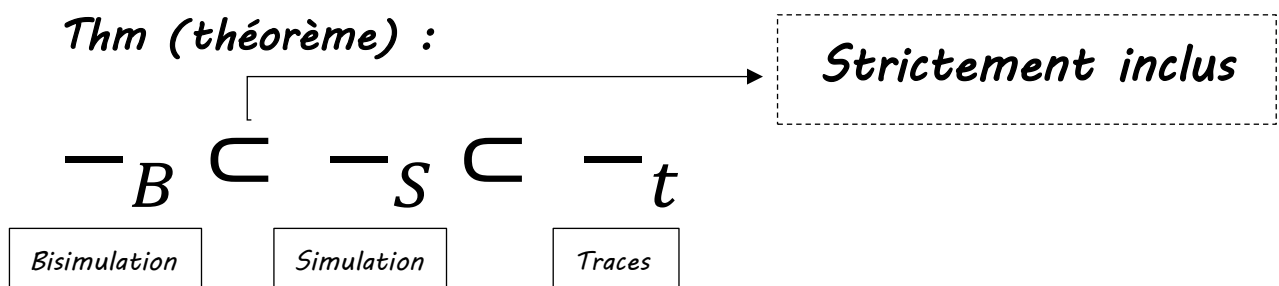
Donc, on a la simulation dans les 2 sens, malgré qu'il sont pas bisimilaire.

Pq ? avec la simulation c'est comme un jeu avec 2 parties : 1 joue, l'autre répond, et après, on inverse les rôles, et si c'est OK pour les 2 cotes, on dit qu'on est équivalent en simulation.

Def :

$$q \sim_S q' \text{ ssi } q \sqsubseteq_S q' \text{ et } q' \sqsubseteq_S q$$

Thm (théorème) :



De manière général, dire qu'un system est équivalent a un autre, ça dépend quelle type de relation on considère (traces, simulation, etc...)