UJF - UFR IMAG : Licence d'informatique, MIAGE2
Polytech' Grenoble : RICM1
ALM : Utilisation du langage Lustre pour décrire les circuits

Les circuits sont des assemblages de portes et de bascules. Les circuits peuvent être décrits de façon hiérarchisée : un circuit comporte des sous-circuits, nommés noeuds, qui eux-mêmes peuvent comporter des sous-circuits (et cela avec n'importe quel niveau de hiérarchie) comportant des portes et des bascules.

1 Description en Lustre des assemblages de portes

La syntaxe générale d'un noeud est :

```
node identificateur ( liste d'entrées et types )
            returns (liste de sorties et types ) ;
var liste de variables internes et types ;
let
  ensemble d'équations définissant les variables internes et les sorties
  les équations sont séparées par des points virgules
tel;
```

Les portes disponibles sont `and`, `or`, `not`, `xor`. Le type principal est `bool`. Les constantes booléennes sont `true` et `false`. Il existe des `int`, et des tableaux de `bool`.

1.1 Description des blocs de base

On décrit ci-dessous un exemple de bloc simple à une sortie (`cir1`). La figure 1 donne le circuit associé.

```
node cir1 (a, b : bool) returns (c : bool);
let
  c = a and not b;
tel;
```

Et voici des blocs simples à plusieurs sorties (`cir2` et `cir2prime` qui est équivalent à `cir2`). La figure 1 donne le circuit associé.

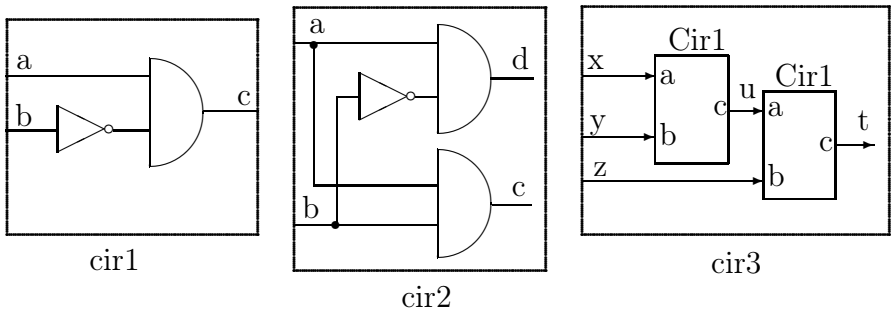

FIG. 1 – Exemples de blocs de base

```
node cir2 (a, b : bool)
  returns (c, d : bool);
let
  c = a and b;
  d = a and not b;
tel;
```

```
node cir2prime (a, b : bool)
  returns (c, d : bool);
let
  c = a and b;
  d = cir1(a, b);
tel;
```

1.2 Blocs contenant des sous-blocs

Les noeuds `cir3`, `cir3prime` et `cir3seconde` décrits ci-dessous contiennent des sous-blocs à une sortie. La figure 1 donne le circuit associé à `cir3` et `cir3prime`. `cir3seconde` est-il équivalent à `cir3` ?

```
node cir3 (x,y, z : bool) returns (t : bool);
let
  t = cir1 (cir1 (x, y), z);
tel;
```

```
node cir3prime (x,y,z:bool)
     returns (t : bool);
var u : bool;
let
  u = cir1 (x, y);
  t = cir1 (u, z);
tel;
```

```
node cir3seconde (x,y,z:bool)
     returns (t : bool);
-- cir3seconde est-il
-- équivalent à cir3 ???
let
  t = cir1 (x, cir1 (y, z));
tel;
```

Les noeuds `cir4` et `cir5` sont définis à l'aide de sous-blocs à plusieurs sorties. La figure 2 donne les circuits associés, sans flèches pour représenter les entrées/sorties. Noter que l'ordre des équations (entre `let` et `tel`) est sans importance.

```
node cir4 (a3,x2,x1,x0:bool)
 returns (y2,y1,y0,a0:bool);
var a2, a1 : bool;
let
  (a2, y2) = cir2 (x2, a3);
  (a0, y0) = cir2 (x0, a1);
  (a1, y1) = cir2 (x1, a2);
tel;
```

```
node cir5 (a,x2,x1,x0:bool)
returns (y2,y1,y0,z2,z1,z0:bool);

let
  (z2, y2) = cir2 (x2, a);
  (z0, y0) = cir2 (x0, a);
  (z1, y1) = cir2 (x1, a);
tel;
```

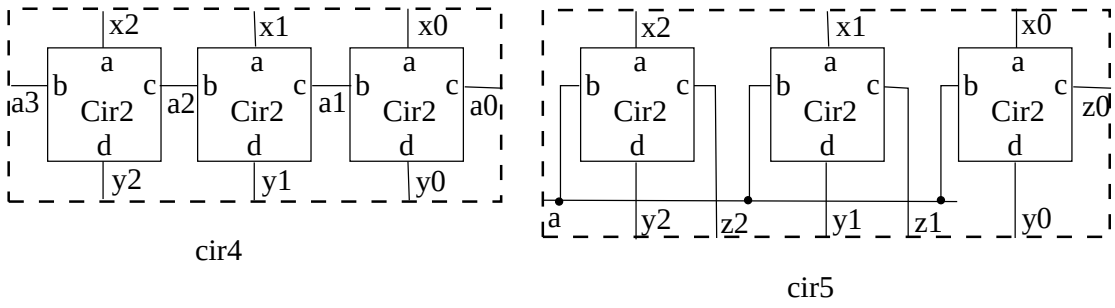

FIG. 2 – Exemples de blocs contenant des sous-blocs

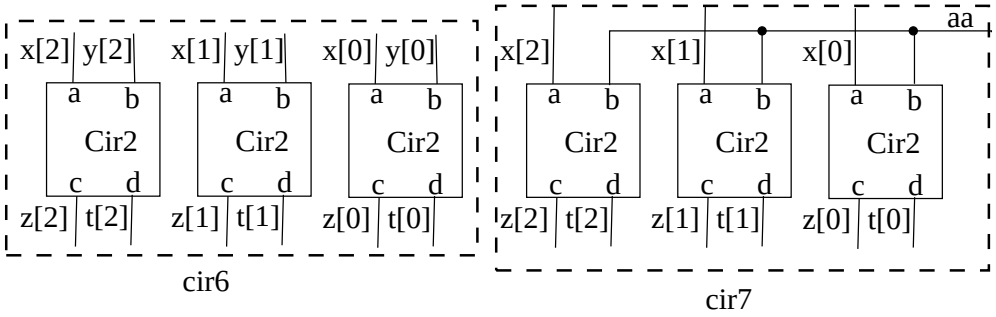

FIG. 3 – Exemples de blocs contenant des répétitions

1.3 Description de blocs contenant des répétitions

Les booléens peuvent être regroupés en vecteurs. Les indices et les tailles sont des **constantes**.

Les noeuds `cir6` et `cir6prime` utilisent des répétitions sans partage. Noter (`cir6prime`) que l'on n'est pas obligé d'écrire les indices. La figure 3 décrit le circuit `cir6`.

```
node cir6 (x, y : bool^3)
   returns (z, t : bool^3);
let
  (z[0..2], t[0..2]) =
    cir2 (x[0..2], y[0..2]);
tel;
```

```
node cir6prime (x, y : bool^3)
      returns (z, t : bool^3);
let
  (z, t) = cir2 (x, y);
tel;
```

Les noeuds `cir7` et `cir7prime` utilisent des répétitions avec partage de signal. La figure 3 décrit le circuit `cir7`.

```
node cir7 (x:bool^3 ; aa:bool)
      returns (z,t:bool^3);
let
  (z[0..2], t[0..2]) =
       cir2 (x[0..2], aa^3);
tel;
```

```
node cir7prime (x:bool^3 ; aa:bool)
        returns (z,t:bool^3);
let
  (z, t) = cir2 (x, aa^3);
tel;
```

Les noeuds `cir8` et `cir8prime` utilisent des répétitions avec mise en cascade. La version `cir8prime` ne comporte pas systématiquement les indices mais notons que l'on est obligé de les préciser pour `a`. La figure 4 décrit ce circuit.

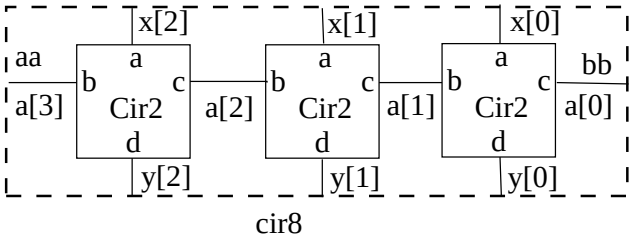

FIG. 4 – Répétition avec mise en cascade

```
node cir8 (x:bool^3 ; aa:bool)
  returns (y:bool^3 ; bb:bool);
var a : bool^4;
let
  a[3] = aa;
  bb = a[0];
  (a[0..2], y[0..2]) =
       cir2 (x[0..2], a[1..3]);
tel;
```

```
node cir8prime (x:bool^3 ; aa:bool)
      returns (y:bool^3 ; bb:bool);
var a : bool^4;
let
  a[3] = aa;
  bb = a[0];
  (a[0..2], y) = cir2 (x, a[1..3]);
tel;
```

1.4 Description de blocs génériques de taille N

On peut déclarer une constante `nn` puis l'utiliser dans une déclaration de tableau :

```
const nn : int ;      var aa : bool^nn;
```

Dans le programme, plus loin, `nn` a une valeur. Déclarer un vecteur de taille `pp` où `pp` n'est pas une constante est un non-sens.

Le noeud `cir9` utilise des répétitions sans partage. `uneinstancedecir9` utilise `cir9` avec une valeur constante pour `nn`. Attention, dans l'instance de `cir9` (`uneinstancedecir9`), `aa` est un paramètre d'un type particulier.

```
node cir9 (const nn:int ; x,y:bool^nn)
               returns (z,t:bool^nn);
let
 (z[0..(nn-1)], t[0.. (nn-1)]) =
   cir2 (x[0..(nn-1)] , y[0..(nn-1)]);
tel;
```

```
const aa = 6 ;
node uneinstancedecir9 (r,s:bool^aa)
             returns (p,q:bool^aa);
let
  (p, q) = cir9 ( aa, r, s );
tel;
```

On ferait de même des blocs génériques avec répétition, partage d'un signal ou mise en cascade.

2 Description des automates et des circuits contenant des bascules

Nous utilisons les opérateurs temporels du langage Lustre notés : -> et `pre`, prononcés "suivi de" et "pré" (comme précédent).

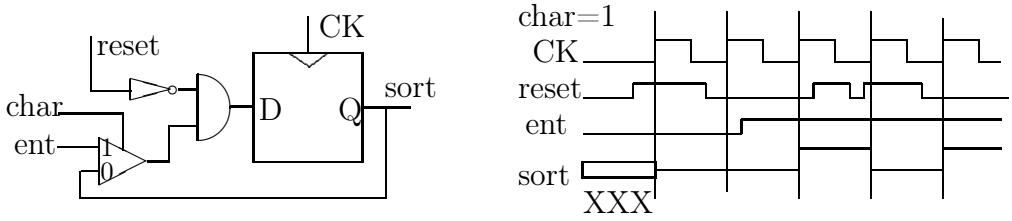

FIG. 5 – Une bascule et son comportement

2.1 Comportement de quelques bascules élémentaires

On se limite à un type de bascule qui évolue au front montant de l'horloge `CK`, à ré-initialisation synchrone (signal `reset`) qui est commandée par un signal de chargement `char`. L'entrée est nommée `ent` et la sortie `sort`.

La figure 5 montre comment cette bascule est obtenue à partir d'une simple bascule à front.

On ne traite pas d'autres types (réinitialisation asynchrone, bascules maître-esclave, verrous,..).

Dans certaines réalisations, le signal de chargement peut être en permanence à `true` . Quand ce signal est à `false` la bascule ne change pas.

La bascule est décrite par le texte Lustre suivant :

```
node unebascule (ent, char, reset : bool) returns (sort : bool);
-- Le signal CK n'a pas à être explicité
var D : bool; -- l'entrée de la bascule D simple interne
let
  D = not(reset) and mux1(char, ent, sort);
  sort = XXX -> pre D;
  -- Attention : XXX peut être remplacé au choix par true ou false.
  -- On ne sait pas ce qu'affiche une bascule avant le premier coup d'horloge
tel;

node mux1(i, t, e : bool) returns (s : bool);
let
        s = i and t or not i and e;
tel;
```

Lors de la simulation (voir plus loin) on prendra garde à la première valeur, `false`, affichée par le simulateur AVANT le premier coup d'horloge. Un moyen de ne pas la confondre avec la "vraie" valeur initiale est d'utiliser systématiquement `XXX = true` pour initialiser les bascules.

Exercices Rajoutez une entrée `set` d'initialisation à 1 à cette bascule. (Le `set` est il prioritaire par rapport au `reset` ?) Donnez la description d'une bascule sans commande de chargement ni de set , ni de reset.

Note : La boucle étrangleuse Un circuit "bien construit" ne peut contenir de boucle sans éléments de mémorisation (hormis, évidemment, à l'intérieur des bascules). De même, pour une

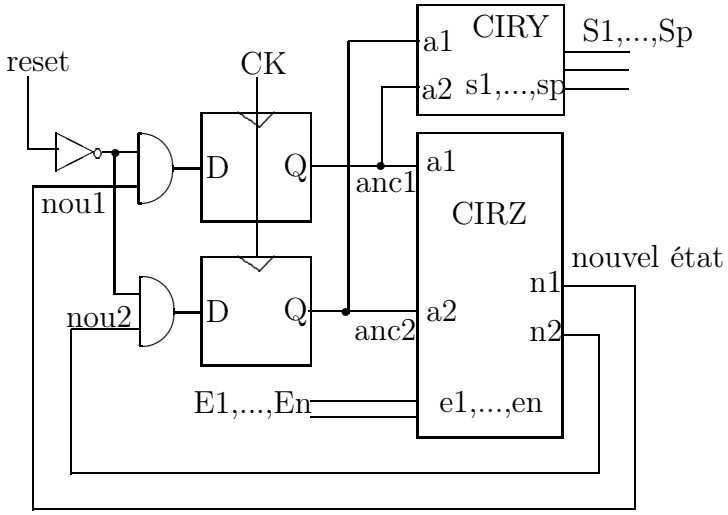

FIG. 6 – Automate de Moore

description Lustre d'un circuit.

2.2 Réalisation d'automates de Mealy, de Moore

La figure 6 décrit la réalisation d'un automate de Moore. Les entrées sont : `E1, .., En`, les sorties : `S1, .., Sp` ; et les variables internes d'état sont ici représentées sur 2 bits. On note `anc` l'ancien état et `nou` le nouvel état. Attention, lors d'une simulation, il est possible d'afficher l'ancien ou le nouvel état ; c'est une source d'erreur d'interprétation parfois ...

On peut décrire cette réalisation en Lustre en utilisant des bascules pour mémoriser l'état. Dans cette réalisation, il y a changement d'état à chaque `CK` car `char=1`.

La description en Lustre est la suivante :

```
node ciry (a1,a2:bool) returns (s1,...,sp:bool);
let
  -- fonction combinatoire de calcul des sorties
tel;

node cirz (a1,a2,e1,...,en:bool) returns (n1, n2:bool);
let
  -- fonction combinatoire de calcul de l'état suivant
tel;

node MooreGlobal (E1, ..., En, reset: bool) returns (S1, ..., Sp: bool);
-- signal char toujours actif
var anc1, anc2, nou1, nou2 : bool;
let
  anc1 = unebascule (nou1, true, reset);
  anc2 = unebascule (nou2, true, reset);
```

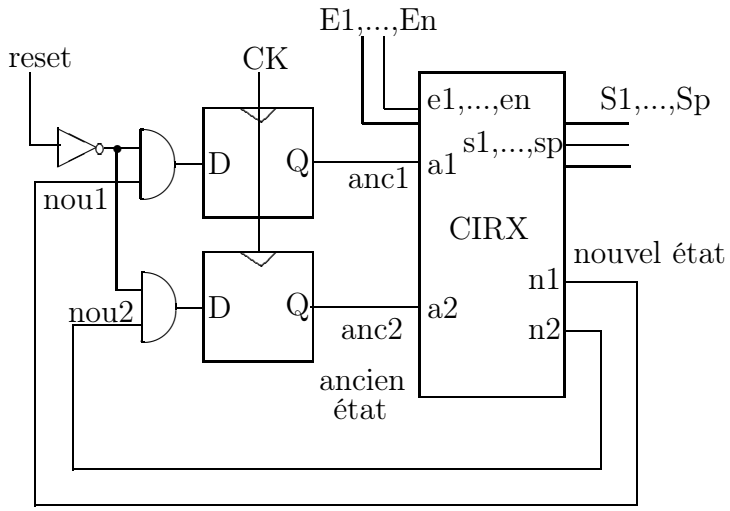

FIG. 7 – Automate de Mealy

```
  (nou1, nou2) = cirz (anc1, anc2, E1, ..., En);
  (S1, ..., Sp) = ciry (anc1, anc2);
tel;
```

Lors de la simulation, il est parfois utile de pouvoir observer les valeurs des états et pas seulement les sorties ; on utilise alors l'en-tête et les déclarations de variables locales suivantes :

```
node automate (reset, a, b : bool) returns (anc1, anc2, x, y : bool);
var ei, e1, e2, e3 : bool;
let ... tel;
```

Exercice La figure 7 décrit un automate de Mealy. Ecrire le programme Lustre qui décrit cette réalisation.

2.3 Réalisation d'un automate avec un codage un parmi n des états

Considérons l'automate de la figure 8. Dans le langage Lustre cet automate peut être décrit par le noeud suivant :

```
node automate (reset, a, b : bool) returns (x, y : bool);
var anci, anc1, anc2, anc3, ei, e1, e2, e3 : bool;
-- Dans la réalisation, e correspond à l'entrée des
-- bascules, anc correspond à la sortie des bascules
let
  -- calcul etat suivant
  ei =  reset or (anc3 or (anc2 and a));
  e1 = not reset and ((anci and a) or (anc1 and not b));
  e2 = not reset and (anc1 and b);
```

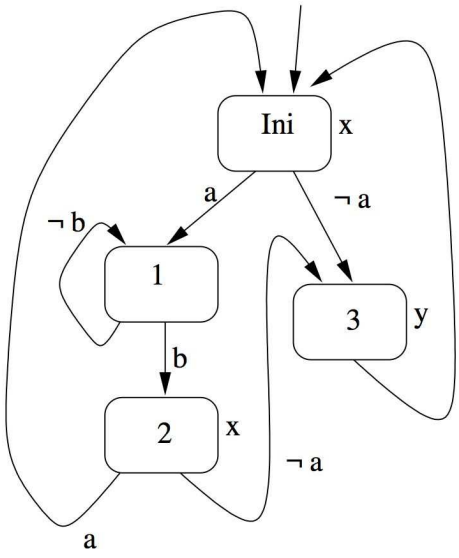

FIG. 8 – Un automate

```
  e3 = not reset and ((anci or anc2) and not a);
  anci = true -> pre ei;       -- bascule 1
  anc1 = false -> pre e1;   -- bascule 2
  anc2 = false -> pre e2;   -- bascule 3
  anc3 = false -> pre e3;   -- bascule 4
  -- calcul des sorties
  x = anci or anc2;
  y = anc3;
tel;
```

Exercice Dessiner le circuit équivalent à cette description Lustre.

2.4 Registres

Les bascules d'un registre partagent le même signal d'horloge `CK`, le même signal de ré-initialisation `reset` et, le cas échéant, le même signal de chargement `char`.

La figure 9 décrit un registre à 4 bits avec signal de chargement commun.

Ce registre peut être décrit par le noeud Lustre suivant :

```
 node unregistre4 (ent3,ent2,ent1,ent0,
                          char,reset:bool)
  returns (sort3,sort2,sort1,sort0:bool);
 let
   sort3 = unebascule (ent3, char, reset);
   -- idem pour sort2, sort1 et sort0
 tel;
```

```
node autrereg4 (ent:bool^4 ; char,reset:bool)
              returns (sort:bool^4);
let
  sort = unebascule (ent, char^4, reset^4);
tel;
```

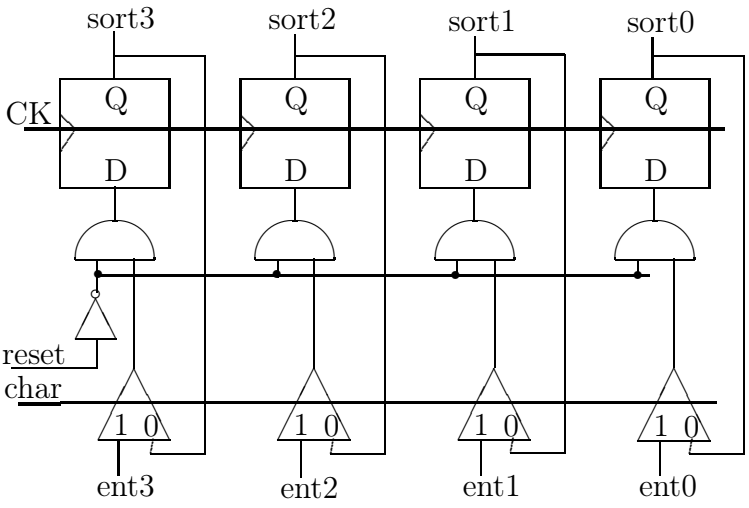

FIG. 9 – Un registre à 4 bits

Exercice La réalisation de l'automate de la figure 6 pourrait être décrite avec un registre à 2 bits. Donner la description correspondante en Lustre.

3 Les outils basés sur Lustre

La description est rangée dans un fichier nommé `NomDeFichier.lus`. Cette description peut contenir une hiérarchie de noeuds. On s'intéresse à un noeud, `NomDeNoeudEnglobant`, qui en englobe d'autres.

A l'UFR-IMA les outils sont sur le serveur mandelbrot. Vérifier que `//usr/local/lustre/bin` figure dans votre "PATH". Des version Linux, Solaris et MacOSX sont disponibles via le placard électronique.

3.1 Simulation

Simulation interactive : luciole

La commande de lancement de la simulation est :

```
luciole  NomDeFichier.lus  NomDeNoeudEnglobant
```

Il existe deux modes de fonctionnement : mode `auto-step` ou mode `compose` (menu Clocks). On n'utilise `auto-step` que dans le cas très particulier des automates qui ont une et une seule entrée vraie à chaque pas de calcul. Cette entrée joue alors, pour le simulateur, le double rôle d'entrée et d'horloge. Cela n'a pas de "vrai" sens physique. Dans le cas général (circuits combinatoires et automates en général) utiliser le mode `compose`.

On installe les entrées individuellement. La touche `step` les valide ET fait progresser l'automate. Le résultat s'affiche. Pour les circuits séquentiels, un "`step`" valide les entrées, un "`step`" fait changer les bascules.

La commande : `luciole NomDeFichier.lus NomDeNoeudEnglobant -comp` lance le simulateur directement en mode `compose`. On peut y sauvegarder la description du panneau de simulation et l'éditer. Le mode `verbose` donne la trace à l'écran au fur et à mesure.

Il existe un autre simulateur : xsimlus. Le choix du mode `compose` ou `auto-step` se fait en cliquant sur le bouton correspondant. Le fonctionnement est du même type.

Trace textuelle de simulation

La commande :

```
luciole  NomDeFich.lus   NomDeNoeud -v  >  NomDeTrace
```

permet d'avoir un listing de la simulation. L'option `-v` signifie `verbose`.

On peut aussi produire des copies d'écran pour les circuits combinatoires.

Dessins de chronogrammes d'exécution : sim2chro

Récupérer le fichier `NomDeTrace` de l'étape précédente et supprimer la première ligne : `(-Pollux)`.

On peut faire les 2 étapes en une seule par :

```
luciole nomfich.lus nomnoeud -v | sed -e '1,1d' > nom_trace
```

Puis générer le chronogramme en Postscript par :

```
sim2chro nomsortie.ps -in nomtrace -ecran -postscript
```

3.2 Caractérisation d'automate

Obtention et minimisation : lustre, ocmin

La commande :

```
lustre  NomDeFichier.lus   NomDeNoeudEnglobant
```

donne la description de l'automate déduit du circuit dans un fichier `NomDeNoeud.oc`.

`ocmin NomDeNoeud.oc` donne l'automate minimal.

Un automate à un seul état est un système combinatoire.

Pour exécution : lux

```
lux NomDeNoeud.oc  (ou  lux NomDeNoeudMin.oc)
```

donne un programme C et un exécutable dont le comportement est équivalent à l'automate décrit.

Pour lecture seule : vieuxlux, clisible

La suite des commandes :

```
lustre  NomDeFichier.lus  NomDeNoeudEnglobant -oc2
vieuxlux NomDeNoeud.oc
clisible NomDeNoeud.c
```

donne une version très "jolie" et très compréhensible du texte C.
On peut ajouter un environnement à la procédure obtenue ...

4 Autres éléments du langage

Il existe en Lustre des entiers. `pre` sur un entier correspond à un registre ...
Ci-dessous un exemple de noeuds utilisant le type entier (`int`).

```
node maxentiers (a, b : int) returns (ma : int);
let
  ma = if (a > b) then a else b;
tel;

node sommateur (a : int) returns (b : int);
let
  b = 0 -> a + pre b ;
tel;
```