Théorie et pratique de la concurrence – Master 1 II
TD 9 : Modélisation avec CCS

www.liafa.jussieu.fr/~sighirea/cours/concur/

Exercice 1 : *Exclusion mutuelle (TD 6)*

L'algorithme suivant (écrit en Algol) a été publié une année après la publication de l'algorithme de Dekker dans le journal *Comunications of the ACM* :

```
 1 Boolean array b(0;1) integer k, i, j,
 2 comment process i, with i either 0 or 1 and j = 1-i;
 3 C0: b(i) := false;
 4 C1: if k != i then begin
 5 C2: if not (b(j)) then go to C2;
 6  else k := i; go to C1 end;
 7  else critical section;
 8  b(i) := true;
 9  remainder of program;
10  go to C0;
11  end
```

1. Modéliser une variable booléenne comme un processus CCS qui permet de lectures de sa valeur à travers les ports `isTrue` et `isFalse` et des écritures de sa valeur à travers les ports `setTrue` et `setFalse`.
2. Réutiliser le processus ci-dessus pour modéliser la mémoire de cet algorithme, c'est-à-dire le tableau `b` et la variable `k`.
3. Modéliser en CCS le processus `i`. En utilisant cette première définition, construire le processus `j`.
4. Quel terme CCS modélise cet algorithme pour observer que les section critiques et non-critiques des processus ?
5. Comment peut-on vérifier que l'exclusion mutuelle est satisfaite ?

Exercice 2 : *Dîner des philosophes (TD 7)*

La modélisation du dîner des philosophes utilise cinq sémaphores binaires pour modéliser les fourchettes.

1. Modéliser en CCS un sémaphore binaire.
2. Réutiliser le terme CCS ci-dessus pour obtenir un terme CCS pour chaque fourchette.
3. Modéliser en CCS un des philosophes en utilisant également comme actions "mange" et "pense".
4. Réutiliser le terme CCS précédent pour obtenir un terme CCS pour chaque philosophe.
5. Utiliser les termes ci-dessus pour construire le terme du dîner des philosophes afin qu'on observe que les actions "mange" et "pense" de chaque philosophe.

Exercice 3 : *Protocole du bit alterné (TD 5)*

Le protocole du bit alterné (ABP) assure la transmission fiable des messages entre un émetteur (*Sender*) et un récepteur (*Receiver*) qui communiquent par des canaux de communication non fiables. L'émetteur prend des messages à transmettre sur le port `accept` et le récepteur doit les délivrer par le port `deliver`. Un canal `Trans` assure la transmission des messages depuis l'émetteur (par le port `send`) vers le récepteur (port `trans`). Un autre canal `Ack` est dédié à la transmission des acquittements du récepteur (port `reply`) pour l'émetteur (port `ack`). On suppose ici que ces deux canaux ont une capacité d'une place et qu'ils perdent ou dupliquent les messages (sans les corrompre).

Le nom du protocole vient de la méthode utilisé : les messages sont transmis accompagnés alternativement des bits 0 et 1 ; ces bits constituent aussi les acquittements.

L'émetteur fonctionne comme suit. Après avoir reçu un message à transmettre (via `accept`), il l'envoie avec un bit b par le canal `Trans` et il arme un *timer*. Trois possibilités peuvent apparaître :

- il observe un *time-out* du *timer*, dans quel cas il retransmet le message avec le bit b ;
- il reçoit par le canal `Ack` un acquittement b et donc il désarme le *timer* et peut accepter un autre message à transmettre (qui sera transmis avec le bit $\overline{b}$ sur `Trans`) ;
- il reçoit un acquittement $\overline{b}$ (dû à une transmission superflue du message précédent) qu'il ignore.

Le récepteur fonctionne comme suit. Après avoir reçu un message avec le bit b, il envoie son acquittement b sur le canal `Ack` et il arme un *timer*. Trois possibilités peuvent apparaître :

- il observe un *time-out* du *timer* et il retransmet l'acquittement avec le bit b ;
- il reçoit un nouveau message avec le bit $\overline{b}$ par le canal `Trans` et il le délivre sur `deliver` puis il l'acquitte avec le bit $\overline{b}$;
- il reçoit un message superflu avec le bit b qu'il ignore.

1. Modéliser les deux processus de ce protocole, les deux timers et les deux canaux en utilisant les termes CCS.
2. Modéliser le système complet en utilisant CCS afin d'observer uniquement les actions `accept` et `deliver`.
3. Une spécification (propriété) de ce protocole est qu'il se comporte comme un tampon fiable de capacité exactement 1. Modéliser ce type de tampon en CCS. Comment peut-on montrer que cette spécification est satisfaite par notre modèle ?

Exercice 4 : *Canaux de communication*

1. Modéliser un tampon à N places sans passer par un tampon à une place.
2. Utiliser le modèle CCS du tampon à une place afin de modéliser un tampon à $N \geq 2$ places. Pour le moment, la politique de service du tampon n'est pas importante.
3. Le même exercice avec une politique de service *FIFO*.
4. Les deux termes ci-dessus sont-ils bisimilaires ? Si oui, définir la bisimulation, sinon donner un contre-exemple.

Correction 1 : *Exclusion mutuelle*

1. Définition d'un processus modélisant une variable booléenne :

$$\begin{aligned} vB_1 &::= \overline{\texttt{isTrue}}.vB_1 + \texttt{setTrue}.vB_1 + \texttt{setFalse}.vB_0 \\ vB_0 &::= \overline{\texttt{isFalse}}.vB_0 + \texttt{setFalse}.vB_0 + \texttt{setTrue}.vB_1 \end{aligned}$$

2. La mémoire contient trois variables booléennes : on considère que le tableau b est initialisé à 1 et la variable k à 0. Les trois variables sont obtenue par renommage des actions des définitions ci-dessus. Pour indiquer l'opération faite par une action on insère dans son nom un r pour la lecture et un w pour l'écriture. Le bit qui suit la lettre de l'opération indique la valeur lue ou écrite.

$$\begin{aligned} \texttt{Memory} &::= b0 \mid b1 \mid k \\ b0 &::= vB_1[\texttt{b0r0/isFalse}, \texttt{b0r1/isTrue}, \texttt{b0w0/setFalse}, \texttt{b0w1/setTrue}] \\ b1 &::= vB_1[\texttt{b1r0/isFalse}, \texttt{b1r1/isTrue}, \texttt{b1w0/setFalse}, \texttt{b1w1/setTrue}] \\ k &::= vB_0[\texttt{kr0/isFalse}, \texttt{kr1/isTrue}, \texttt{kw0/setFalse}, \texttt{kw1/setTrue}] \end{aligned}$$

3. La définition du processus avec $i = 0$ est la suivante (on utilise plusieurs noms de processus avec le suffixe du nom indiquant la ligne ou l'étiquette dans le code) :

$$\begin{aligned} P0 &::= \overline{\texttt{b0w0}}.P0C1 \\ P0C1 &::= \texttt{kr0}.P07 + \texttt{kr1}.P0C2 \\ P0C2 &::= \texttt{b1r0}.P0C2 + \texttt{b1r1}.\overline{\texttt{kw0}}.P0C1 \\ P07 &::= \texttt{enter0.exit0}.\overline{\texttt{b0w1}}.P0 \end{aligned}$$

Le processus avec $j = 1$ est obtenu par renommage du processus ci-dessus qui tiens compte du comportement symétrique des processus :

$$P1 ::= P0 \left[\begin{array}{l} \texttt{b1w0/b0w0}, \texttt{b1w1/b0w1}, \\ \texttt{b0r0/b1r0}, \texttt{b0r1/b1r1}, \\ \texttt{kr1/kr0}, \texttt{kr0/kr1}, \texttt{kw1/kw0}, \\ \texttt{enter1/enter0}, \texttt{exit1/exit0} \end{array} \right]$$

4. Pour obtenir le modèle CCS de l'algorithme, la mémoire est composée en parallèle avec les deux processus et la synchronisation est forcée en restreignant les actions observable à celles d'entrée/sortie de la section critique :

$$Algo ::= (Memory \mid P0 \mid P1) \backslash \{\texttt{b}i\texttt{r}j \mid i,j \in \{0,1\}\} \cup \{\texttt{b}i\texttt{w}j \mid i,j \in \{0,1\}\} \cup \{\texttt{kr}i, \texttt{kw}i \mid i \in \{0,1\}\}$$

5. L'exclusion mutuelle est modélisé par le terme CCS suivant :

$$CS ::= \texttt{enter1.exit1}.CS + \texttt{enter0.exit0}.CS$$

qui exclue l'entrelacement des actions d'entrée et sortie entre les deux sections critiques. L'algorithme vérifie la propriété d'exclusion mutuelle si on trouve une relation d'équivalence (par exemple un bisimulation) entre les termes $Algo$ et CS.

Correction 2 : *Dîner des philosophes*

1. Le sémaphore binaire est :

$$\begin{array}{rcl} BSem1 & ::= & \texttt{wait}.BSem0 + \texttt{signal}.BSem1 \\ Bsem0 & ::= & \texttt{signal}.BSem1 \end{array}$$

2. Les fourchettes sont obtenue par renommage du sémaphore binaire. Par exemple, pour la fourchette 1 (initialement libre) :

$$\texttt{Four1} \quad ::= \quad Bsem1[\texttt{get1}/\texttt{wait}, \texttt{put1}/\texttt{signal}]$$

3. Le comportement d'un philosophe générique est :

$$\texttt{Philo} \quad ::= \quad \texttt{pense}.\texttt{takeR}.\texttt{takeL}.\texttt{mange}.\texttt{leaveR}.\texttt{leaveL}.Philo$$

4. Les philosophes sont obtenus par renommage, par exemple pour le philosophe 1 qui utilise les fourchettes 1 et 5 on obtient :

$$\begin{array}{rcl} \texttt{Philo1} & ::= & \texttt{Philo}\ [\texttt{pense1}/\texttt{pense}, \texttt{get1}/\texttt{takeR}, \texttt{get5}/\texttt{takeL}, \\ & & \quad \texttt{mange1}/\texttt{mange}, \texttt{put1}/\texttt{leaveR}, \texttt{put5}/\texttt{leaveL}] \end{array}$$

5. Le dîner est obtenu par composition parallèle des processus ci-dessus :

$$Diner \quad ::= \quad (|_{i \in 1..5}\ \texttt{Four}i \mid \texttt{Philo}i) \backslash \{\texttt{get}i, \texttt{put}i \mid i \in 1..5\}$$

Correction 3 : *Protocole du bit alterné*

1. Les définitions des différents composantes de ce protocole sont :

$$\begin{array}{rcl} \texttt{Timer} & ::= & \texttt{time}.\overline{\texttt{timeout}}.\texttt{Timer} \\ \texttt{Buf}_1^1 & ::= & \texttt{in0}.\texttt{Buf0}_1^0 + \texttt{in1}.\texttt{Buf1}_1^0 \\ \texttt{Buf0}_1^0 & ::= & \overline{\texttt{out0}}.\texttt{Buf}_1^1 + \tau.\texttt{Buf}_1^1 + \overline{\texttt{out0}}.\overline{\texttt{out0}}.\texttt{Buf}_1^1 \\ \texttt{Ack} & ::= & \texttt{Buf}_1^1[\texttt{reply}b/\texttt{in}b, \texttt{ack}b/\texttt{out}b \mid b \in \{0,1\}] \\ \texttt{Trans} & ::= & \texttt{Buf}_1^1[\texttt{send}b/\texttt{in}b, \texttt{trans}b/\texttt{out}b \mid b \in \{0,1\}] \\ \texttt{Sender}b & ::= & \texttt{accept}.\texttt{PSend}b \\ \texttt{PSend}b & ::= & \overline{\texttt{send}b}.\overline{\texttt{time}}.\texttt{WaitAck}b \\ \texttt{WaitAck}b & ::= & \texttt{timeout}.\texttt{PSend}b + \texttt{ack}b.\texttt{timeout}.\texttt{Sender}\overline{b} + \texttt{ack}\overline{b}.\texttt{WaitAck}b \\ \texttt{Receiver}b & ::= & \texttt{trans}b.\overline{\texttt{deliver}}.\texttt{PRec}b + \texttt{trans}\overline{b}.\texttt{Receiver}b \\ \texttt{PRec}b & ::= & \overline{\texttt{reply}b}.\overline{\texttt{time}}.\texttt{WaitNext}b \\ \texttt{WaitNext}b & ::= & \texttt{timeout}.\texttt{PRec}b + \texttt{trans}\overline{b}.\overline{\texttt{deliver}}.\texttt{PRec}\overline{b} + \texttt{trans}b.\texttt{WaitNext}b \end{array}$$

   où $b \in \{0,1\}$.

2. Le système complet est obtenu en composant en parallèle les processus ci-dessus. Dans un premier temps, on compose l'émetteur et le récepteur avec leur *timer* afin d'éviter l'interférence de ces timers :

$$\begin{array}{rcl} \texttt{Sender}b' & ::= & (\texttt{Sender}b \mid \texttt{Timer}) \backslash \{\texttt{time}, \texttt{timeout}\} \\ \texttt{Receiver}b' & ::= & (\texttt{Receiver}b \mid \texttt{Timer}) \backslash \{\texttt{time}, \texttt{timeout}\} \end{array}$$

Puis, un compose ces processus avec les deux canaux de communication en supposant que la valeur de bit est la même dans les deux processus communicants, ici 0 :

$$ABP \quad ::= \quad (\texttt{Sender}0' \mid \texttt{Receiver}0' \mid \texttt{Ack} \mid \texttt{Trans}) \backslash \{\texttt{ack}b, \texttt{reply}b, \texttt{trans}b, \texttt{send}b \mid b \in \{0,1\}\}$$

3. Un tampon fiable de taille 1 est donné par l'expression suivante :

$$\texttt{Buf}_1 \quad ::= \quad \texttt{accept}.\overline{\texttt{deliver}}.\texttt{Buf}_1$$

Correction 4 : *Canaux de communication*

1. On définit le tampon comme un compteur ayant des valeurs entre $[0, N]$:

$$\begin{array}{lcl} \texttt{Buf}_N^N & ::= & \texttt{in}.\texttt{Buf}_N^{N-1} \\ \texttt{Buf}_N^i & ::= & \texttt{in}.\texttt{Buf}_N^{i-1} + \overline{\texttt{out}}.\texttt{Buf}_N^{i+1} \\ \texttt{Buf}_N^0 & ::= & \overline{\texttt{out}}.\texttt{Buf}_N^1 \end{array}$$

   avec $i \in 2..N-1$.

2. En utilisant un tampon a une place, la définition dévient :

$$\texttt{PBuf}_N^N \quad ::= \quad \underbrace{(\texttt{Buf}_1^1 \mid \cdots \mid \texttt{Buf}_1^1)}_{N}$$

3. Pour la politique FIFO, on connecte les tampons à une place en séquence comme suit :

$$\texttt{SBuf}_N^N \quad ::= \quad (\texttt{Buf}_1^1[a_1/\texttt{out}] \mid \texttt{Buf}_1^1[a_1/\texttt{in}, a_2/\texttt{out}] \mid \cdots \mid \texttt{Buf}_1^1[a_{N_1}/\texttt{in}]) \backslash \{a_i \mid i \in 1..N-1\}$$

4. Les termes $\texttt{PBuf}_N^N$ et $\texttt{SBuf}_N^N$ ne sont pas bisimilaires car $\texttt{SBuf}_N^N$ comporte des actions internes τ issues de la synchronisation entre les tampons.