

TD 2 : Systèmes communicants et Comparaison de systèmes : *traces, échecs, bi-simulation forte et faible*

Durée 3h

L3 : Formation initiale

Melliti Tarek

27 Mars 2008

Préambule : l'outils CWB-NC

CWB-NC est un outil qui regroupe plusieurs langages de spécification algébrique pour les systèmes à événements discrets. Il offre également plusieurs fonctionnalités permettant la vérification et la validation des systèmes. On va, durant les séances de cours, découvrir la théorie de ses fonctionnalités et les appliquer en séances de TDs. Aujourd'hui on va s'initier à l'outil et appliquer les concepts vus en cours à savoir.

- écriture des LTSs et composition parallèle (produit synchrone et produit asynchrone) et alternative.
- écriture des systèmes communicants et composition parallèle (ouverte et encapsulation) et alternative.
- comparaison de deux LTSs (classiques ou communicants) :
 - pré-ordre de traces et d'échecs
 - équivalence de traces et d'échecs
 - bi-simulation forte et faible

Obtenir l'outils :

L'outils est gratuit et disponible ici : <http://www.cs.sunysb.edu/~cwb>

Pour l'installation, consultez le manuel des utilisateurs.

Prise en main

Les systèmes sont décrits par des LTSs. À chaque LTS on peut associer un nom (un identifiant). Définir un système revient donc à décrire l'automate de son comportement. Il est aussi possible de définir des systèmes en appliquant des opérations de composition (alternative et/ou parallèle) sur d'autres systèmes. Les systèmes sont définis toujours dans des fichiers texte qui portent des extensions différentes selon qu'on s'intéresse à des systèmes classiques (.csp) (vus dans la première partie du cours) ou des systèmes communicants (.ccs).

Les LTS non-communicants :

Les LTSs sont déclarés dans un fichier texte qui porte l'extension ".csp". Un fichier est une suite de définitions de processus, voici sa grammaire BNF :

```

fichierCSP ::= (lien)*
lien ::= "proc" id "=" definitionSys
definitionSys ::= lts | definitionSys "[" definitionSys | definitionSys "[||]" definitionSys |
definitionSys "[|" list-action "]" definitionSys | definitionSys "[|" list-renommage "]" |
definitionSys "\" "{" list-action "}" | "(" definitionSys ")"
renommage ::= action "<-" action
lts ::= "Aut" "(" "start" "=" entier, (etat)* ")"
état ::= entier ":" (transition)*
transition ::= action "{" list-entier "}"
action ::= "t" | "delta" | id

```

les LTS communicant :

Idem, sauf que l'extension du fichier est ".ccs". Parmi les actions on distingue les actions et leurs compléments.

```

fichierCCS ::= lien*
lien ::= "proc" id "=" definitionSys
definitionSys ::= lts | definitionSys "+" definitionSys | definitionSys "|" definitionSys |
definitionSys "[|" list-renommage "]" | definitionSys "\" "{" list-action "}" | "(" definitionSys ")"
renommage ::= id "/" id
lts ::= "Aut" "(" "start" "=" entier, (etat)* ")"
état ::= entier ":" (transition)*
transition ::= action "{" list-entier "}"
action ::= "t" | id | " ' "id

```

lien avec le cours

Ces deux langages permettent de spécifier des systèmes sous forme de LTS et offrent également un certain nombre d'opérateurs pour créer des systèmes composés. En cours on a vu deux classes de système; les systèmes non-communicants et les systèmes communicants.

Les systèmes non communicants : Généralement pour les systèmes non communicants on utilise CSP. En cours on a vu quatre formes de produit : le produit libre, le produit cartésien, le produit cartésien avec ϵ et le produit synchrone par un vecteur de synchronisation.

Le langage CSP est incapable de réaliser le produit cartésien (sans et avec ϵ) par contre il est possible pour les autres produit de les réaliser parfois directement parfois moyennant des renommages et des extensions de comportement. Voici la signification des opérateurs CSP. Soit S1 et S2 deux LTS et B un ensemble d'actions :

- S1 || S2 : représente le choix alternatif entre les systèmes S1 et S2 (attention les τ ne résolvent pas le choix, cad elles le reportent.).
- S1 ||| S2 : le produit libre entre S1 et S2.
- S1 [| B || S2 : le produit synchrone entre S1 et S2 avec comme vecteur de synchronisation $V = \{ \langle a, a \rangle \mid a \in B \} \cup \langle a, \epsilon \rangle \mid a \in A1 \setminus B \cup \langle \epsilon, a \rangle \mid a \in A2 \setminus B$. En d'autres termes, le système résultat synchronisera les actions portant les mêmes noms et qui figurent dans l'ensemble B.
- S1 A : masquage des actions A dans le système S1.
- S1 [| a₁ <- b₁, ...] : permet de construire un système où on remplace dans S1 les a_i par les b_i .

Le produit synchronisé est le plus difficile à manipuler. Le cas le plus simple est quand chaque action d'un système participe à une et une seule ligne de la matrice de synchronisation. Il suffit donc de renommer chaque action participant à ce vecteur par le nom de l'action du système global. Par contre si l'action d'un système participe à plus qu'une ligne dans la matrice de synchronisation, il faudra la dupliquer dans le LTS avec un nom différent à chaque fois qu'elle apparaît. Pour illustrer ça on va résoudre le produit synchronisé de la question 4 du TD1 où l'action v du système S2 participe à deux lignes. Voici le fichier CSP correspondant.

```

proc S1 = Aut(start = 0,
0 : a {1}
1 : b {2} c {3}
2 : d {0}
3 : e {0} )

```

```

proc S2= Aut (start = 0,
0 : u {1}
1 : v {2}
2 : tt {0})

```

```

proc S21= Aut (start = 0, 0 : u {1}
1 : v1 {2} v2 {2}
2 : tt {0})

```

```

proc S= ((S1 [[a <- act1, b <- act2 , c <- act3,d <- act4, e <- act5 ]]) [[{act1,act2,act3,act4}]]
(S21[[u <- act1, v1 <- act2, v2 <- act3, tt <- act4]]))

```

le Système $S21$ est la modification du Système $S2$ car l'action v participe dans deux vecteurs d'où sa duplication en $v1$ et $v2$.

Cas des systèmes Communicants :

Pour spécifier des systèmes communicants, on utilise le langage CCS (voir grammaire plus haut). CCS permet de définir la complémentarité entre les actions; Deux actions sont complémentaires si elles portent le même nom. On distingue l'action d'envoi de l'action de réception en la précédant par le caractère "'". CCS offre un ensemble d'opérateurs pour la composition. Soit $S1$ et $S2$ deux systèmes et B un ensemble d'actions :

- $S1 + S2$: permet une composition alternative.
- $S1 [a_1 \setminus b_1, \dots]$: permet la création d'un système similaire à $S1$ où les b_i sont renommées par les a_i .
- $S1|S2$: permet une composition parallèle de $S1$ et $S2$. Le Système résultat est un système ouvert (voir cours).
- $(S1|S2)\setminus B$: permet de construire une composition parallèle de $S1$ et de $S2$. Le Système résultat est un système clos sur les communications concernant les actions de B .

Exemple : on va illustrer l'utilisation CCS sur l'exemple des bi-piles du cours. Voici à quoi va ressembler le fichier ccs :

```

/* définition de la pile 1 */ proc P1 = Aut (start=0,
0 : e1 {1}
1 : 'd1 {0} e1{2}
2 : 'd1 {1}
)
/* définition de la pile 2 */ proc P2 = Aut (start=0,
0 : e2 {1}
1 : 'd2 {0} e2 {2}
2 : 'd2 {1}
)

```

```

/* définition du système bi-piles clos */
proc CLOS = (P1 [dec / d1] | P2 [dec / e2] ) \ {dec}

```

```

/* definition du système bi-piles ouvert */

```

```

proc OPEN = (P1 [dec / d1] | P2 [dec / e2] )

```

La seule contrainte de CCS est que les actions complémentaires doivent porter le même nom. Dans le cas de composition de systèmes clos ceci ne pose pas de problème car toutes les communications sont remplacées par des τ . Par contre ça pose un petit problème dans le cas d'un système ouvert où on aura du mal, parfois, à identifier à quel sous système une action appartient (mais ça n'a pas un impact sur la correction de l'analyse). Sur l'exemple du bi-piles, dans le système OPEN

les actions $e2$ apparaîtront comme dec et les actions $'d1$ apparaîtront comme $'dec$.

Exercice - 1 *Modélisation et composition de systèmes communicants*

1- Modélisez le système de parking de l'exercice 3 du TD1 en utilisant CSP.

On considère maintenant l'exemple de l'algorithme de Peterson vu en cours. Si on fait les hypothèses suivantes :

- La lecture de la valeur d'une variable est vue par la variable comme une action d'envoi.
- Le test de la valeur d'une variable par l'un des processus est vu comme une réception.
- L'action d'affectation d'une valeur est vue comme une réception par la variable.
- L'action d'affectation d'une valeur est vue comme un envoi par le processus.

2- Modélisez le comportement de l'algorithme de Peterson comme un système communicant en utilisant CCS.

Exercice - 2 *Pre-ordre et équivalence de traces et d'échecs*

1- Donnez à chaque fois le process P qui génère les traces suivantes :

- $trace(P) = \{\epsilon, a, aa, aaa, aaaa, \dots\}$
- $trace(P) = \{\epsilon, a, ab, ac\}$
- $trace(P) = \{\epsilon, a, ab, cd, bc\}$
- $trace(P) = \{\epsilon, a, ab, abb, abbc, abbcbb, abbcbbb, \dots\}$
- $\{\epsilon, abcd, acbd, acdb, cabd, cadb, cdab\} \subset trace(P)$

2- Ordonnez les systèmes de la figure 1 selon les relations de pre-ordre de traces (\sqsubseteq_T) et d'échecs (\sqsubseteq_F). Justifiez à chaque fois votre réponse.

3- Écrivez les cinq systèmes dans un fichier "trace.csp"

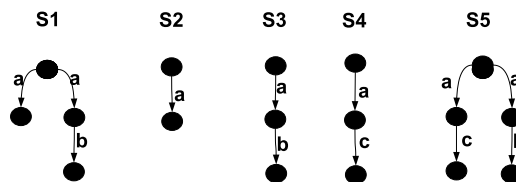


FIGURE 1 – cinq systèmes à ordonner

4- Ouvrez CWB-NC avec option *csp*, chargez le fichier et validez vos réponses.

Exercice - 3 *Bi-simulation forte*

1- Testez la bi-simulation forte entre les systèmes suivants :

2- Écrivez les systèmes dans un fichier "bissimulation.csp" et validez vos réponses de la **question 1**.

Exercice - 4 *Bi-simulation faible et congruence*

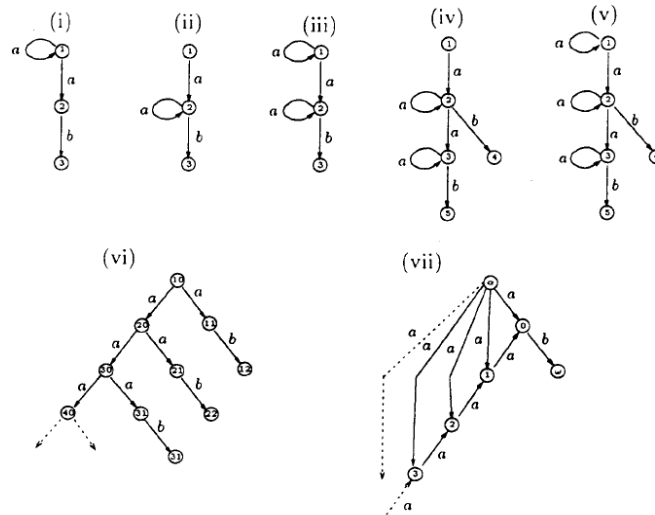


FIGURE 2 – Quelques systèmes de transition

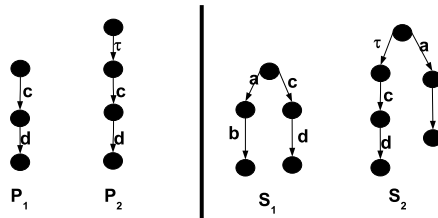


FIGURE 3 – bi-simulation faible et congruence

Soient les systèmes P_1 , P_2 , S_1 et S_2 suivants :

- 1- Est-ce que les systèmes P_1 et P_2 sont faiblement bi-similaires? (vous pouvez utiliser l'outil pour répondre)
- 2- Écrivez S_1 et S_2 en fonction de P_1 et P_2 .
- 3- Est-ce que S_1 est faiblement bi-similaire à S_2 ?
- 4- Que constatez vous concernant la congruence de la bi-simulation faible?
- 5- À votre avis quelle modification doit-on apporter à la définition de la bi-simulation faible pour obtenir une relation congruente (qu'on note \approx^C)?