Systemes Synchrones
MPRI – cours 2.23-1 2008-2009

Examen
19 novembre 2008

Compilation d'Esterel pur vers Lustre (10 points)

L'objectif de cet exercice est de définir une traduction d'un petit langage synchrone impératif *à la* Esterel vers Lustre.

Pour simplifier, on considère d'abord un sous-ensemble très simple d'Esterel avec une seule entrée (toujours nommée `E`), une seule sortie (toujours nommée `S`), et un jeu d'instructions réduit. La syntaxe des programmes est la suivante :

$$
\begin{array}{rcl}
\mathit{prg} & ::= & \texttt{module P ; input E ; output S ; } \mathit{inst}. \\
\mathit{inst} & ::= & \texttt{nothing} \mid \texttt{pause} \mid \texttt{emit S} \mid \texttt{await E} \\
& \mid & \texttt{present E then } \mathit{inst} \texttt{ else } \mathit{inst} \texttt{ end} \\
& \mid & \mathit{inst} \texttt{ ; } \mathit{inst} \mid \mathit{inst} \texttt{ || } \mathit{inst} \mid \texttt{loop } \mathit{inst} \texttt{ end} \mid \texttt{suspend } \mathit{inst} \texttt{ when } E
\end{array}
$$

On considère ici qu'un programme est formé d'un seul module ayant une seule entrée `E` et une seule sortie `S`. L'instruction `nothing` est l'instruction qui ne fait rien et passe instanément le contrôle ; `pause` est une instruction qui attend un instant (sur l'horloge de base) ; `emit S` émet la sortie `S` et passe le contrôle instantanément ; `await E` fige le contrôle jusqu'à la prochaine occurrence de `E` (dans le futur strict) ; l'instruction `present E then` $inst_1$ `else` $inst_2$ `end` exécute $inst_1$ si `E` est présent et $inst_2$ sinon ; $inst_1$ `;` $inst_2$ est l'opération de séquence où le contrôle passe instantanément de $inst_1$ à $inst_2$; $inst_1$ `||` $inst_2$ est la composition parallèle synchrone et `loop` *inst* `end` est la répétition infinie qui se comporte comme *inst* `; loop` *inst* `end`. L'instruction `suspend` *inst* `when E` permet de suspendre l'exécution d'une instruction : *inst* est activé seulement lorsque `E` n'est pas présent.

Principe de la traduction structurelle : l'idée est d'associer à toute sous-instruction Esterel un nœud Lustre qui implémente son comportement : c'est essentiellement le principe de la représentation en circuits d'automates.

Le comportement est basé sur la notion de *passage de contrôle* : l'instruction reçoit le contrôle de l'extérieur, implémente sa fonctionnalité, puis passe le contrôle en séquence si et quand elle termine (ce qui peut être instantané, prendre un certain temps, ou même ne jamais arriver, suivant les cas).

À toute instruction Esterel *inst*, on va associer un nœud Lustre qui permet de traduire la notion de passage de contrôle. L'en-tête de ce nœud est toujours de la forme :

$$\texttt{node M}_{\mathit{inst}}\texttt{(E, Cin : bool) returns (Sloc, Cout : bool) ;}$$

où les entrées sont :

- `E` correspond à l'entrée du programme global ; on adopte l'équivalence habituelle entre signaux purs (d'Esterel) et flots booléens (de Lustre), c'est-à-dire « présent=vrai » et « absent=faux »,
- `Cin` (Control-in) est vrai à un instant t ssi le contexte passe le contrôle au nœud,

et les sorties sont :

- `Sloc` représente la contribution locale à l'émission de la sortie `S` : à tout instant, elle est vraie ssi la sortie est « émise » à l'intérieur de l'instruction Esterel implémentée pas le nœud,
- `Cout` (Control-out) est vrai à un instant t ssi le nœud, qui a donc terminé son comportement, passe le contrôle en séquence.

Exemple de « `nothing` » : cette instruction Esterel ne fait rien d'autre que passer instantanément le contrôle à la suite, donc :

- chaque fois qu'elle reçoit le contrôle, elle le passe immédiatement (`Cout = Cin`),
- elle ne participe jamais à l'émission de la sortie (`Sloc = false`).

Le nœud modulaire complet est donc :

```
node Mnothing(E, Cin : bool) returns (Sloc, Cout : bool);
let
   Cout = Cin;
   Sloc = false;
tel
```

Exemple de « `present E then` $inst_1$ `else` $inst_2$ `end` » : on suppose qu'on a contruit le nœud modulaire `M1` pour l'instruction $inst_1$ et le nœud modulaire `M2` pour l'instruction $inst_1$.

Le nœud modulaire correspondant à « `present E then` $inst_1$ `else` $inst_2$ `end` » est le suivant :

```
node Mpresent(E, Cin : bool) returns (Sloc, Cout : bool);
var Sloc1, Cout1, Sloc2, Cout2 : bool;
let
   Sloc1, Cout1 = M1(E, Cin and E);
   Sloc2, Cout2 = M2(E, Cin and not E);
   Cout = Cout1 or Cout2;
   Sloc = Sloc1 or Sloc2;
tel
```

Ce nœud s'interprète comme suit :

- chaque fois qu'il reçoit le contrôle alors que l'entrée `E` est vraie, il passe le contrôle à `M1` : `Sloc1, Cout1 = M1(Cin and E, E)`,
- chaque fois qu'il reçoit le contrôle alors que l'entrée `E` est fausse, il passe le contrôle à `M2` : `Sloc2, Cout2 = M2(Cin and not E, E)`,
- il passe le contrôle en sortie quand un des nœuds internes passe le contrôle : `Cout = Cout1 or Cout2`,
- il participe à l'émission de la sortie quand un des nœuds internes « émet » la sortie : `Sloc = Sloc1 or Sloc2`.

Question 1

Expliquer en quelques phrases comment se comportent respectivement les instructions « `pause` » et « `emit S` » en terme de passage de contrôle et de contribution à l'émission de la sortie. Écrire en conséquence les nœuds modulaires `Mpause` et `MemitS`.

Question 2

On s'intéresse maintenant à l'instruction « `await E` ». Expliquer le comportement de cette instruction et écrire le nœud modulaire correspondant.

Question 3

On se penche maintenant sur les instructions composées.

On suppose que l'on a été capable de construire les nœuds modulaires associés aux instructions $inst_1$ et $inst_2$ (resp. `M1` et `M2`).

Donner (en utilisant `M1` et `M2`) le nœud correspondant à « $inst_1$ `;` $inst_2$ ».

Même question pour « $inst_1$ `||` $inst_2$ ».

Question 4

Soit `M1` le nœud correspondant à $inst_1$, donner le nœud modulaire pour l'instruction « `loop` $inst_1$ `end` ».

Une boucle Esterel dont le corps peut ne pas prendre de temps est incorrecte car le système ne réagit pas (boucle de causalité). Pouvez-vous caractériser en quelques mots le nœud Lustre que l'on obtient en traduisant une boucle Esterel incorrecte ?

Question 5

Soit `M1` le noeud correspondant à *inst*. Donner le noeud modulaire pour l'instruction « `suspend` *inst* `when` E ».

Question 6

On s'intéresse maintenant au programme principal. Soit un programme mini-esterel :

```
module P ;
input E ;
output S ;
    inst.
```

et soit `M` le nœud modulaire obtenu pour l'instruction *inst*, écrire un nœud Lustre principal qui implémente le programme `P` (et qui a donc pour seule entrée `E` et pour seule sortie `S`).

Question 7 (bonus)

On considère maintenant le cas plus général d'un système ayant plusieurs entrées (notées $E_1, ..., E_n$) et produisant plusieurs sorties $S_1, ..., S_k$.

- Généraliser le principe de traduction précédent dans ce cas.
- Redéfinir les règles de traduction des structures de contrôle dans ce cas. Pour simplifier la traduction, on pourra se ramener au cas où tous les noeuds ont la même interface (même nombre d'entrées, de sorties et de "bits" de contrôle tels que `Cin` et `Cout`