Ecole Polytechnique

Cours : "Parallélisme"
Travaux dirigés
E. Goubault, S. Putot & O. Bouissou

TD 6

15 Mars 2006

1 Bisimulations

Prouver que les termes suivants sont bisimulation équivalents (où P, Q et R sont n'importe quel terme CCS),

- $P+Q$ et $Q+P$
- $P+(Q+R)$ et $(P+Q)+R$
- $P \mid (Q \mid R)$ et $(P \mid Q) \mid R$

2 Sémaphores

- Donner un terme CCS `Sem` qui représente les actions possibles associées à un sémaphore binaire (`P` et `V`).
- Définir (dans le style du cours) un terme CCS $\texttt{Sem}_n\texttt{(k)}$ qui représente un sémaphore n-aire, dont k entrées sont déjà prises.
- Peut-on définir directement $\texttt{Sem}_n$ à partir de `Sem`? Comment peut-on prouver que ces deux définitions sont équivalentes?

3 Protocole du bit alterné

On s'intéresse ici à un protocole de communication entre deux agents, supposés permettre le bon acheminement des messages même lorsque les lignes de communications ne sont pas parfaites. On doit quand même supposer un certain nombre de choses concernant ces lignes: elles peuvent simplement perdre ou dupliquer un message, mais pas le corrompre.

On va représenter nos deux agents par des termes CCS `Send` et `Reply`, dont le seul but est pour l'un d'envoyer des messages sur une ligne (un buffer à capacité infinie), également représentée par un terme CCS `Trans`, vers `Reply`, qui lui, va renvoyer un reçu à `Send`, par la ligne `Ack`. Ces deux lignes sont censées être imparfaites dans le sens explicité plus haut.

Pour communiquer, `Send` et `Reply` vont utiliser le protocole du bit alterné. Le nom du protocole vient du fait que chaque message va être taggé, alternativement par un 1 et par un 0:

- `Send` fonctionne de la façon suivante:

- Il prend un message à envoyer sur un canal `accept`
- Il l'envoie taggé avec un bit b le long de `Trans` et démarre une horloge (représentée par un terme CCS `Timer`)
- Il y a alors trois possibilités:
  * Il reçoit un *timeout* du timer: alors il renvoie le même message sur `Trans`. toujours taggé avec le même bit.
  * Il reçoit une confirmation de réception sur la ligne `Ack` avec la valeur b: il est alors prêt à recevoir un nouveau message (qu'il renverra avec le bit $1-b$).
  * Il reçoit $1-b$ sur la ligne `Ack`: il ne fait rien.

- `Reply` fonctionne d'une façon duale:
  - Lorsqu'il reçoit un message sur `Trans`, il renvoie le bit b avec lequel il avait été taggé, sur la canal `Ack`, renvoie le contenu du message sur le canal `deliver` et met en route une horloge.
  - Il y a alors trois possibilités:
    * Il reçoit un timeout de l'horloge: il renvoie le message b sur la ligne `Ack`.
    * Il reçoit un nouveau message taggé avec le bit $1-b$ sur la ligne `Trans`: il peut alors délivrer le nouveau message, qu'il confirme sur la ligne `Ack` avec le bit $1-b$.
    * Il reçoit un message avec bit b: il l'ignore.

On répondra successivement aux questions suivantes:

- Définir les termes CCS `Send` et `Reply`. Il sera pratique de les paramétrer par la valeur du bit courant.
- Définir le terme `Timer`: on supposera qu'il aura deux canaux, `time` pour sa mise en route, et `timeout` pour signaler le timeout.
- Simplifier le terme `Send'(b)=(Send(b) | Timer)`$\backslash\{$`time,timeout`$\}$, de même pour `Reply`.
- Définir les transitions que `Ack` et `Trans` doivent vérifier. On paramètrera également ces termes: par quoi cette fois-ci?
- On considère maintenant le buffer à une place:

  $$\texttt{Buff=accept.Buff'}$$
  $$\texttt{Buff'=}\overline{\texttt{deliver}}\texttt{.Buff}$$

  Prouver que le système complet, dans un état initial qu'il reste à définir, est équivalent à `Buff`.