Corrigé (Esterel vers Lustre)

December 19, 2008

pause

- passe le contrôle avec UN INSTANT de retard
- ne participe pas à l'émission de la sortie

```
node Mpause(E, Cin : bool) returns (Sloc, Cout : bool);
let
   Cout = false -> pre Cin;
   Sloc = false;
tel
```

emit S

- passe le contrôle immédiatement
- émet la sortie SI ELLE A LE CONTROLE

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

await E

- quand elle recoit le contrôle, se bloque dans l'attente d'une occurrence FUTURE de E
- ne participe pas à l'émission de la sortie
- N.B. Utilisation d'une variable locale : waiting "on est EN TRAIN D'ATTENDRE"

```
node MawaitE(E, Cin : bool) returns (Sloc, Cout : bool);
var waiting : bool;
let
   waiting = false -> pre(Cin or (waiting and not E));
   Cout = waiting and E;
   Sloc = false;
tel
```

await immediate E

- quand on recoit Cin, si E est la on passe immédiatement le contrôle, sinon on se bloque dans l'attente d'une occurrence future de E
- ne participe pas à l'émission de la sortie

```
node Mawait_immediateE(E, Cin : bool) returns (Sloc, Cout : bool);
var waiting : bool;
let
   waiting = Cin or (false -> pre(waiting and not E));
   Cout = waiting and E;
   Sloc = false;
tel
```

Séquence `<inst1> ; <inst2>`

- quand on a Cin, c'est Minst1 qui prend immédiatement le contrôle, quand Minst1 "emet" SON Cout, c'est Minst2 qui prend immédiatement le contrôle, quand Minst2 passe le contrôle, c'est l'instruction `<inst1> ; <inst2>` DANS SA GLOBALITE, qui passe le contrôle.
- S est émis par `<inst1> ; <inst2>`, s'il est émis par `<inst1>` OU par `<inst2>`

```
node Minst1Sinst2(E, Cin : bool) returns (Sloc, Cout : bool);
var Cout1, Sloc1, Sloc2 : bool;
let
   Sloc1, Cout1 = Minst1(E, Cin);
   Sloc2, Cout  = Minst2(E, Cout1);
   Sloc = Sloc1 or Sloc2;
tel
```

Conditionnelle `if E then <inst1> else <inst2> end`

- quand on a Cin, si E est présent (vrai), alors on passe immédiatement le contrôle a ¡inst1¿, sinon on passe le contrôle à ¡inst2¿ ; au final :
- `Cin1 = Cin and E; Cin2 = Cin and not E`
- ATTENTION : le `"if ... end"`, DANS SA GLOBALITE, passe le contrôle immédiatement si et quand sa branche `<inst1>` passe le controle OU si et quand sa branche `<inst2>` passe le controle, et ceci INDEPENDAMMENT DE LA VALEUR COURANTE DE E ; au final :
- `Cout = Cout1 or Cout2;`
- raisonnement similaire pour la sortie : elle est émise à l'intérieur du `"if .. end"` ssi elle est émise par la branche `<inst1>` OU par la branche `<inst2>`, et ceci INDEPENDAMMENT DE E.

```
node MIfEinst1ELSEinst2(E, Cin : bool) returns (Sloc, Cout : bool);
var Sloc1, Cin1, Cout1 : bool;
    Sloc2, Cin2, Cout2 : bool;
let
   Cin1 = Cin and E;
   Cin2 = Cin and not E;
   Sloc1, Cout1 = Minst1(E, Cin1);
   Sloc2, Cout2 = Minst2(E, Cin2);
   Cout = Cout1 or Cout2;
   Sloc = Sloc1 or Sloc2;
tel
```

Composition parallèle: `<inst1> || <inst2>`

- quand on a `Cin`, on passe le contrôle immédiatement à la fois dans `<inst1>` ET `<inst2>` ; on rend globalement le contrôle :
- si `<inst1>` et `<inst2>` rendent le contrôle en même temps `(Cout1 and Cout2)`,
- si `<inst1>` rend le contrôle, et que `<inst2>` l'avait rendu précédemment,
- et vice-versa, si `<inst2>` rend le contrôle, et que `<inst1>` l'avait rendu précédemment.
- la sortie est émise (éventuellement en même temps) si elle est émise à l'intérieur de `<inst1>` ou de `<inst2>`

L'idée est d'utiliser des "mémoires", assez similaires à celle qu'on a introduite pour le "await immediate" ; par exemple : `fini1` est vrai ssi `<inst1>` termine dans l'instant, ou si `<inst1>` avait déjà terminé avant alors que `<inst2>` n'avait par encore terminé (et vice-versa pour `fini2`).

```
node Minst2PARAinst2(E, Cin : bool) returns (Sloc, Cout : bool);
var Sloc1, Cout1 : bool;
    Sloc2, Cout2 : bool;
    fini1, fini2 : bool;
let
   Sloc1, Cout1 = Minst1(E, Cin);
   Sloc2, Cout2 = Minst2(E, Cin);
   fini1 = Cout1 or (false -> pre (fini1 and not fini2));
   fini2 = Cout2 or (false -> pre (fini2 and not fini1));
   Cout = fini1 and fini2;
   Sloc = Sloc1 or Sloc2;
tel
```

boucle `loop <inst> end`

- quand on a Cin, on passe immédiatement le contrôle au corps de la boucle "`<inst>`"; quand le corps de la boucle passe le contôle, on le re-passe immediatement au début de la boucle (`<inst>`); au final : * le `Cin_loop` (`Cin` de `Minst`) est le OU du `Cin` global et du `Cout_loop` (`Cout` de `Minst`) * le Cout GLOBAL est toujours FAUX (une boucle ne rend jamais le contrôle)
- la sortie est emise par la boucle ssi elle est emise par le corps de la boucle

```
node Mloop_inst(E, Cin : bool) returns (Sloc, Cout : bool);
var Cin_loop, Cout_loop : bool;
let
Sloc, Cout_loop = Minst(E, Cin_loop);
   Cin_loop = Cin or Cout_loop;
   Cout = false;
tel
```

Question subsidiaire : si la boucle Esterel est INCORRECTE, c'est qu'elle contient une branche de contrôle instantanée qui va du début à la fin. Une fois compilée en Lustre, on va avoir dans ce cas `Cout_loop` qui dépend instantanément de `Cin_loop` (et donc de lui-même). Le programme Lustre sera donc lui aussi incorrect.

ERREURS COMMUNES : Ecrire un noeud récursif, ce qui n'existe pas en Lustre : par analogie avec les circuits, c'est comme définir un circuit infini !

Main Le contrôle est passé UNE SEULE FOIS, au premier instant `(true -> false)`. Il faut "capter" la sortie `Cout` dans une variable locale dont on ne se sert pas.

```
node M(E : bool) returns (S : bool);
var Cin, Cout : bool;
let
   Cin = true -> false;
   S, Cout = Minst(E, Cin);
tel
```