LASSO : un langage pour la description et la simulation de systèmes informatiques : mise en œuvre

Jean-François Grabowiecki

CONSERVATOIRE NATIONAL DES ARTS ET METIERS
CENTRE AGREE DE GRENOBLE
(C.U.E.P.A.)

MEMOIRE

présenté en vue d'obtenir
LE DIPLÔME D'INGENIEUR CNAM

en
INFORMATIQUE

par
Jean François GRABOWIECKI

L A S S O :
UN LANGAGE POUR LA DESCRIPTION ET LA SIMULATION DE SYSTEMES INFORMATIQUES
– MISE EN OEUVRE –

Soutenu le 31 mars 1981

Président : L. BOLLIET, Professeur Université Grenoble II

Membres : D. BORRIONE, Attachée de Recherche CNRS, IMAG, Grenoble
P. CHOMAT, Ingénieur CNAM, IUT II Informatique, Grenoble
M. DELAUNAY, Ingénieur AM, Labo IMAG, Grenoble
J. MERMET, Maître de Recherche CNRS, IMAG, Grenoble
P. NAMIAN, Professeur CNAM, Paris

Je voudrais remercier,

Monsieur le Professeur L. BOLLIET *qui a bien voulu présider au jury de cette thèse et qui m'a toujours encouragé à sa rédaction,*

Monsieur le Professeur P. NAMIAN *qui a accepté d'être le président adjoint du jury,*

Madame D. BORRIONE *qui est à l'origine du projet LASSO et qui en a dirigé le déroulement depuis la définition du langage LASSO jusqu'à sa mise en oeuvre,*

JEAN MERMET *qui nous a toujours conseillé et soutenu tout au long de ce projet,*

MICHEL DELAUNAY *avec qui les nombreuses séances de travail ont permis de profiter de toute son expérience dans le domaine de la compilation,*

PAUL CHOMAT *qui m'a soutenu dans la rédaction de cette thèse,*

JOSIANE CARRY *et le* SERVICE TIRAGE DE L'I.M.A.G.

LASSO :

UN LANGAGE POUR LA DESCRIPTION ET LA SIMULATION DE SYSTÈMES INFORMATIQUES

- MISE EN OEUVRE -

CHAPITRE TROIS : Le simulateur LASSO

INTRODUCTION

Le domaine dans lequel nous nous plaçons est celui de la construction d'outils d'aide à la conception des circuits logiques; ces outils offrent au concepteur le moyen de décrire ses circuits et d'aller aussi loin que possible dans leurs validations par des méthodes logicielles (analyses de cohérences, simulations de fonctionnement, tests, simulations de pannes etc ...) avant d'en faire une réalisation physique.
Plus précisément nous avons abordé les aspects de validation de fonctionnement et d'évaluation de performances par simulation.
La construction d'un nouvel outil, le système LASSO, a été motivée par notre participation au projet de Définition et d'Evaluation d'un système multiprocesseur [34] mené à l'E.N.S.I.M.A.G. Dans le cadre de ce projet nous devions réaliser deux études :

. Dans un premier temps, décrire et simuler des organisations d'architectures multi-processeurs en vue de déterminer, en fonction de ses performances, la configuration la plus apte à répondre aux critères des concepteurs. Le langage LASCAR développé au sein de l'équipe par D. BORRIONE [6] fut notre outil de modélisation. Le niveau de description fin adopté permettait de spécifier la micro-synchronisation lors du décodage des instructions; cependant les temps de simulation importants ne permettaient pas de faire fonctionner l'architecture décrite sur de longues traces d'instructions.

. Une fois terminées les évaluations fines de l'architecture retenue, nous devions dans un deuxième temps étudier le problème du parallélisme et plus précisément étudier la possibilité d'occuper efficacement à tout instant le maximum de processeurs.

Pour cette étude il n'était pas nécessaire de décrire de manière très détaillée l'architecture à simuler; par contre le modèle devait être suffisamment abstrait pour que l'on puisse étudier son comportement sur des programmes très volumineux. Il apparaissait que le langage LASCAR ne pouvait pas décrire des modèles au niveau auquel nous voulions nous placer. Après un tour d'horizon des langages existants et susceptibles de nous apporter une aide nous étions rapidement conduits à définir et construire un outil spécifique [33], [35]. La réalisation de cet outil était faite à partir du compilateur du langage LP15 et du simulateur de l'ordina-

teur MITRA 125; la disponibilité de cet outil n'était pas immédiate et le coût de son développement était important; mais nous disposions d'un système très adapté au problème que nous avions à traiter.

De cette réalisation est né le projet de définir un langage général (nommé LASSO) permettant de décrire rapidement des modèles de circuits.
Ce langage, tout en étant proche des préoccupations du concepteur, devait répondre aux objectifs suivants :

- conserver la notion de modularité de LASCAR
- faciliter l'expression des interactions entre les composants d'une description
- fournir des primitives de lancement d'actions asynchrones et des primitives de synchronisation
- offrir les moyens d'espionner automatiquement le modèle en cours de fonctionnement.

Après avoir participé à la définition du langage LASSO et à l'analyse des premières options de mise en oeuvre nous avons eu la charge complète de la réalisation du système prototype.

Après une présentation très informelle du langage LASSO dans le premier chapitre du mémoire, nous justifirons au chapitre deux la structure générale du système réalisé et donnerons les raisons qui nous ont conduits à prendre certaines solutions de mise en oeuvre.

Le chapitre trois présente l'organisation du simulateur et la structure de donnée interne représentant une description.

Le chapitre quatre est consacré au compilateur du langage de description LASSO.

CHAPITRE PREMIER

PRÉSENTATION DU LANGAGE LASSO

I - INTRODUCTION

Le domaine dans lequel nous nous plaçons est celui de l'aide à la conception des systèmes logiques.

→ La nécessité d'un découpage : tout système, en particulier un système logique de taille importante doit pouvoir être décomposé en "modules" indépendants pour plusieurs raisons :

1) de par la quantité d'informations que représente un tel système, il est difficile au concepteur d'appréhender, d'un seul coup, le système complet; d'où la nécessité de le fractionner.

2) il serait en outre très coûteux de vouloir mettre au point globalement un modèle constitué de plusieurs "modules" (en vue d'une simulation par exemple).
Dans cette optique, chaque module doit pouvoir être conçu et vérifié indépendamment des autres "modules" de la description.

3) de par la nature même du sujet, la décomposition en modules d'une description reflète dans la plupart des cas un découpage structurel et/ou fonctionnel du système réel que le concepteur essaie de représenter.

Cette notion de découpage des langages CASSANDRE [2] et LASCAR [6] développés au sein de l'équipe, a été reprise dans la définition de LASSO car nous avons voulu développer un nouvel outil qui garde une certaine cohérence avec ceux déjà existants. La notion d'unité présente dans CASSANDRE et LASCAR a été étendu à la notion d'entité paramétrable, ce qui offre le moyen de définir des classes d'unités.

Une description en LASSO est un ensemble d'unités interconnectées. Une unité peut en contenir une ou plusieurs autres et ceci jusqu'à un niveau quelconque d'imbrication. La communication entre unités se fait uniquement via un interface; celui-ci définit les signaux et les données qui peuvent être reçus ou expédiés par l'unité : les interactions entre unités se font au moyen des messages.

→ Le corps d'une entité est composé :

1) d'une partie fonctionnelle décrite par des actions; celles-ci sont une suite d'instructions exécutées séquentiellement.

2) d'une partie controle représentée par un graphe de transition [1], [4] permettant de lancer des actions en parallèle et de les synchroniser. Ceci est une généralisation de la notion d'automate de CASSANDRE et LASCAR.

Elevant le niveau d'abstraction des descriptions des systèmes logiques, LASSO devient un langage de spécification pour CASSANDRE et LASCAR. Il permet alors de réduire les temps de simulation.

→ Enfin une partie mesure peut être adjointe à toute unité du modèle. Cette partie (optionnelle) permet d'exprimer, à l'aide d'un langage dont la définition et les principes sont exposés dans [7], [8], [27], des actions d'espionnage automatiques en cours de simulation de l'unité. Les résultats de ces actions sont rangés dans un fichier sur disque en vue d'une exploitation ultérieure hors simulation.

L'objectif de la présentation très pragmatique du langage LASSO, est d'exposer les principales primitives du langage et leur utilisation dans une description. Pour une définition plus formelle du langage, on se rapportera à [26], [28], [30, (TOME 3)].

Notations Par la suite, dans la présentation des exemples, les mots réservés du langage sont soulignés, les commentaires sont écrits entre %...%.

II - NOTIONS DE BASE ET STRUCTURE D'UNE ENTITE

II-1 Notions d'entité et d'unité

Une description LASSO est un module composé de sous-modules. Chaque sous-module peut être lui-même décomposé en sous-modules plus élémentaires.

En LASSO un module ou sous-module est appelé unité. Chaque unité est issue de la définition préalable d'une entité qui la décrit. Une entité paramétrée autorise donc la définition de familles d'unités; une unité est un exemplaire particulier d'une entité pour laquelle tous les paramètres formels (s'ils existent) ont reçu une valeur.

Exemple 1 :

```
entité processeur;
     :
fin;   % entité processeur %
unité process : processeur; % déclaration de l'unité process %
```

Exemple 2 :

```
entité molécule(entier nbprocesseurs);
  :
fin;   % molécule %
  :
unité multiprocesseur : molécule(8);
        % déclaration de l'unité multiprocesseur contenant 8 processeurs %.
```

II-2 Structure générale d'une entité LASSO

Une entité LASSO comprend :

une entête d'entité qui déclare le nom de l'entité suivi de ses paramètres formels optionnels.
entité identificateur (paramètres formels);
Les paramètres peuvent être de type entier, caractère alphanumérique ou booléen.

L'entête est suivie des parties interface, corps et mesure dans cet ordre.

L'interface contient :

- les déclarations des objets d'interface
- les déclarations des signaux d'entrée ou de sortie.

Dans le corps on trouve :

a) Les déclarations :
   - des variables (tableau ou scalaire) internes à l'entité (entiers, booléens, caractères, signaux, nomsig)
   - des entités externes
   - des procédures et fonctions externes
   - des unités.

b) L'expression des connexions

c) La description des transitions.

Dans la partie mesure on trouve :

- des déclarations de compteurs
- les expressions des mesures.

Explicitons en détail chacune des 3 parties.

III - L'INTERFACE

Il constitue le seul moyen de communication de l'entité avec son environnement. Il contient :

1) la déclaration de variables (scalaires ou tableaux) de type booléen, caractère alphanumérique ou entier

2) suivie de la déclaration de signaux unidirectionnels entrée ou sortie.

Un signal (d'entrée ou sortie) peut à lui seul avoir une signification complète soit de requête envoyée à une autre unité soit d'acquittement pour un travail réalisé. Cependant dans la plupart des cas de circuits logiques un signal est accompagné d'une ou plusieurs données associées. L'association signal (données) forme dans LASSO un message qui est transmis globalement vers l'entité destinataire. Une même donnée peut être associée à la fois à un signal d'entrée et un signal de sortie et est bidirectionnelle.

Exemple :

```
entité nom;
interface
   entier a, b [1 : 8];
   bool c;
   entrée e; % signal seul %
   entrée e1(a, c); % le signal e1 lorsqu'il est reçu indique que les valeurs des
                      variables a et c sont disponibles %
   sortie s(c, b); % le signal s et les données associées c et b sont transmis
                     globalement vers l'entité destinataire %.
```

La réception ou l'émission d'un signal d'interface est matérialisée par son positionnement à la valeur 1.

Un signal d'entrée reçu conserve sa valeur 1 jusqu'à sa prise en compte qui le positionne à 0.

Un signal de sortie positionné à 1 dans le corps d'entité est transmis (avec ses données associées) vers l'entité destinataire. Après la transmission il est repositionné à 0.

IV - LE CORPS DE L'ENTITE

Il contient la déclaration de tous les objets locaux à l'entité, l'expression des connexions entre les unités fils directs de l'entité, la description du fonctionnement de la partie contrôle. Ces 3 parties doivent se succéder obligatoirement dans l'ordre décrit ci-dessus.
Notons que dans la partie déclaration des objets locaux l'ordre est quelconque avec la contrainte suivante : la déclaration des entités doit préceder la déclaration des unités qui en dérivent.

IV-1 Déclarations des variables

Les déclarations des variables internes de type booléen, caractère ou entier se font de manière similaire à celles de l'interface.
Les variables de type signal et nomsig demandent quelques précisions :

. Les variables de type signal sont représentées par des valeurs booléennes. Le passage de 0 à 1 d'un signal indique l'occurrence d'un évènement. Le passage de 1 à 0 indique que cet évènement a été pris en compte.

Un signal déjà positionné à 1 ne peut pas recevoir une nouvelle valeur 1 (pas de perte d'informations).

. Les variables de type nomsig permettent une écriture plus souple de certaines descriptions. Ces variables peuvent être chargées ou initialisées avec des identificateurs de signaux (entrée, sortie, signal) uniquement.

. Toutes les variables locales peuvent être initialisées à la compilation à l'aide de la directive init.

Exemple :

```
  :
corps
   signal s;
   signal s1, s2[0 : 7] % s2 est un tableau de 8 signaux %
  :
   nomsig N, N1[0 : 3] init(s, s1, s2, s);
         % Le tableau N1 est initialisé aux 4 valeurs s, s1, s2, s %
```

IV-2 Déclarations de procédures et fonctions

Toute référence à une procédure dans une entité doit être précédée de la déclaration de son entête avec ses paramètres formels éventuels. Ces paramètres peuvent être des scalaires ou des tableaux de type entier ou caractère ou booléen. Lors de l'invocation d'une procédure une vérification de compatibilité entre paramètres formels et paramètres effectifs est faite.
La procédure ou la fonction doit être sous forme de texte source PASCAL [5] dans un fichier bibliothèque.

Exemple :

```
entité processeur;
   :
corps
    fonction externe f(entier d) résultat : entier;
    procédure externe p(entier c, var : bool b);
   :
fin; % entité processeur %
```

. Lors de la déclaration d'une fonction le type du résultat fourni doit être précisé.
. Le qualificatif var indique un passage de paramètre par référence.

IV-3 Déclarations d'entités

Une description LASSO est un ensemble d'unités interconnectées. Chaque unité est issue d'une entité préalablement définie et dont le texte peut être :

- locale à l'entité : il fait partie dans ce cas du texte de la description
- externe au modèle : le texte se trouve alors dans une bibliothèque d'entités et sera changé par l'utilisateur du système.

Dans la version prototype actuelle les entités sont considérées comme externes et leur texte source doit précéder toute description les référençant.

Exemple :

```
entité a;
  :
fin; % entité a %
  :
entité b;
  :
  corps
    :
  entité externe a; % le texte source de l'entité a doit précéder la définition de
                      l'entité b %
    :
fin; % entité b %
```

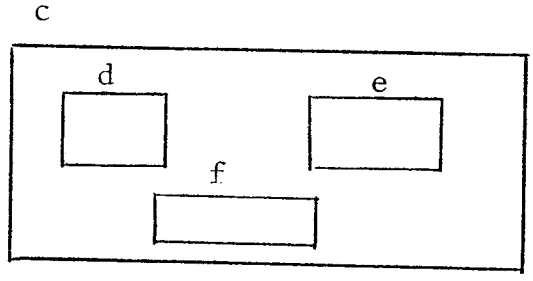
IV-4 Déclarations d'unités

C'est la définition d'un exemplaire particulier d'une entité.

Exemple :

```
entité a;
   :
    corps
   :
fin; % a %
entité b;
   :
   corps
     :
fin : % entité b %
entité c;
   :
   corps
        entité externe a;
        entité externe b;
        unité d, e : a;
          :   % d, e sont 2 exemplaires identiques de l'entité a définie aupara-
          :     vant %
        unité f : b;
   :
fin; % entité c %
```

La structure d'imbrication de la description générée est la suivante :



d, e, f, sont 3 fils directs de c.

IV-5 Les connexions d'unités

L'ensemble des unités d'une description LASSO fonctionne d'une manière autonome et asynchrone. Cependant chaque unité a la faculté de correspondre via les signaux d'interface (cf paragraphe III) :

1) avec une unité de même niveau d'imbrication
2) avec l'unité de niveau immédiatement englobante.

L'établissement des liaisons entre unités se fait à l'aide des instructions de connexions qui établissent un lien permanent et à tout instant entre les unités. Les instructions de connexions entre 2 unités de même niveau s'écrivent au niveau de l'unité immédiatement englobante qui seule a accès aux variables d'interface des unités fils par notation 'pointée'.

Suivant les liens établis, les transmissions des signaux ou des messages entre unités peuvent être instantannées ou retardées.

Exemple :

Décrivons les interfaces des entités a et b de l'exemple du paragraphe précédent.

```
entité a;
interface
  entier interruption;
  bool traite;
  entrée reçu(interruption);
  sortie fait(traite);
    :
fin; % a %
entité b;
  interface
    entier interrupt;
    bool termine;
    entrée e(termine)
    sortie s(interrupt)
      :
fin; % b %
```

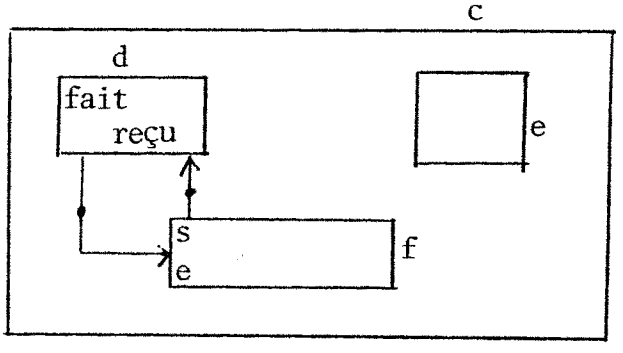
Etablissons des connexions entre les unités d et f de la description; on peut écrire dans l'entité c :

```
entité c;
  :
corps
  :
    % connexions %
    f_e ← d_fait; %(1) %
    d_reçu ← f_s retard(3); %(2) %
  :
fin; % c %
```

. La connexion (1) est instantannée et exprime le fait que lorsque le signal 'fait' passe à 1, il sera transmis instantanément avec sa donnée associée booléenne 'traite' à l'unité f.

Le signal correspondant (dans l'unité f) sera positionné à 1 et la donnée associée 'termine' sera initialisée avec le contenu de la variable 'traite'.

. La connexion (2) est retardée et exprime le fait que la transmission du signal emetteur prend un temps non nul. Le signal et la donnée associée seront disponibles dans l'unité d'après 3 unités de temps.



IV-6 Les actions et les primitives de synchronisation

L'interprétation des signaux d'entrée et des données associées d'une unité déclenche l'exécution d'une ou plusieurs actions séquentielles ou parallèles.

IV-6-1 Les actions

Elles correspondent à la partie opérative d'un circuit et permettent d'opérer des transformations sur les contenus des variables locales et d'interface d'une entité. Elles sont décrites par un ensemble d'instructions s'exécutant séquentiellement et sans interruption. Cet ensemble est préfixé par une condition de signal entre points d'interrogation et se termine par une validation de signal.

Exemple :

```
? signala ? début
            :
          liste d'instructions
            :
          fin délai(3)
          valider(signalb);
```

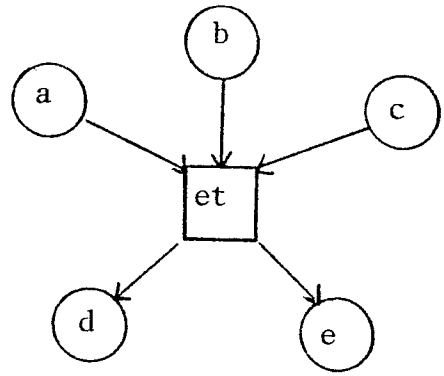
Si signala est positionné à 1 et si signalb est à 0, les instructions sont exécutées; signala est remis à 0 et signalb est positionné à 1.
Dans l'exemple la primitive délai indique que l'action dure 3 unités de temps. Si rien n'est indiqué l'action est exécutée instantanément.
Les instructions exécutées dans les actions sont les instructions classiques des langages de programmation de type PASCAL; on trouvera :

. l'affectation (:=), . l'appel de procédure ou fonction, . les instructions composées (début ... fin),
. l'instruction conditionnelle (si ...),
. les instructions d'itérations (pour, pourtant),
. les instructions d'itérations conditionnelles (tantque, repeter)

IV-6-2 Le contrôle

Il sert à positionner les signaux de sortie d'unité et les signaux conditionnant les actions. Ce positionnement se fait en fonction des signaux d'entrée ou internes, des valeurs de certaines variables d'interface ou des valeurs de certaines variables locales à l'entité. Nous représenterons le contrôle par un graphe dans lequel les signaux sont des places et les transitions des boîtes. La primitive délai a ici la même signification que dans les actions.

A) La transition 'et'

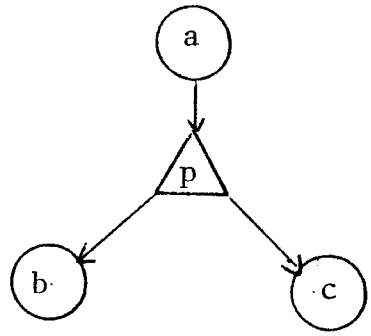


syntaxiquement s'exprime par :

?a et b et c? délai(4) valider(d, e);

Si tous les signaux en sortie de la transition sont à 0 et si tous les signaux en entrée de la transition sont à 1, ces derniers sont tous remis à 0 et tous les signaux en sortie sont mis à 1.
Dans le cas contraire, la transition n'est pas effectuée (attente). Le nombre de signaux en entrée et en sortie est quelconque.

B) La transition 'selon'

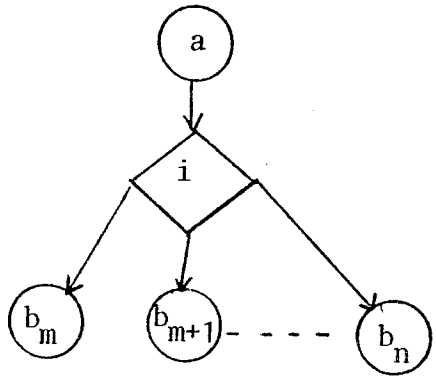


syntaxiquement s'exprime par :

?a? selon p délai(1) valider(b, c);

p est une expression booléenne calculée à partir de variables internes ou d'inter face (tests de valeurs). Si les signaux b et c sont à 0, et si le signal a est à 1, p est calculé. Si p est vrai, b est validé, sinon c est validé. Dans les deux cas, a est remis à 0.
Cette transition peut être utilisée comme un aiguillage à 2 positions, auquel est associée une fonction de décision.

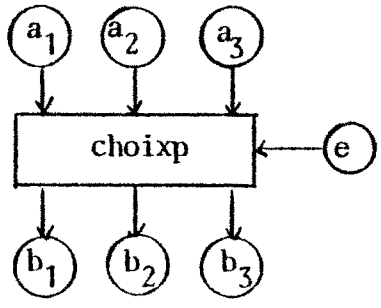
C) La transition 'index'



?a? index i de m a n délai(3) valider(b_m, b_{m+1} ... b_n);

i est une expression entière scalaire appartenant à l'intervalle fermé [m, n]. Le nombre de sorties doit être égal à n-m+1.
Si toutes les sorties sont à 0 et si a a la valeur 1, i est calculé et sert à indexer la place de sortie mise à 1.
a est remis à 0.

D) La transition 'choixp'

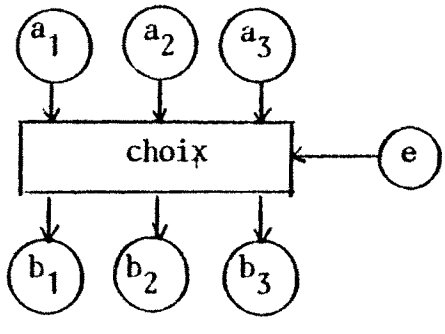


syntaxiquement, s'exprime par :

?e? choixp(a1, a2, a3) délai(4)
valider(b1, b2, b3);

Cette transition correspond au test de plusieurs signaux dont la priorité va décroissant de la gauche vers la droite, en vue de choisir le signal valide de plus forte priorité.
Si toutes les sorties sont à 0 et si e a la valeur 1, la transition sélectionne, parmi toutes ses autres entrées à 1, celle de plus forte priorité. La sortie de même rang est mise à 1, et e est l'entrée sélectionnée sont remis à 0.

E) La transition 'choix'

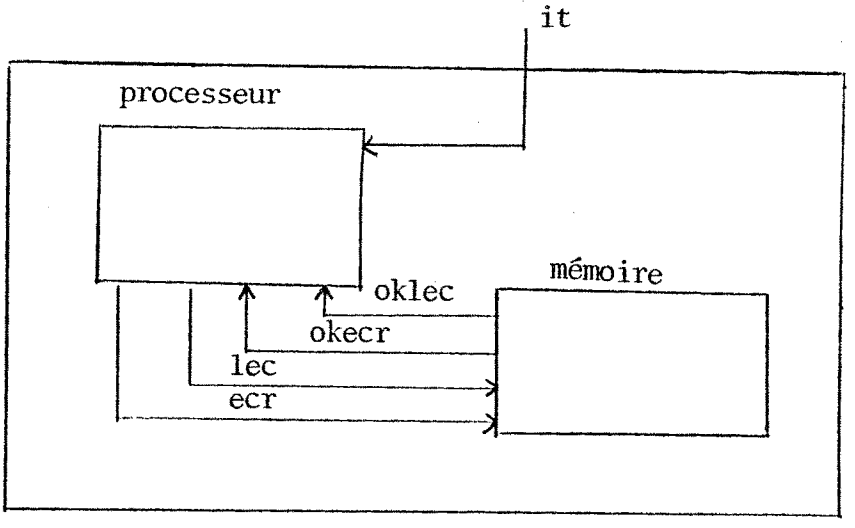


?e? choix(a1, a2, a3) délai(n+2) valider(b1, b2, b3);

Cette transition ressemble à la précédente, en ce sens qu'à chaque entrée (exceptée l'entrée e, qui a un rôle particulier), correspond une place de sortie. Mais ici, les entrées ont une égale priorité, et une seule à la fois peut être valide (dans le cas contraire, il y a erreur). Si toutes les sorties sont à 0 si e vaut 1 et si l'une, et une seule des autres entrées vaut 1, la sortie correspondante prend la valeur 1 et l'entrée sélectionnée et e sont remis à 0.

IV-7 Un exemple d'entité

L'entité décrite ici est un processeur relié à une mémoire, le tout sous le contrôle d'un système superviseur.
Pour des raisons de simplicité nous ne décrivons ni la mémoire, si le système; cependant dans l'entité processeur l'interface est décrit en vue de ces connexions.



Le processeur possède un compteur ordinal co et un compteur i du nombre d'écritures mémoire.

Il est capable d'exécuter 2 types d'instructions :

- les instructions à opérande immédiat appelées courtes
- les instructions à adressage mémoire direct ou indirect appelées longues.

Les demandes de lecture/écriture sont envoyées à la mémoire qui répond par oklec ou okecr respectivement; l'appel à la procédure paramétrée 'alerte' permet d'imprimer sur un dispositif quelconque le message sélectionné par la valeur du paramètre.

```
entité processeur;
  interface
    entier ad, % adresse mémoire pour une lecture/écriture % donnée, % tampon
           d'entrée/sortie %
           itextrn, % numéro interruption externe %
           arrêtgénéral; % système en panne %
```

```
      entrée
            oklec(donnée), % acquittement après demande de lecture %
            okecr, % acquittement après demande d'écriture %
            it(itextrn), % interruption externe %
            stop(arrêtgénéral), % arrêt du processeur %
      sortie
            lec(ad), % demande de lecture %
            ecr(ad, donnée); % demande d'écriture %
   corps
procédure externe alerte(entier noit);
signal panneprocess, pannememoire, étatdémarche;
signal testit,arrêtcomplet, lireinst, ok,
                test1, courte, calculad,
      adindirect, addirect, liread, lireop, jump;
signal ecrop, longue;

bool immédiat; % instruction avec opérande immédiat %
entier i, % nombre d'écritures mémoire exécutées %
      co, % compteur ordinal %
      ri, % registre instruction %
      codeop, % code opération %
      indirect, % bit d'indirection %
      adr, % adresse opérande %
      accu; % accumulateur %
      % les transitions %
?testit? index itextrn valider(panneprocess, pannemémoire, étatdémarche);
        % décodage de l'interruption externe %

?panneprocess? début
            % traitement de la panne %
            alerte(itextrn); % appel d'une procédure d'impression de message %
            arrêtgénéral := 3; % code erreur interne %
            fin
            valider(arrêtcomplet);

?arrêtcomplet? début
            alerte(arrêtgénéral); % impression message d'arrêt du système %
            fin;
```

```
?étatdemarche? choixp(stop, it, ok)
               valider(arrêtcomplet, testit, lectureinst);

?lectureinst? début % demande de lecture instruction %
               ad := co;
               co := co + 1;
               fin valider(lec, lireinst, ok);

?oklec & lireinst? début % récupération instruction %
            ri := donnée;
            codeop := ri/2**9; % 3 bits poids forts %
            adr := ri rem 2**8; % 8 bits poids faibles %
            indirect := (ri/2**8) rem 2; % 9ème bit à partir de la droite %
            si codeop = 7 alors immédiat := 1
               sinon immédiat := 0;
                     fin valider(test 1);

?test1? selon(immédiat) valider(courte, calculad);
      % si immédiat = 1, instruction sans référence mémoire; sinon calcul adresse
        opérande nécessaire %

?calculad? selon(indirect) valider(adindirect, addirect);
         % test du bit d'indirection %

?adindirect? début
            ad := adr;
            fin valider(lec, liread);

?oklec & liread? début
            adr := donnée, % récupération adresse indirecte %
                fin valider(addirect);

?addirect? index(codeop) valider(lireop, lireop, lireop, lireop, ecrop, lireop,
          jump); % décodage des instructions longues %

?lireop? début
        ad := adr
        fin valider(lec, longue);
```

```
?ecrop? début % écriture en mémoire %
        donnée := accu; % donnée à écrire %
        ad := adr; % adresse mémoire %
        i := i+1;
        fin valider(ecr);

?jump? début % instruction de saut %
       do := adr
      fin délai(2) valider(étatdemarche);

?okecr? valider(étatdemarche);

?courte? délai(2) valider(étatdemarche);
        % instructions courtes exécutées en 2 unités de temps; on ne détaille pas
          les opérations effectuées %

?oklec & longue? délai(6) valider(étatdemarche);
        % on ne détaille pas les opérations effectuées %

fin; % processeur %
```

V - LA PARTIE MESURE

Elle offre le moyen d'exprimer des prises de mesures automatiques qui seront effectuées sur une entité lors de sa simulation.
Ces prises de mesures peuvent être :

- soit des relevés de valeurs de variables au cours du fonctionnement de l'entité
- soit des tracés d'évolution du graphe de contrôle de l'entité.

La partie mesure est optionnelle et peut être associée à une entité à l'aide du mot clef avec qui suit le dernier fin; de l'entité.
Les mesures comprennent une partie déclaration de compteurs suivie de la partie ordres de prises de mesures.

Exemple :

```
entité
  :
fin; %
avec
  :
   déclarations des objets locaux
   ordres de prises de mesures
;
```

V-1 Déclarations de compteurs

Ce sont des variables entières de portée limitée à la partie mesure et de valeur initiale 0.

Exemple :

```
compteur k1, k2;
```

V-2 Les ordres de prises de mesures

V-2-1 Les opérations sur les compteurs

- Incrémentation de 1 d'un compteur :

```
inc(k1);
```

- Remise à 0 d'un compteur :

```
init (k2);
```

V-2-2 Les opérations de relevé de valeurs

Elles permettent de relever les valeurs de compteurs et de variables de l'entité en les écrivant dans un fichier disque exploitable hors simulation.

- Relevé de valeurs de compteurs

```
relever(k1);
```

Cette opération écrit sur le fichier :

. l'identificateur k1
. sa valeur
. la date du relevé.

. Relevé de valeurs de variables

mes(id); % id est une variable de l'entité %

On écrit dans le fichier :

. la date courante
. les noms de la variable et de l'unité où elle est définie
. la valeur de la variable.

V-2-3 Impression d'un texte

date('chaîne de caractères);

Sont écrites dans le fichier :

. la date courante
. la chaîne de caractères.

V-2-4 Condition restrictive

Les opérations précédentes peuvent n'être exécutées que si certaines conditions portant sur des variables de l'entité sont vérifiées. On emploie l'instruction conditionnelle classique

si <condition> alors <opérations mesures>
sinon <opérations mesures> finsi

(la partie sinon est optionnelle).

Exemple :

```
si id = 100 alors inc(k1)
        sinon init(k1) finsi;
```

V-3 Détermination des instants de prises de mesures

L'activation des opérations inc, init, relever, mes, date, peut être effectuée soit à l'occurrence d'un évènement, soit à des intervalles réguliers de temps au cours de la simulation.

V-3-1 Les évènements élémentaires

Un évènement élémentaire est le passage de 0 à 1, ou le passage de 1 à 0 d'un signal.
La notation est la suivante :

*1 suivi d'un identificateur de signal, exprime l'évènement passage de 0 à 1 du signal.

*0 suivi d'un identificateur de signal, exprime l'évènement passage de 1 à 0 du signal.

Exemples :

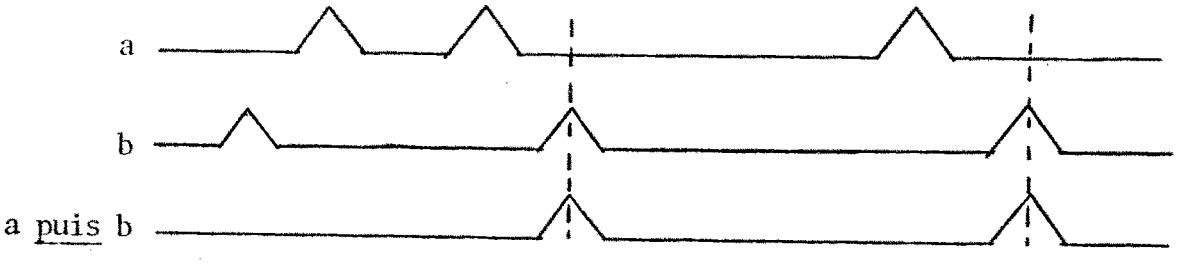
*1s est un évènement qui sera vrai quand le signal s transitera de 0 à 1.
*0t est un évènement qui sera vrai quand le signal t transitera de 1 à 0.

V-3-2 Les expressions d'évènements

On appelle expression d'évènements, l'évènement résultant de la combinaison de plusieurs évènements élémentaires. 3 opérateurs ont été définis, admettant des évènements comme opérandes.

A) L'opérateur puis

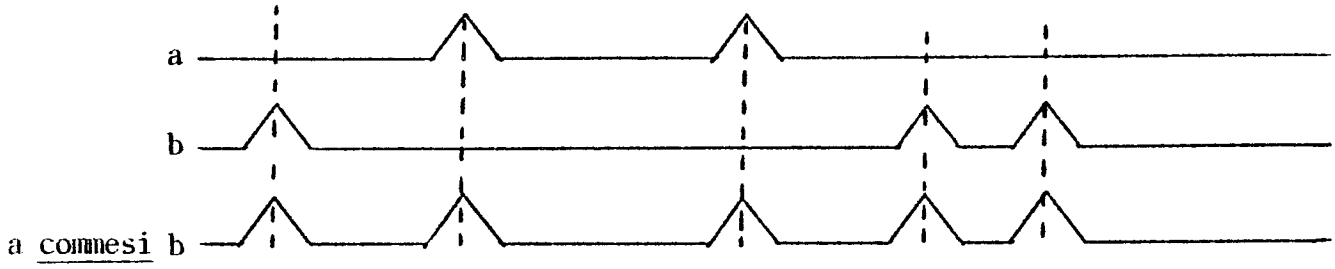
Cet opérateur exprime la séquence de deux évènements; le résultat est un évènement qui se produit chaque fois qu'il y a eu au moins une occurrence du 1er opérande, suivi d'une occurrence du 2ème opérande.



Remarque : on ne mémorise pas le nombre de validations de l'évènement a, rencontrées avant la validation de l'évènement b.

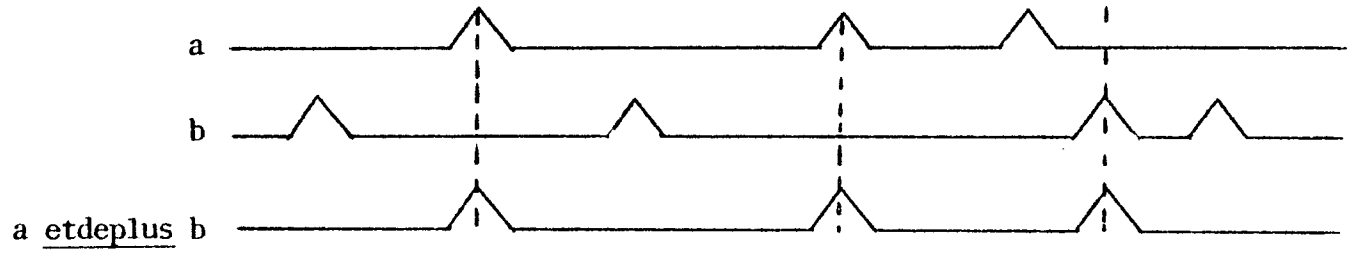
B) L'opérateur commesi

Cet opérateur engendre un évènement chaque fois que l'un ou l'autre des deux opérandes se produit; c'est en fait l'union ensembliste des évènements décrits par les deux opérandes.



C) L'opérateur etdeplus

Cet opérateur exprime conjointement l'occurrence de deux évènements, sans considérer l'ordre dans lequel ces deux évènements sont validés, comme le montre le chronogramme suivant :



Remarque :

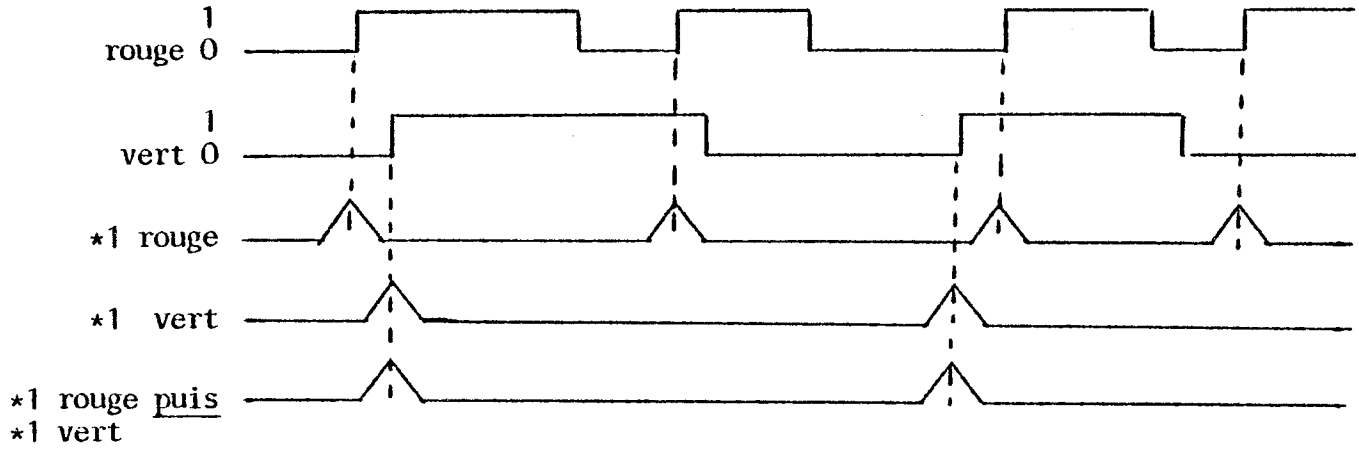
a etdeplus b ≠ (a puis b) commesi (b puis a)

Ceci est dû au fait que, lors de l'évaluation de l'expression "a etdeplus b", on ne mémorise pas le dernier évènement validé.

Les 3 opérateurs cités ont la même priorité.

Exemple :

Soient 2 signaux rouge, vert déclarés dans une entité; l'évènement : *1 rouge puis *1 vert est traité suivant le diagramme :



V-3-3 Détermination des instants effectifs de prises de mesures

Les instants de prises de mesures peuvent être effectués soit à intervalles réguliers soit à l'occurrence d'un évènement.

Les instructions dèsque, chaque, lafoisno et tousles spécifient ces instants. Chaque instruction peut conditionner plusieurs opérations de prises de mesures;

Exemple :

```
dèsque ev
  :
    opérations de prises de mesures
  :
fin;
```

On note ev un évènement.

A) L'instruction : dèsque ev

Elle sélectionne la première occurrence de l'évènement ev.

Exemple :

```
dèsque  *1 rouge puis  *0 rouge date ('rouge éteint') fin;
```

La 1ère fois que le signal rouge passe à 0 après être passé à 1 on écrit le texte entre apostrophes et la date courante.

B) L'instruction : chaque ev

Repère toutes les occurrences de l'évènement.

Exemple :

```
chaque *1 vert inc(k1) fin;
```

Chaque fois que le signal vert passe à 1 le compteur k1 est incrémenté de 1.

C) L'instruction : lafoisno j que ev

j est une constante entière > 0

Exemple :

lafoisno 13 que *0 vert date ('début arrêt') fin;

Au 13ème passage à 0 du signal vert le texte entre apostrophes et la date courante sont écrits.

D) L'instruction : tousles j

j est une constante entière > 0
Cette instruction repère à partir d'une date t (par défaut la date de début de simulation) tous les instants t + j, t + 2 j .....

Exemple :

tousles 33 date ('respirer') fin;

Toutes les 33 unités de temps on écrira le texte entre apostrophes et la date courante.

V-3-4 Exemple de partie mesure (cf entité décrite au paragraphe IV-7)

```
entité processeur;
  interface
     :
  corps
     :
  fin;
  avec % partie mesures %
   % déclarations de compteurs %
  compteur nblec, % nombre de lectures %
           nbacces; % nombre d'accès mémoire %
  % opérations de prises de mesures %
```

```
% sur front montant du signal 'it' les compteurs sont remis à 0 %
chaque *1 it
    init(nblec, nbacces);
            fin;

% à chaque demande de lecture ou écriture on comptabilise l'accès mémoire %
*1 lec  commesi *1 ecr inc(nbacces) fin;

% sur front montant du signal 'lec' %
chaque *1 lec
    inc(nblec) fin;

% lors de l'arrêt complet du processeur on imprime des valeurs sur le fichier
disque %
chaque *1 arrêtcomplet
    relever(nblec, nbacces); % relevé des compteurs %
    mes(co, i); % relevés de la valeur du compteur ordinal et du nombre d'écri-
                tures en mémoire %
% fin partie mesure %
```

CHAPITRE DEUX

LE SYSTÈME LASSO

I - INTRODUCTION

I-1 Les conditions de réalisation du système LASSO

Nous devions réaliser en 2 ans un système prototype de mise en oeuvre du langage LASSO.
A ce délai s'ajoutait un changement imminent d'ordinateur nous imposant de prévoir un transport rapide et à moindres frais des programmes en cours d'écriture.

Guidés par ces 2 impératifs et l'étude [31] que nous avions faite sur les langages de programmation de haut niveau disponibles nous avons choisi (influencés peut être aussi par l'engouement général pour ce langage) le langage PASCAL installé alors sur l'ordinateur CII IRIS 80.

Ce langage fournit des facilités pour la structuration des programmes et l'expression des algorithmes en offrant au programmeur des structures de contrôle particulièrement agréables. Ce langage fortement typé permet de mettre au point aisément des programmes et offre des commodités pour la manipulation des structures de listes.
Cependant quelques lacunes apparaissent dans les possibilités de manipulations des chaînes de caractères ainsi qu'aux niveaux des entrées/sorties du langage; les types de fichiers autorisés sont uniquement séquentiels ou directs et leur nombre en est limité dans un programme (7).
Enfin d'un point de vue général, il se dégage du langage une certaine rigidité (les différentes parties d'un programme sont dans un ordre imposé) qui cependant induit une certaine discipline non négative au niveau de la programmation.

Nous considérions que les inconvénients de ce langage n'étaient pas majeurs et ne nous ralentiraient pas dans notre réalisation.

I-2 Les choix de mise en oeuvre de LASSO

Une grande partie des choix qui ont été pris pour la mise en oeuvre de LASSO sont issus de l'expérience que nous avons dans le domaine de la construction des outils d'aide à la conception de circuits logiques.

Nous rappelons les 3 principes fondamentaux énnoncés en introduction générale :

1) la modularité d'une description
2) la possibilité de faire fonctionner indépendamment chaque module
3) simuler interactivement un modèle.

qui sont, selon nous les caractéristiques minimales d'un système d'aide à la conception. Pour réaliser ces 3 objectifs nous adoptons le schéma suivant de traitement :

1) Une étape de compilation du langage de description en un code intermédiaire.
2) Une étape interprétation - exécution de ce code intermédiaire correspondant à la simulation.

II - L'ETAPE DE COMPILATION

II-1 Structure de compilation

Les principes 1) et 2) impliquent qu'une description LASSO soit traitée entité par entité. Toute entité précédemment compilée peut être référencée lors de sa connexion à d'autres composants du modèle. Le compilateur doit pouvoir accéder alors aux variables d'interface de l'entité pour établir les connexions; dans ce but un context est associé à chaque entité précompilée contenant :

. l'adresse de sa table des identificateurs
. l'adresse du code intermédiaire généré
. l'adresse de sa table des unités fils directs.

II-2 Choix de la représentation interne d'un modèle

Le code objet interne généré par le compilateur doit :

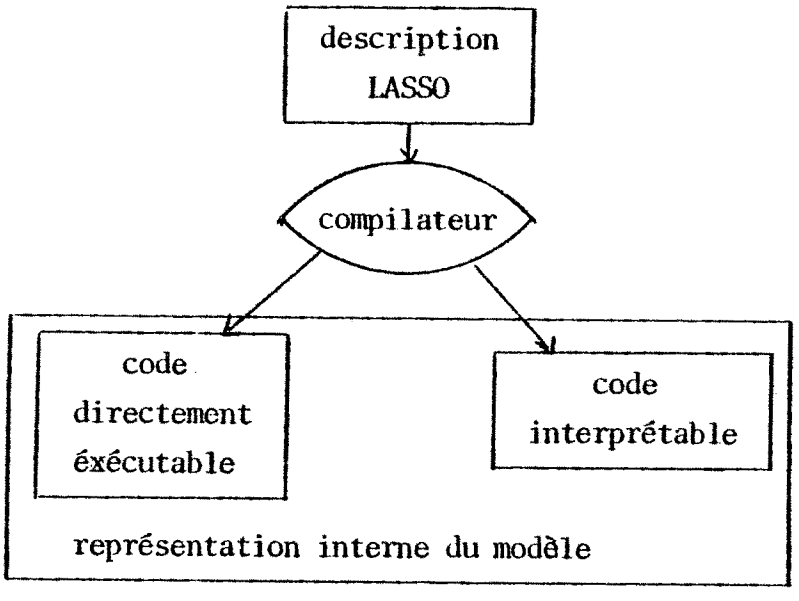
1) représenter le modèle décrit par le concepteur
2) offrir la possibilité de simuler ce modèle.

Plusieurs formes de code objet sont possibles; cependant deux impératifs apparemment incompatibles sont à considérer :

- la simulation doit être interactive
- le code objet doit être efficace et rapide à l'exécution.

Des mesures de performance [6] réalisées sur la simulation de descriptions LASCAR nous ont montré le coût élevé que représente une totale interprétation d'un code; en revanche un code directement exécutable rend les possibilités d'interactions avec le modèle simulé très difficiles à implanter.
Nous optons pour une solution mixte dans laquelle une partie du code est interprétée et une partie est directement exécutée.
La partie du modèle interprétée correspond aux primitives du graphe de contrôle, aux connexions et, dans la partie mesure, aux instants de prises de mesures (cf chapitre premier).

Le code directement exécutable correspond dans une description aux actions (code algorithmique) et, dans la partie mesure, aux instructions de prises de mesures effectives.



II-3 Structure du compilateur et choix du code généré

Le compilateur LASSO réalisé comporte 2 phases distinctes :

- la phase d'analyse syntaxique classique
- la phase génération de code.

Nous avons utilisé, pour réaliser la première phase, un outil d'aide à la construction automatique de compilateurs décrit au chapitre 4.

Par contre nous ne disposons pas d'outils d'aide à la construction automatique de générateurs de code bien que des recherches importantes soient faites dans ce domaine [9], [10], [11], qui aboutissent à des techniques de constructions différentes; nous en citerons 3.

- Génération d'un code intermédiaire

Il s'agit de fragmenter la génération de code en 2 étapes; dans une première étape un code intermédiaire est généré comme par exemple le P-Code [15]; dans une deuxième étape ce code est interprété pour la production du code machine.

- Structuration du compilateur et construction d'un générateur de code paramétrable [19]

Il s'agit dans ce cas d'une compilation en 3 niveaux. Un premier niveau conduit du code source à un langage intermédiaire LI_1 indépendant de toute machine; le second niveau traduit LI_1 en un code LI_2 dépendant d'une famille de machines; le troisième niveau permet à partir de LI_2 de générer le code objet pour la machine réelle. Le générateur de code machine est alors paramétré par les descriptions des langages source et cible.

- Génération automatique d'un générateur de code [22]

Il s'agit ici de décrire la machine cible à l'aide d'un langage de description à partir duquel est créé automatiquement le générateur de code pour cette machine.

Cependant dans les 3 cas cités nous devions soit écrire un interpréteur du code intermédiaire soit disposer du langage de description et de son compilateur ce qui n'apparaissait pas aisément réalisable.
Notre objectif étant de disposer rapidement d'un prototype; nous avons choisi de générer du code de haut niveau PASCAL syntaxiquement et sémantiquement correct.

III - L'ETAPE DE SIMULATION

III-1 Simulateur évènementiel

Le simulateur à mettre en oeuvre doit répondre à 3 objectifs :

- être interactif
- pouvoir interpréter une partie du modèle interne LASSO
- pouvoir lancer l'exécution de portions de programmes faisant partie d'une description.

En outre les possibilités offertes par le langage (expressions de retards dans les actions, les connexions et les primitives de contrôle) nous ont conduits à réaliser un simulateur de type évènementiel.
La rapidité d'exécution d'un tel simulateur est très dépendante de la représentation interne des évènements et des algorithmes de gestion associés. On trouvera dans [23] et [24] des études et des propositions d'algorithmes pour une gestion efficace des évènements; cependant la spécificité de notre application nous a conduits à disposer d'une structure à 2 échéanciers pour gérer les évènements d'un modèle :

. un échéancier global pour tout le modèle indiquant, pour chaque unité présente dans l'échéancier, sa prochaine date d'activation,

. des échéanciers locaux (un échéancier par unité du modèle), indiquant pour chaque unité, la liste des blocs évènements à activer.

III-2 Structure fonctionnelle du simulateur

En plus des tâches de gestion des évènements et de communication avec le concepteur, le simulateur contient son propre outil de mise au point.

Il peut être fonctionnellement décomposé en 5 grandes parties :

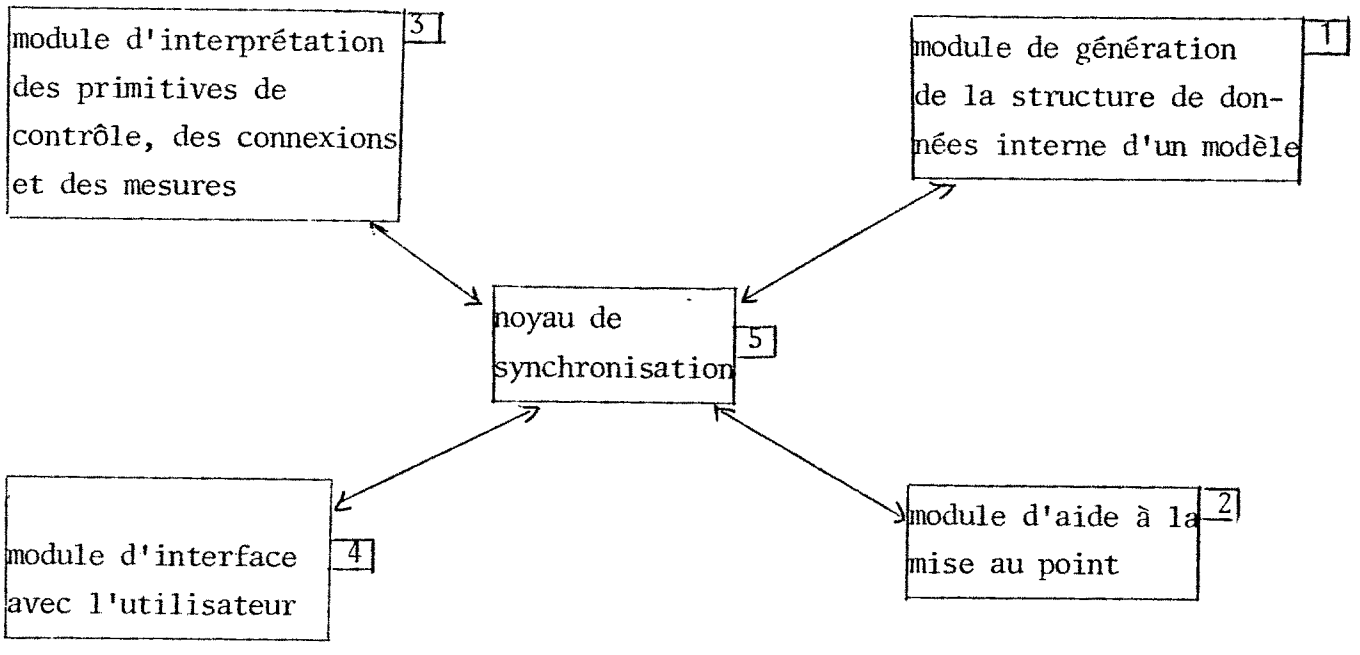
1) Un ensemble de procédures destinées à générer la structure de données interne d'un modèle simulable (ce point sera explicité au paragraphe 4 de ce chapitre),

2) un ensemble de procédures d'aide à la mise au point du simulateur; elles sont appelées en cours de simulation et permettent l'impression de tout ou partie de la structure interne de données simulée à des fins de contrôle,

3) une partie interprétation :
   a) des primitives de contrôle,
   b) des connexions (avec ou sans retard),
   c) des évènements de prises de mesures,

4) un noyau de synchronisation gérant l'échéancier local de chaque unité, l'échéancier global et les activations d'unités,

5) un superviseur de simulation assurant l'interface entre l'utilisateur et le modèle; il décode les ordres du langage de commande et lance les procédures de traitement correspondantes.



III-3 Langage de commande de simulation

A chaque instant observable de la simulation, c'est-à-dire après stabilisation du système simulé, l'utilisateur a la possibilité à l'aide d'un langage de commandes spécifique soit de visualiser n'importe quel type de variable du modèle, soit d'en modifier le contenu, demander la simulation du modèle jusqu'à une date exprimée par un nombre entier, demander l'arrêt de la simulation.
Le dialogue s'effectue à l'aide des commandes suivantes :

(le symbole { } signifie que le paramètre est optionnel)

RANGER <identificateur> = <valeur>
Initialise une variable scalaire ou tableau à une valeur compatible avec le type de la variable.

IMPRIM <identificateur>
Imprime la valeur d'une variable scalaire ou tableau de type quelconque.

DATE Imprime la date courante de simulation.

UNITE Imprime le nom de l'unité dans l'environnement de laquelle se trouve l'utilisateur, ci-dessous désignée par "unité courante".

ENVIRON <identificateur>
Permet à l'utilisateur de se placer dans le contexte de l'unité fournie en paramètre.

FINSIM Permet de sortir de l'environnement de simulation et rend le contrôle au système hôte.

SIMULE {<entier>}
Lance la simulation du modèle. Si le paramètre < entier > n'est pas précisé, le modèle est simulé jusqu'à stabilisation complète et le contrôle est rendu au superviseur qui se met en attente de commandes.
Si le paramètre <entier> est précisé, le modèle est simulé jusqu'à la date spécifiée. Si le modèle se stabilise avant cette date, la date courante est celle à laquelle le modèle s'est stabilisé.

SIGNAUX Imprime la valeur (précédée de son identificateur) de tous les signaux de l'unité courante.

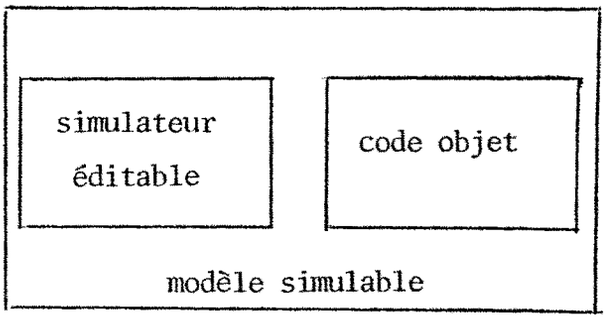
* Un astérisque en première position d'une ligne, indique que ce qui suit est un commentaire.

!SYTEST Permet de se placer dans un environnement de mise au point du simulateur. Cette commande n'est pas fournie à l'utilisateur.

L'ensemble des requêtes ennoncées forme un langage simple dans le prototype actuel. Ce langage peut être aisemment étendu à la demande des utilisateurs.

IV - LIAISON CODE OBJET GENERE - SIMULATEUR

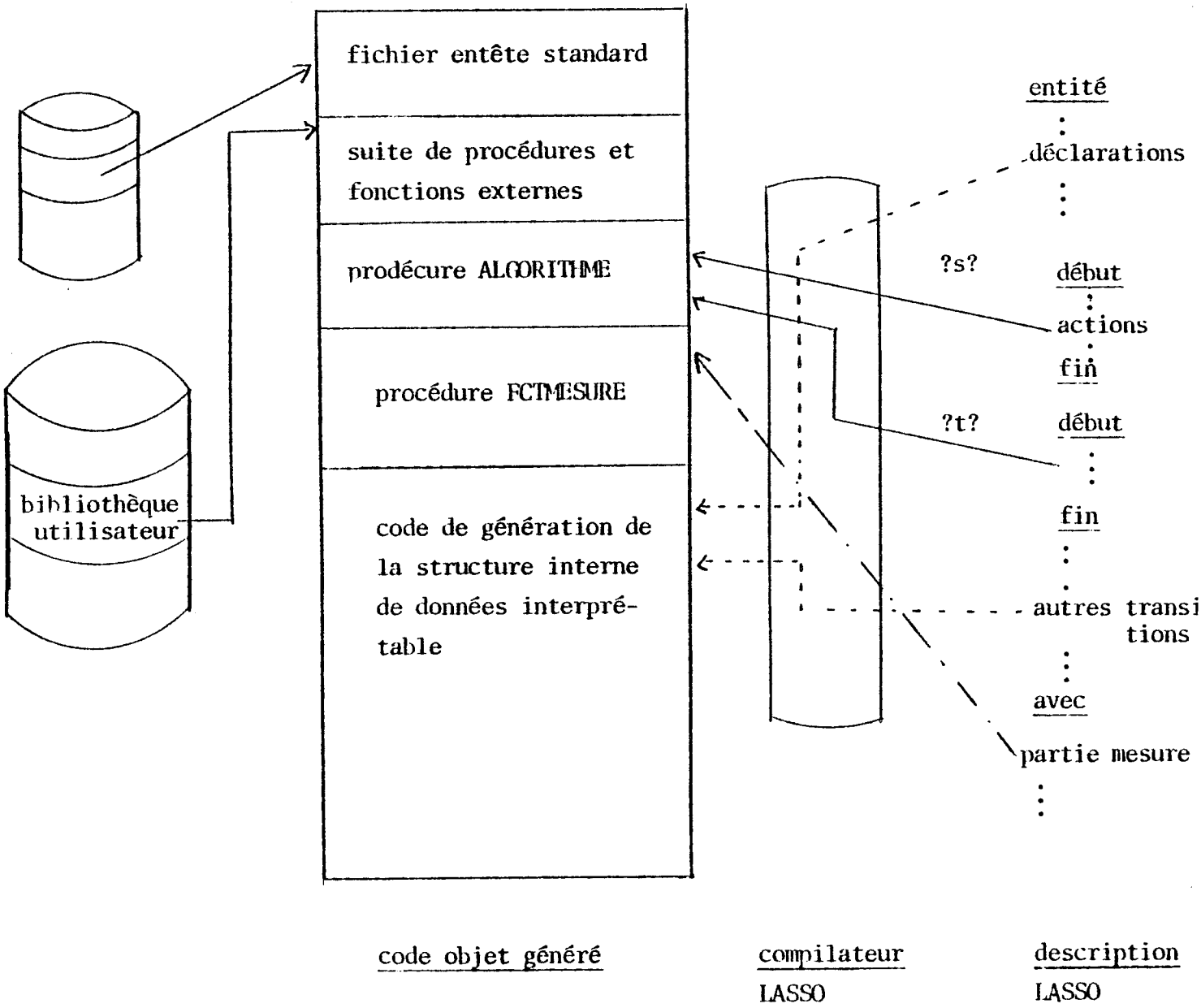
Le simulateur est un module objet éditable indépendant qui lié au code objet généré par le compilateur pour une description LASSO quelconque forme un modèle simulable.



Le modèle simulable est ensuite chargé en mémoire et le système d'exploitation hôte donne automatiquement le contrôle du simulateur qui dans un premier temps génère la structure de donnée interne du modèle puis se place dans l'état attente de commande utilisateur.

IV-1 Le code objet généré - structure et contenu

Tout code objet généré par le compilateur LASSO pour une description quelconque a la structure suivante :


fichier entête standard
suite de procédures et fonctions externes
prodécure ALGORITHME
procédure FCTMESURE
code de génération de la structure interne de données interprétable
bibliothèque utilisateur
entité
déclarations
?s?
début
actions
fin
?t?
début
fin
autres transitions
avec
partie mesure
code objet généré
compilateur
LASSO
description
LASSO

. L'entête standard : il est le même quelle que soit la description et contient la déclaration de toute la structure de données interprétable par le simulateur.

. Si l'utilisateur a déclaré dans un modèle des procédures ou fonctions externes celles-ci sont rangées à la suite de l'entête standard.

. Toutes les actions d'une description LASSO sont regroupées au sein d'une procédure unique appelée ALGORITHME.

. La partie des mesures non interprétée est regroupée dans la procédure FCTMESURE.

L'accès aux actions et aux mesures est fait en fournissant le couple (numéro d'unité, numéro d'action (ou numéro de l'action mesure)).

. La dernière partie est une suite d'appels à des procédures standards de construction de la structure de données interprétable du modèle.

IV-2 Liaisons code objet - simulateur

Les liaisons entre ces 2 modules sont établies par une suite de références externes faites dans les 2 modules.

IV-2-1 Liaisons nécessaires à la génération de la structure de données

- Le simulateur contient :
  - les déclarations de la structure de données nécessaires à la représentation d'une description LASSO
  - les définitions de toutes procédures standards de construction de la structure de données interprétable.

Les déclarations et les définitions sont décrites comme des objets accessibles de l'extérieur du simulateur.

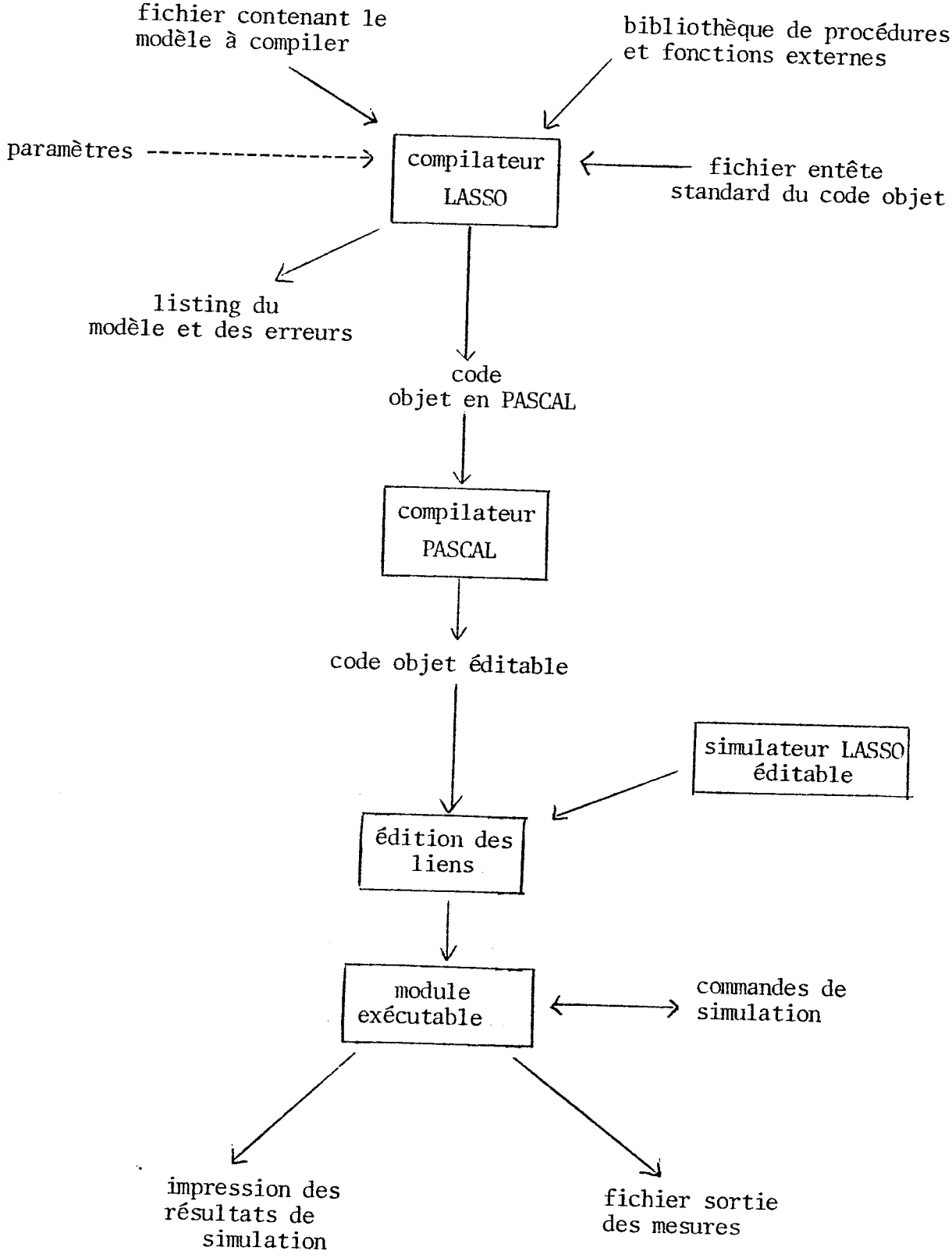
- Le code objet :
  - accède à la structure de données définie dans le simulateur
  - appelle les procédures standards définies dans le simulateur.

- La procédure de génération de la structure de données interne est prédéfinie au niveau du code objet et est accessible de l'extérieur du module. Lorsque le simulateur prend le contrôle sa première action est l'appel à cette procédure pour lancer la construction du modèle interne.

IV-2-2 Liaisons nécessaires à la simulation d'un modèle

Au niveau du code objet, les procédures ALGORITHME et FCTMESURE (qui peuvent être vides) sont déclarées comme des points d'entrée du module. Elles sont accédées au cours de la simulation chaque fois qu'une action ou un ordre de prise de mesure doit être exécuté pour le modèle simulé.

V - ORGANISATION GENERALE DU SYSTEME LASSO



V-1 L'étape de compilation

V-1-1 Fichiers en entrée

. Dans le système prototype toutes les entités d'une description doivent se trouver dans un seul fichier.

. Les procédures et fonctions référencées dans un modèle doivent être regroupées dans une bibliothèque unique.

. Le fichier contenant l'entête standard est permanent et est fourni avec le compilateur.

V-1-2 Fichiers en sortie

. La liste de la description suivie de la liste des erreurs éventuelles.

. Le code objet PASCAL généré si la description est sans erreur.

V-2 L'étape de simulation

Cette étape est divisée en 3 phases qui sont :

1) La compilation du code objet pour le compilateur PASCAL
2) L'édition des liens entre le simulateur et le code objet
3) La simulation proprement dite.

Au cours de la phase de simulation un fichier de sortie mesure peut être créé. L'enchaînement de ces 3 phases est lancé par une seule commande construite sous le système hôte.

Nous présentons en détail dans les chapîtres 3 et 4 le simulateur puis le compilateur. L'ordre de cette présentation est destiné à faciliter la compréhension du chapître de compilation en décrivant en premier la structure à laquelle on désire parvenir.

CHAPITRE TROIS

LE SIMULATEUR LASSO

I - INTRODUCTION

Le simulateur est un module précompilé indépendant de toute description LASSO. Ses fonctions sont :

- interpréter les structures de données représentant le modèle (cf paragraphe II)
- lancer l'exécution des actions et des ordres de prises de mesures et assurer leur synchronisation
- assurer la gestion des échanges de messages entre les unités
- assurer la communication entre le modèle en cours de fonctionnement et le concepteur.

Il contient entre autre les déclarations de données statiques nécessaires à la représentation interne d'un modèle LASSO.

II - LA STRUCTURE DE DONNEES INTERNE DU SIMULATEUR

La structure de données présentée ici est essentiellement un ensemble de tables qui seront interprétées par le simulateur. Ces tables sont initialisées en début de simulation par une partie du code objet, issu de la compilation d'une description LASSO, l'autre partie du code correspondant aux parties algorithmiques des actions et aux prises de mesures.
Différentes catégories d'objets sont distinguées en fonction de leur rôle à la simulation :

. les objets déclarés dans la description LASSO, et qui sont :

- les signaux (internes et d'interface)
- les variables (internes et d'interface)

. les objets interprétables de la description, qui sont les primitives de contrôles, les connexions et les prises de mesures.

. les objets système, transparents pour l'utilisateur, mais permettant, à partir du langage de commande du simulateur, d'accéder de l'extérieur à des objets du modèle.
Ce sont la table de correspondance, le graphe d'imbrication des unités, etc...

Toute la structure de données interne d'un modèle écrit en LASSO est statique. Cette caractéristique est commune à tous les langages de description de matériel et permet de ne conserver qu'un minimum d'informations pour la simulation du modèle. Nous présentons à la suite, la structure de données associée à une unité du modèle, puis la structure de données globale au modèle tout entier.

II-1 Structure de données locale à une unité

II-1-1 La table de correspondance

Cette table contient tous les éléments de mémorisation déclarés dans une unité. Ces éléments sont les suivants :

. les variables internes référencées dans le corps des primitives action, dans la primitive INDEX comme variable d'indexation entière, dans la primitive SELON comme valeur booléenne. Ces variables peuvent être de type entier, booléen, caractère ou nomsig : scalaires, tableaux ou structures.

. les signaux internes scalaires ou tableaux référencés en entrée-sortie des primitives de contrôle ou dans les mesures.

. les signaux d'interface (en entrée ou en sortie) scalaires ou tableaux référencés dans les connexions ou les primitives.

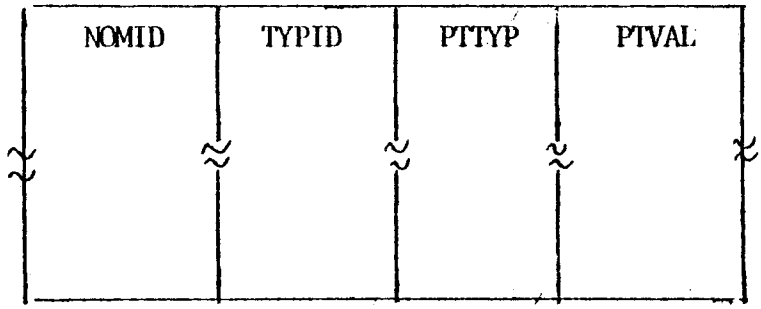
. les données d'interface associées aux signaux d'entrée-sortie et qui comprennent tous les types permis pour les variables internes.

Tous ces éléments doivent être observables et éventuellement modifiables par un utilisateur. Dans ce but, une table est créée, permettant d'associer à chaque objet déclaré, son type et sa valeur : cette table est appelée table de

correspondance. Elle a pour fonction de permettre l'accès interactivement à partir du langage de commande aux objets de l'unité par leur nom externe. Cet accès en lecture ou écriture nécessite la connaissance du type de la variable. En lecture, le type permet de réaliser les conversions nécessaires pour une impression de la valeur. En écriture, le type de la variable permet de vérifier les compatibilités (dimensionnelles et de type) avant de faire les modifications effectives.

A) Description de la table de correspondance

Chaque entrée est divisée en 4 champs :



NOMID — Ce champ contient l'identification externe de la variable déclarée dans la description. La longueur des identificateurs est paramétrable et a été fixée à 16 caractères.

TYPID — Ce champ indique si une variable est de type scalaire ou non.
Si la variable est scalaire, ce champ prend une des valeurs BOOL, ENTR, CARAC, SIGL, NSIG.
Si la variable n'est pas scalaire, le champ prend la valeur COMPLEXE.

PTTYP — est un champ contenant une valeur de pointeur sur un descripteur.
Si la variable est scalaire, ce champ est initialisé à la valeur NUL, sinon il contient le pointeur sur le descripteur de la variable. (TYPID contient alors la valeur COMPLEXE).

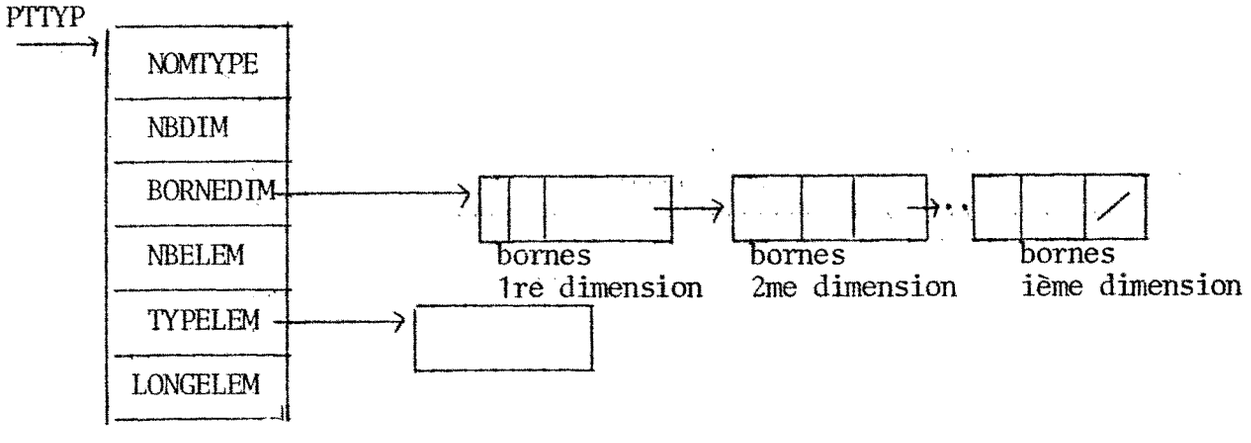
PTVAL — Ce champ contient le déplacement, par rapport au début de la zone des valeurs, de la zone valeur d'une variable scalaire ou du premier élément d'une variable COMPLEXE.

Pour un signal, ce champ contient la valeur de l'indice dans la table des signaux.
Pour une variable de type nomsig, une indirection est utilisée. Ce champ contient le déplacement, dans la zone des valeurs entières, d'une suite de une ou plusieurs valeurs d'indices sur la table de correspondance.
Les valeurs de ces indices permettent d'accéder aux noms des signaux contenus dans la variable nomsig. La valeur 0 d'un indice signifie que l'entrée correspondante de la variable nomsig n'est pas initialisée.

Lorsqu'une variable n'est pas scalaire, elle nécessite la présence d'un descripteur, afin que l'on puisse accéder partiellement ou totalement à ses zones valeurs. Dans le langage LASSO, nous pouvons rencontrer soit des tableaux, soit des structures ou une combinaison des deux.
Nous donnons pour chaque cas le descripteur associé.

B) Descripteur de tableau

Le descripteur d'une variable est repéré par le champ PTTYP de la table de correspondance.



NOMTYPE identifie un descripteur de tableau (valeur TABLEAU)

NBDIM spécifie le nombre de dimensions du tableau

BORNEDIM est un pointeur sur une liste dont chaque élément contient les valeurs des bornes inférieure et supérieure de chaque dimension de tableau.

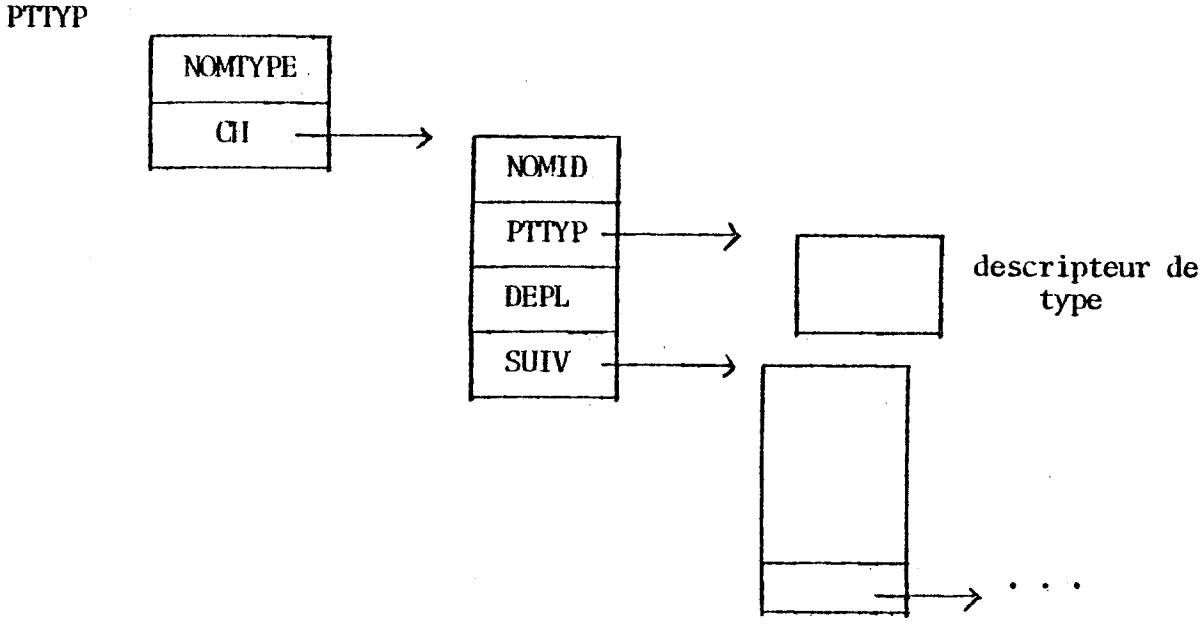
NBELEM exprime le nombre total d'éléments du tableau.

TYPELEM est un pointeur sur le type des éléments du tableau. Ceux-ci peuvent être des entiers scalaires, des booléens, etc... ou bien des structures. Dans cette dernière alternative, le champ TYPELEM pointe sur un descripteur de structure (décrit à la suite).

LONGELEM Contient la taille d'un élément de tableau. Cette information permet d'accéder efficacement à un élément d'un tableau de structure par exemple.

C) Descripteur de structure

L'entête d'un descripteur de structure est pointée par le champ PTTYP de la table de correspondance. Le descripteur est composé d'une entête pointant sur une liste de descripteurs de chaque champ de la structure.



. entête du descripteur

Elle se compose de deux champs qui sont :

NOMTYPE contient la valeur STRUCT

CH est le pointeur sur le descripteur du premier champ de la structure.

. descripteurs de champs

NOMID contient sous forme caractères, le nom du champ de la structure.

PTTYP repère le descripteur du type du champ. Le type peut être simple ou être lui-même un tableau ou une structure.

DEPL est le déplacement de la zone valeur du champ par rapport au début de la zone générale des valeurs (correspondant au type).

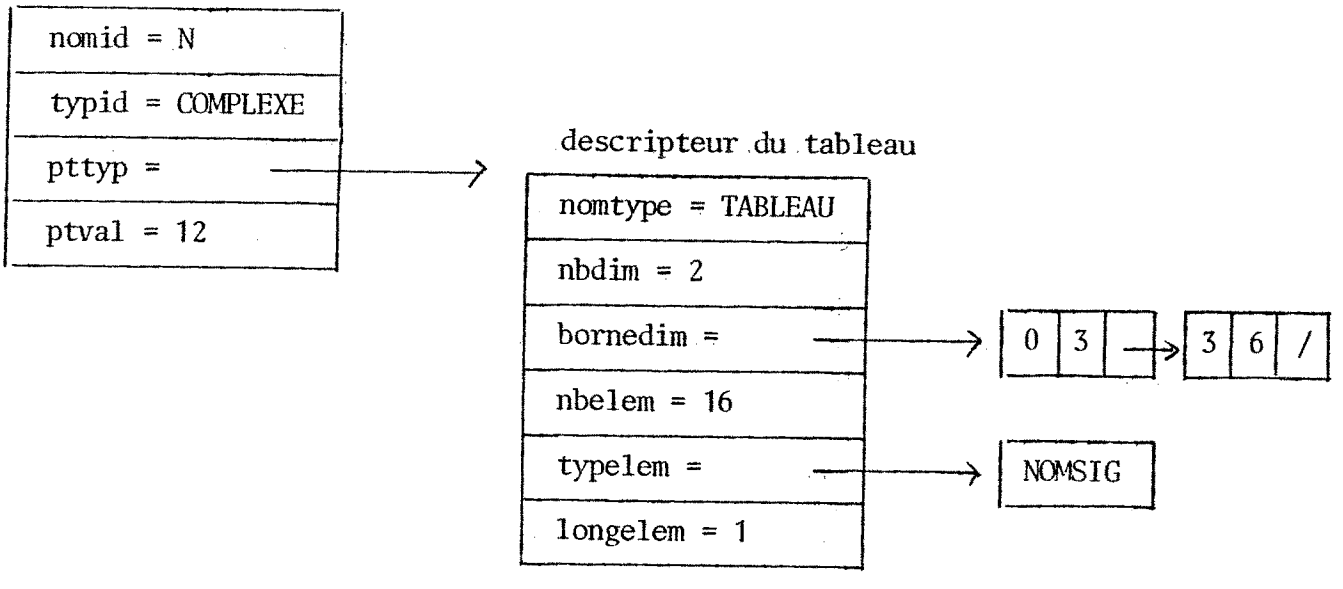
SUIV repère le descripteur du champ suivant de la structure. Si le descripteur est le dernier, ce champ vaut NUL.

D) Exemple 1 : structure de données générée par la déclaration d'un tableau

NOMSIG N[0:3,3:6]

Nous donnons la description d'une entrée de la table de correspondance, puis le descripteur lui-même.

1 entrée de la table de correspondance

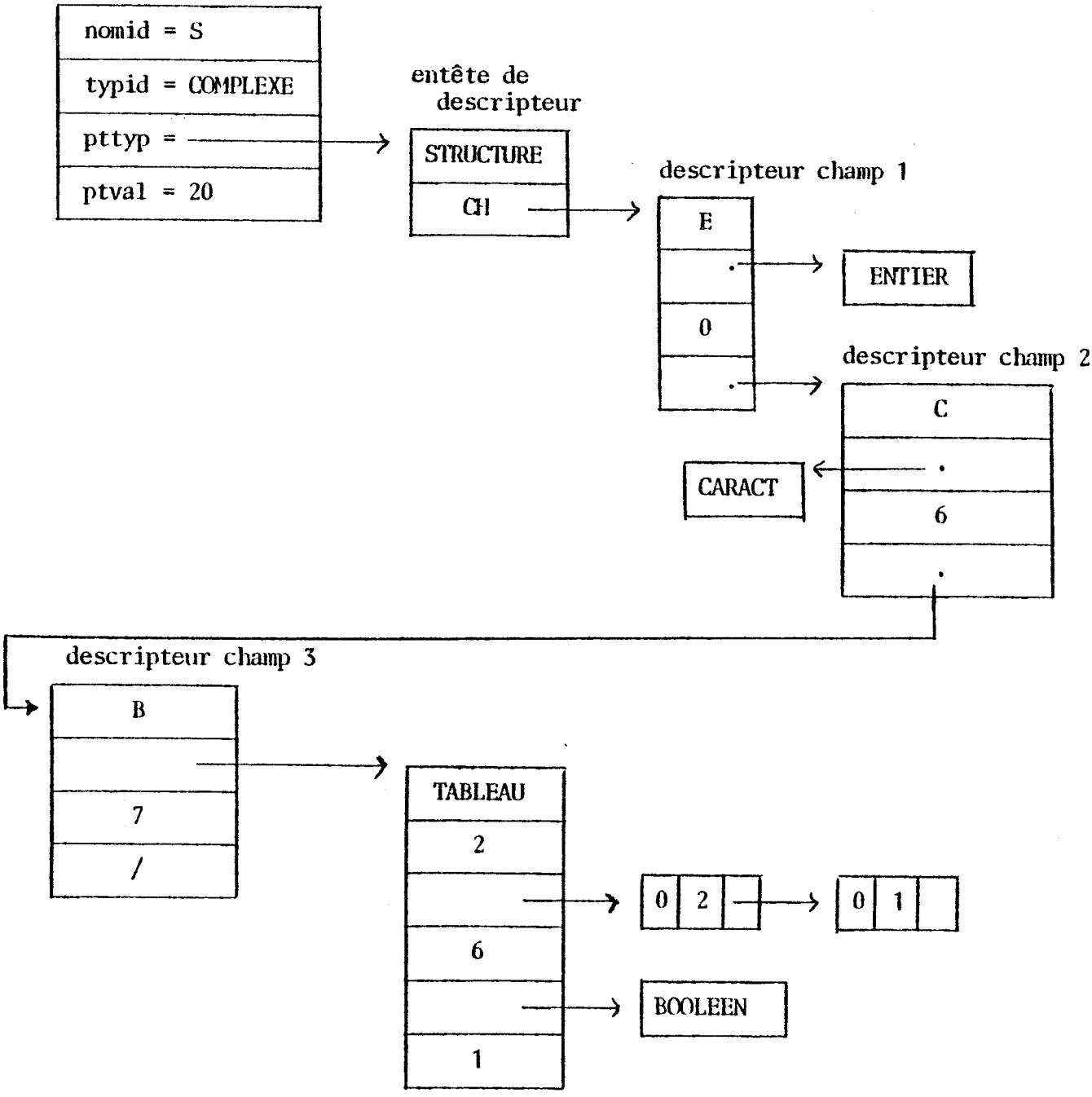


(longelem est exprimé en nombre de mots)

E) Exemple 2 : structure de données générée par la déclaration d'une structure

STRUCTURE (ENTIER E, CHAR C, BOOL B[0:2,0:1]) S ;

1 entrée de la table de correspondance

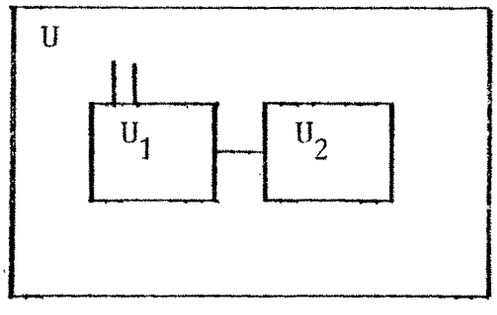


II-1-2 La table des signaux

L'ensemble des signaux déclarés interne, entrée ou sortie d'interface sont rangés dans cette table.

. Les signaux internes servent à contrôler le déclenchement des transitions et à en synchroniser le déroulement. Ces signaux doivent donc être accessibles par les procédures de simulation. A chaque signal interne est associée une liste des transitions ayant ce signal en entrée et une liste des transitions ayant ce signal en sortie (en paramètre de l'ordre VALIDER). A chaque changement de valeur d'un signal interne, il est possible de calculer le nouvel état de chaque transition dépendant de ce signal.

. Les signaux d'entrée ou sortie d'interface permettent d'établir les communications entre unités. Ces signaux sont accessibles dans l'unité où ils sont définis et dans l'unité immédiatement englobante. Soit l'exemple suivant :



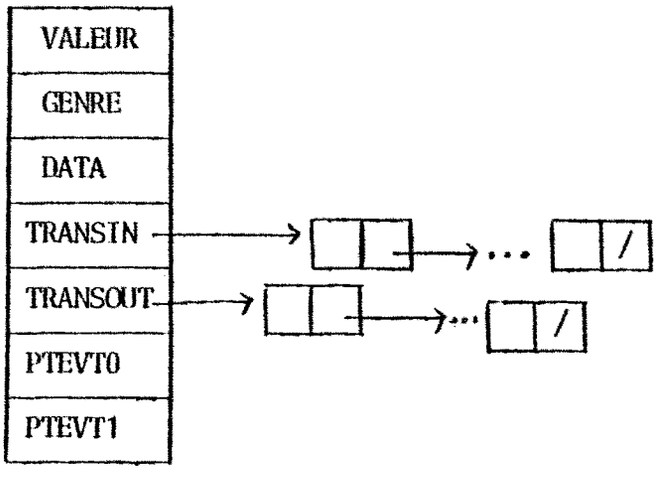
L'interface de l'unité U1 est partagé entre l'unité U et l'unité U1. De même l'interface de U2 est partagé entre l'unité U et U2.
Les communications entre l'unité U1 et U et les communications entre les unités U1 et U2 sont décrites par des instructions de connexions au niveau de U et sont simulées dans U.

. Les signaux d'entrée d'interface peuvent apparaître dans les conditions d'entrée des transitions de l'unité et dans les conditions de la partie mesure associée à l'unité. La réception d'un signal extérieur, avec ses données associées éventuelles, entraîne le positionnement d'un signal d'entrée d'interface, le calcul des nouveaux états des transitions qui dépendent de ce signal et le calcul des conditions de prises de mesures référençant ce signal.

. Les signaux de sortie d'interface peuvent apparaître en sortie des transitions ainsi que dans les mesures associées à l'unité. La validation d'un tel signal entraîne le calcul des conditions dans la partie mesure référençant ce signal. La transmission d'un signal vers l'extérieur entraîne l'activation de l'unité englobante dans laquelle est exprimée la connexion entre l'unité englobée et le monde extérieur. Si des données associées sont présentes, celles-ci doivent être transmises avec le signal. Si un retard de transmission est exprimé, les valeurs des données à transmettre sont celles qui ont été relevées à l'instant de validation du signal.

Structure de la table des signaux d'une unité

A chaque signal interne ou d'interface, est associée une entrée de la table des signaux. Chaque entrée a la structure suivante :



VALEUR est le champ valeur du signal (0 ou 1)

GENRE indique si le signal est interne, entrée d'interface, sortie d'interface.

DATA est le pointeur sur la tête de liste des données associées. Chaque élément de la liste contient l'adresse de la valeur de la donnée associée et son type. Si aucune donnée associée n'est présente, le champ est positionné à la valeur NIL.

TRANSIN est le pointeur sur la tête de liste des blocs évènements ayant ce signal en condition d'entrée de transition.

TRANSOUT est le pointeur sur la tête de liste des blocs évènements ayant ce signal en sortie de transition (en paramètre de l'ordre VALIDER).

PTEVT0 est l'indice sur la première occurrence dans la table des évènements mesures du passage à zéro du signal considéré.

PTEVT1 est l'indice sur la première occurrence dans la table des évènements mesures du passage à un du signal considéré.

Remarques :

1) Les listes pointées par TRANSIN et TRANSOUT contiennent les numéros de blocs évènements.

2) Pour un signal de type entrée d'interface, la liste pointée par TRANSOUT est vide.

   Pour un signal de type interne, les listes pointées par TRANSIN et TRANSOUT font référence strictement à des blocs évènements internes à l'unité.

   Pour un signal de type sortie d'interface, la liste pointée par TRANSIN fait référence à des numéros de blocs évènements de l'unité englobante.

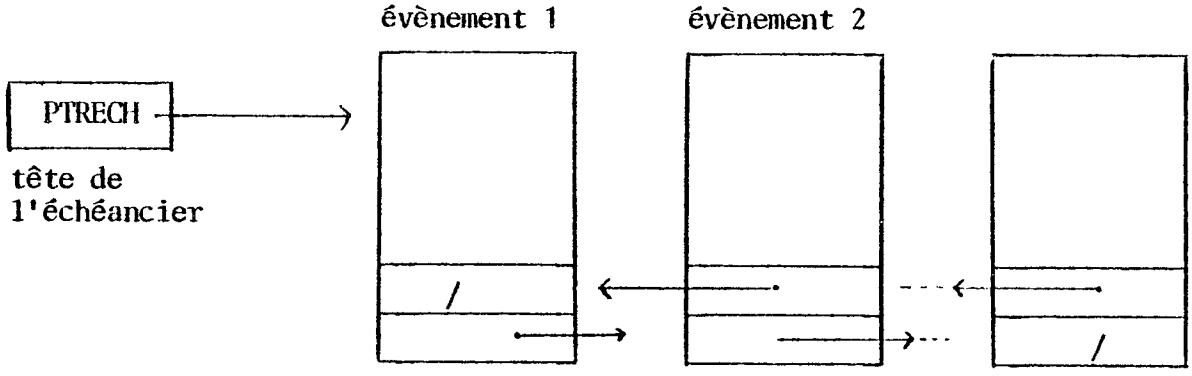
3) Pour un signal de type interne, le champ DATA peut pointer une liste de données associées dans le cas où le signal est directement connecté à un signal d'entrée ou sortie d'une unité englobée.

II-1-3 Les blocs évènement

Lors de la simulation d'un modèle LASSO, une partie du modèle est interprétée, une partie du modèle (les algorithmes des actions et les ordres de prises de mesures) est exécutée directement.
La partie interprétée du modèle comprend :

- Les transitions qui sont annoncées syntaxiquement par une suite de signaux entre '?' (? S1 et S2 ... ?) et terminées par un ordre VALIDER (Si...) optionnel. A chaque transition correspond un bloc évènement.
- Les connexions qui peuvent être simples (connexion signal par signal) ou connexions multiples qui seront alors décomposées à la compilation en connexions simples. A chaque connexion simple correspond un bloc évènement.
- Les mesures pour lesquelles une détermination de fenêtre d'observabilité et/ou une détermination d'instant d'observabilité doivent être calculées (cf. partie mesure).

La structure des blocs évènement sera donc fonction du type des transitions des connexions ou des mesures. L'ensemble de tous les blocs générés pour une entité sont numérotés et chacun d'eux possède une adresse dans la table des adresses des blocs évènements. En cours de simulation, les blocs évènements activables sont rangés par ordre chronologique de leur date d'activation dans l'échéancier local de l'unité. Cet échéancier est repéré par la variable globale PTRECH qui pointe sur le premier bloc activable.



Cet échéancier local est représenté sous forme d'une liste à double chaînage dans laquelle un traitement standard est fait pour tous les types de blocs évènements.

La présence ou non d'un bloc évènement dans l'échéancier local est déterminé par l'état du bloc évènement.

A un instant T de simulation, un bloc peut être dans l'un des 4 états suivants :

(nous indiquons alors s'il est ou non présent dans l'échéancier)

. le bloc est activable : toutes les conditions sont réunies pour qu'il soit interprété. Il est placé dans l'échéancier.

. le bloc est dans l'état bloqué : la condition sur les signaux en entrée est vérifiée pour qu'il puisse être interprété mais un ou plusieurs signaux en sortie sont positionnés à 1. Il est ôté de l'échéancier.

. le bloc est dans l'état différé si l'une des alternatives suivantes se présente :

- une primitive DELAI exprimée dans une transition, vient d'être exécutée,
- une primitive RETARD exprimée dans une connexion de signaux, vient d'être exécutée,
- une détermination d'instant d'observabilité vient d'être calculée dans les mesures associées à l'unité.

Dans tous les cas, le bloc différé est placé dans l'échéancier local dans l'ordre chronologique.

. Le bloc évènement est dans l'état inactif : ceci signifie que la condition d'entrée sur les signaux n'est pas vérifiée. Le bloc n'est pas présent dans l'échéancier.

Description des blocs évènements

La structure d'un bloc évènement comprend deux parties :

- une partie standard commune à tous les blocs évènements,
- une partie variable qui dépend du type de l'évènement à représenter.

A) Structure de la partie fixe

NUMBLOC est le numéro du bloc évènement attribué à la compilation et utilisé dans le traitement des erreurs de synchronisation.

TEMPSEXEC permet de simuler une durée d'exécution exprimée dans l'une des primitives DELAI (transition) ou RETARD (connexion).

ECHEANCE est la date à laquelle le bloc évènement doit être exécuté.

ECHAVANT chaînage avec le bloc précédent dans l'échéancier local (égal à NUL si premier bloc de l'échéancier)

ECHAPRES chaînage avec le bloc suivant dans l'échéancier local (égal à NUL si dernier bloc de l'échéancier)

BLOQUE si le champ contient la valeur VRAI, au moins une des sorties du bloc évènement est à la valeur 1 ; il n'est plus exécutable.

DATEBLOQUE indique la date à laquelle le bloc est passé dans l'état bloqué. Il est alors possible de vérifier en cours de simulation, qu'un bloc n'est pas resté dans cet état plus de n unités de temps.

ADFONCTION ce champ est utilisé pour lancer l'exécution des parties algorithmiques des actions, ou le code de prise effective des mesures. Il exprime soit le numéro de l'alternative 'CAS' dans la procédure ALGORITHME, ou le numéro de l'alternative 'CAS' dans la procédure FCTMESURE.

NBENTREE indique le nombre de signaux en entrée du bloc

NBSORT indique le nombre de signaux en sortie du bloc (en paramètre de la primitive VALIDER).

ENTREEA1 contient à tout instant le nombre de signaux en entrée du bloc, qui sont à 1.

SORTIEA0 contient à tout instant le nombre de signaux en sortie du bloc, qui sont à 0.

Les 4 derniers champs ont été introduits pour permettre une optimisation dans l'interprétation des blocs évènements.

VARIANT ce champ indique le type du bloc évènement à interpréter et permet :
1) de connaître la structure de la partie variable du bloc,
2) de choisir l'algorithme d'interprétation correspondant au type du bloc.

B) Structure de la partie variable

1) Des primitives de synchronisation

La transition ET

ENTRESIG est un pointeur sur la tête de liste des signaux en entrée de la transition.

SORTIESIG est un pointeur sur la tête de liste des signaux en sortie de transition. Cette liste peut être vide éventuellement.

La transition SELON

Pour cette transition, 3 listes sont générées à la compilation : la liste des signaux en entrée de transition, une liste de sortie de transition qui sera validée si l'expression booléenne est VRAI, une liste de sortie de transition qui sera validée si l'expression booléenne est FAUX.

SIGP pointe la tête de liste des signaux en entrée de transition.

SORTIEOUI pointe la tête de liste des signaux en sortie qui seront validés si l'expression est VRAI.

SORTIENON pointe la tête de liste des signaux en sortie qui seront validés si l'expression est FAUX.

La transition INDEX

Afin de faire un contrôle dynamique de la valeur de l'index avant la sélection du signal à valider, deux champs ont été définis, qui sont :

- la valeur initiale de l'index,
- le nombre d'éléments "validables" en sortie de transition.

SIGENT est la tête de liste des signaux en entrée de la transition.

BORNEINF est la valeur de la borne inférieure de l'index.

NBELEM est le nombre de signaux en sortie de transition.

VALINDEX est l'adresse de la zone valeur de l'index.

ADTABSIG est la tête de liste des signaux en sortie de transition. Un seul signal sera validé après interprétation de la transition.

transition ACTION

SIGENTREE repère la tête de liste des signaux en entrée de la transition.

SIGSORTIE repère la tête de la liste des signaux en sortie de la transition.

transition CHOIX ou CHOIXP

La même structure de données est utilisée pour représenter ces deux primitives. La distinction se fait au moment de l'interprétation en testant le champ FONCHOIX.

SIGPRI pointe la tête de la liste des signaux en entrée principale de la transition.

SIGENTR repère la liste des signaux en paramètre de la primitive CHOIX ou CHOIXP.

SIGSORT repère la liste des signaux en sortie de transition. A la fin de l'interprétation de CHOIX ou CHOIXP, un seul signal de la liste sera validé.

FONCHOIX permet de différencier la fonction CHOIX (valeur du champ = -1) de la fonction CHOIXP (valeur du champ = 1).

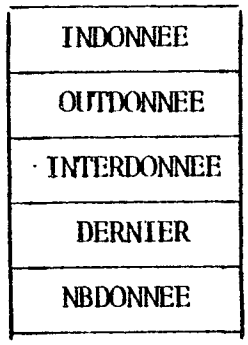
2) Des mesures

Dans les cas où les prises de mesures sont conditionnées par l'apparition d'un ou plusieurs évènements, des blocs évènements sont créés à la compilation. Ces blocs évènements ont une partie variable vide. Seuls sont dignificatifs les champs de la partie fixe, à l'exception des champs BLOQUE, DATEBLOQUE, NBENTRE, NBSORT, ENTREEA1, SORTIEA0.

A ces blocs évènements mesures sont appliquées toutes les procédures de gestion standard de l'échéancier local à chaque unité.

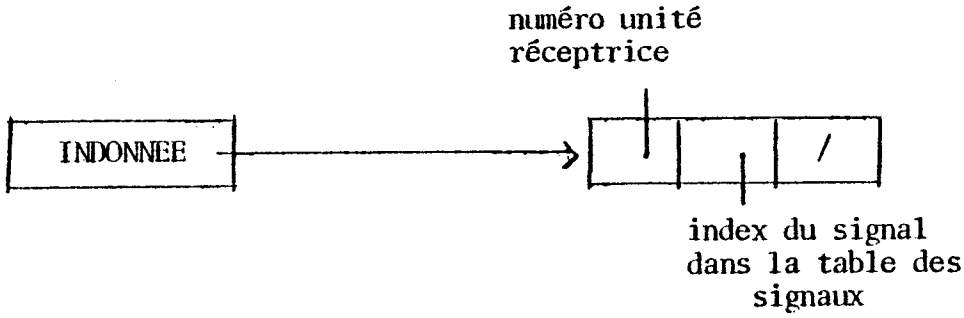
3) Des connexions

A chaque connexion simple de signal correspond un bloc évènement de type connexion. La structure de la partie variable est la suivante :

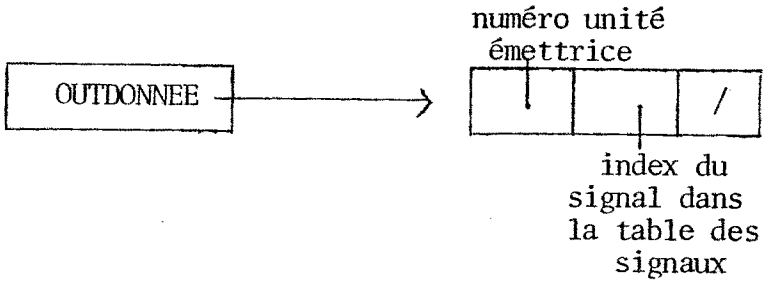


Deux types de connexions sont possibles : les connexions retardées et les connexions sans retard. Le retard est exprimé dans le champ TEMPSEXEC de la partie fixe. Pour une connexion non retardée, ce champ vaut 0.

Explicitons le rôle de chaque champ de la partie variable du bloc évènement :

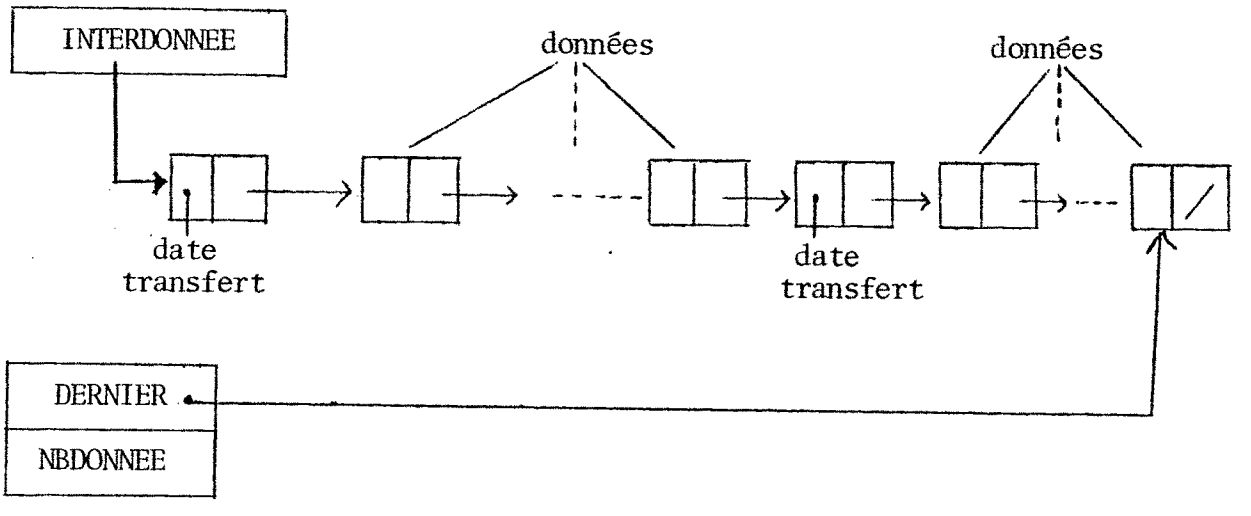


INDONNEE est un repère sur une liste dont les éléments ont 3 champs. Le premier champ contient le numéro d'une unité réceptrice, le second champ contient le déplacement dans la table des signaux du signal d'entrée à valider au moment de l'interprétation de la connexion.
Le 3ème champ pointe l'élément suivant de la liste.
La liste a plus d'un élément si un même signal est connecté à plusieurs signaux d'interface d'unités ("fanout").



OUTDONNEE repère une liste dont les éléments ont 3 champs.
Le 1er champ est le numéro de l'unité émettrice.
Le 2ème champ est le déplacement du signal de sortie de l'unité (repère par le champ 1), dans la table des signaux, à invalider à l'interprétation du bloc connexion.
Le 3ème champ est un chaînage vers l'élément suivant. La liste a plus d'un élément dans le cas de connections conditionnelles (non implémentées dans le prototype).

Les adresses des données associées à transférer en même temps que les signaux sont fournies pour chaque signal dans leur table de signaux respective.



Le champ INTERDONNEE est interprété différemment suivant que la connexion est retardée ou non.

Connexion non retardée

Ce champ contient le pointeur sur la liste des données associées au signal de sortie de l'unité émettrice.

Connexion retardée

Une unité peut envoyer des signaux et des données à une cadence supérieure à la valeur du retard. Les valeurs des données émises au moment de l'envoi du

signal, doivent donc être stockées temporairement dans des listes de données en transit. Ces listes sont chaînées les unes à la suite des autres. La première liste est repérée par le champ INTERDONNEE. Chaque liste est précédée d'un élément contenant la date effective à laquelle signal et données doivent être transmis. Cette date effective sert à mettre à jour le champ ECHEANCE du bloc évènement connexion retardé.
Dans le but d'accélérer l'interprétation des blocs connexion, 2 champs ont été introduits :

DERNIER est le pointeur sur le dernier élément de la dernière liste de données à transmettre. Il permet d'effectuer directement le chaînage aux listes précédentes lorsqu'un nouvel ensemble de données en transit arrive.

NBDONNEE indique le nombre d'éléments de données à transmettre pour une vague de données intermédiaires.

Ces deux champs sont sans signification lorsque la connexion est sans retard.

II-1-4 La table des évènements mesures

Les mesures présentent à la simulation des parties qui sont interprétées et des parties qui sont directement exécutées. Les ordres de prises de mesures et les conditions restrictives portant sur les variables LASSO sont transformées à la compilation en code PASCAL directement exécutable. La détermination des moments d'observation de modèle est interprétée : ceci concerne les blocs évènements, les évènements mesures et leur gestion.
Les évènements sont évalués en postfixé. Chaque entrée dans la table des évènements représente un opérande de l'expression postfixée, ou une sous-expression évaluée. Les feuilles de l'arbre postfixé sont les passages de 0 à 1 ou de 1 à 0 d'un signal.
Chaque entrée de la table des évènements est composée de 3 champs.

VALEVT	FCTION	PTEUR

VALEVT ce champ nous renseigne sur l'état de l'évènement (invalidation définitive, temporaire, partielle ou totale ; évènement prêt à être évalué) suivant la convention suivante :

VALEVT = 0 l'évènement à qui appartient cette entrée est en cours d'évaluation. Dès la validation de l'entrée, on positionne VALEVT à 1.

VALEVT = 1 deux cas se présentent. Soit l'évènement est temporairement invalidé, soit il est en cours d'évaluation et, dans ce cas, l'entrée considérée a déjà été validée.

Si l'entrée reçoit une nouvelle validation, cette arrivée sera ignorée. (La structure est "sans mémoire").

VALEVT = 3 l'évènement est dans ce cas définitivement invalidé ; nous avons intérêt à avoir une valeur différente de la précédente car, dans ce cas, nous pouvons retirer cet élément de la liste décrite plus haut. Ce déchaînage présente un intérêt : on optimise les parcours suivants de la liste en ôtant les éléments désormais inutiles.

En conclusion, lorsqu'un signal est validé ou invalidé, on parcourt la liste associée en faisant les actions suivantes :

- si VALEVT = 0 alors VALEVT := 1 ;
- si VALEVT = 1 alors on passe à l'élément suivant de la liste.
- si VALEVT = 3 alors on détruit le chaînage et on passe à l'élément suivant de la liste.

FCTION toutes les actions à effectuer par le simulateur, lorsqu'une entrée dans la table des évènements est validée, sont contenues dans une procédure (FCTMESURE) qui se réduit en fait à une instruction choix. (Cette procédure est générée à la compilation). Le champ FCTION contient un entier qui est un numéro de choix dans cette procédure.
Les actions présentes dans ce choix peuvent être :

- des appels à des procédures pré-définies dans le simulateur,
- des instructions écrites en PASCAL, correspondant aux conditions restrictives sur les variables et aux instructions de mesures, directement exécutables par le simulateur.

PTEUR pointeur vers un élément de la table des évènements permettant de parcourir l'arbre post-fixé.

Exemple

Soit l'expression suivante :

*1A ETDEPLUS (((*1A PUIS *1B) COMMESI *1C ETDEPLUS *1B) PUIS *1D)

La structure de données interne générée à la compilation, est la suivante :

Table des signaux

Nom	.....	PTEVT0	PTEVT1
:			
A		nil	1
:		nil	
B		nil	3
C		nil	5
D			9

Table des évènements

	mesures VALEVT	FCTION	PTEUR	
1	/	9	2	→A
2	/	1	nil	→A
3	/	2	7	→B
4	/	3	nil	→A puis I
5	/	4	nil	→C
6	/	5	nil	→(4)commesi
7	/	6	nil	B
8	/	7	nil	(6)etdeplu
9	/	8	nil	D
10	/	10	nil	(8) puis D
11	/	11	nil	A etdeplus

Procédure FCTMESURE

```
cas   sur n° de fonction
      début
        1 : ;
        2 : puis (2,3,4) ;
        3 : commesi (4,6,5,5) ;
        4 : commesi (5,6,2,4) ;
        5 : etdeplus (6,7,8) ;
        6 : etdeplus (7,6,8) ;
        7 : ;
        8 : puis (8,9,10) ;
        9 : etdeplus (1,10,11) ;
       11 : code-mesure ;
      fin
```

Cette procédure générée à la compilation fait partie du fichier code objet.

II-2 Structure de données globale à une unité

Un modèle LASSO est composé d'une ou plusieurs unités interconnectées. Chaque unité peut contenir une ou plusieurs unités internes. L'ensemble forme une hiérarchie d'unités imbriquées d'une profondeur quelconque. En cours de simulation, l'utilisateur a la possibilité d'accéder à toutes les variables d'une quelconque des unités du modèle. Il faut donc lui offrir, via le langage de commandes, le moyen de se positionner dans le contexte d'une unité en la nommant par son nom externe. De plus, lors de l'interprétation de certains blocs évènements (les connexions par exemple), il est nécessaire de se positionner dans le contexte d'une autre unité que celle qui est active.

II-2-1 La table des unités

Cette table traduit le graphe d'inclusion des unités du modèle. Elle est créée à la compilation lorsque tout le modèle a été analysé. Il y a une entrée par unité déclarée. Chaque entrée est structurée de la façon suivante :

NOMUNIT contient le nom externe déclaré de l'unité. Il permet à l'utilisateur de se positionner dans le contexte de cette unité en la nommant. La longueur (paramétrable) de l'identificateur, a été fixée à 16 caractères.

RCONTXT est l'adresse de la table de contexte associée à l'unité. La table de contexte est une suite d'adresses de tables propres à chaque unité.

RPROG est un entier correspondant au numéro de l'alternative CAS dans la procédure ALGORITHME associée à l'unité. Ce champ est utilisé lors de l'appel de la procédure ALGORITHME comme 1er paramètre.

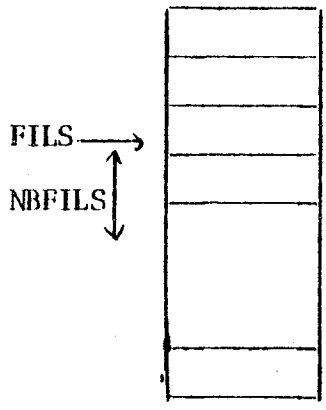
PERE est le numéro de l'unité père. C'est aussi l'index sur la table des unités. Pour l'unité la plus englobante, ce numéro vaut 0.

NBFILS est le nombre de fils directs de l'unité.

FILS est l'index dans la table des fils du premier fils direct.

{ DATES, SUIV, PRECEDENT } Ces trois champs sont utilisés pour la gestion de l'échéancier global au niveau du modèle. Ils sont explicités plus loin.

II-2-2 La table des fils



Chaque entrée de la table contient un entier qui est un index sur la table des unités. La liste des fils directs d'une unité est repérée par un index de début (champ FILS) et un nombre de fils (champ NBFILS) de la table des unités .

II-2-3 L'échéancier global

L'activité dans une unité consiste en l'exécution :
1) de transitions (en parallèle ou séquentiellement),
2) d'interprétations de blocs connexions (retardés ou non),
3) d'interprétations d'actions de prises de mesures.

Dans un but d'efficacité de l'algorithme de simulation, les changements de contexte d'unités doivent être minimisés. Ceci signifie que le simulateur doit exécuter à l'instant T tout ce qui est possible d'exécuter dans une unité ; pour cela, il dispose d'un échéancier local pour chaque unité.
Au niveau du modèle, le simulateur doit connaître la prochaîne unité à activer sans avoir à consulter tous les échéanciers locaux de toutes les unités : c'est le but de l'échéancier global du modèle. Cet échéancier est réalisé dans la table des unités par les 3 champs DATES, SUIV et PRECEDENT.

DATES est la date de la prochaine activation de l'unité. Cette date correspond à la date d'activation de l'évènement en tête de l'échéancier local de l'unité et est mise à jour après que tout ait été exécuté dans l'unité courante et avant de donner le contrôle à l'unité chronologiquement suivante.

SUIV indice dans la table des unités, l'unité qui sera activée après l'unité courante.

PRECEDENT indice l'unité qui est rangée chronologiquement avant.

Remarque En début de simulation, l'échéancier global est vide (SUIV et PRECEDENT sont initialisés à -1) et le champ DATES est positionné à 0.

II-2-4 Tables de contexte d'une unité

Chaque activation d'unité ou chaque demande utilisateur de positionnement dans une unité, entraîne le chargement de son context dans les variables de travail du simulateur. Ce context est constitué d'une table d'adresses et d'une table d'indicateurs.

A) La table des adresses

Elle est adressée par le champ RCONTXT de la table des unités. Cette table contient une suite d'adresses et le pointeur sur la tête de l'échéancier local de l'unité.

RID adresse de la table de correspondance

RENT adresse de la zone des valeurs entiers

RBOOL adresse de la zone des valeurs booléennes

RCHAR adresse de la zone des valeurs caractères

RSIG adresse de la table des signaux

RUNIT adresse de la table des unités

REVENT adresse de la table des adresses des blocs évènements

RFILS adresse des fils directs

RINDIC adresse de la table des indicateurs

DEBECH pointeur sur tête de l'échéancier local.

B) La table des indicateurs

Elle est adressée par le champ RINDIC de la table des adresses. Elle contient une série d'indicateurs permettant d'accélérer le traitement de la partie conversationnelle du simulateur.

TEMPS indique la date du prochain évènement qui se produira dans l'unité.

MOI est le numéro de l'unité courante. Ce numéro est attribué séquentiellement à la compilation au-fur-et-à-mesure des déclarations des unités.

NBSIG longueur de la table des signaux.

NBUNIT longueur de la table des unités.

NBIDC longueur de la table de correspondance.

II-3 Organisation générale de la structure interne de données d'un modèle LASSO

Le schéma présente une vue globale de la structure interne de données. Il précise les liens entre les objets locaux à une unité et les objets globaux à un modèle.

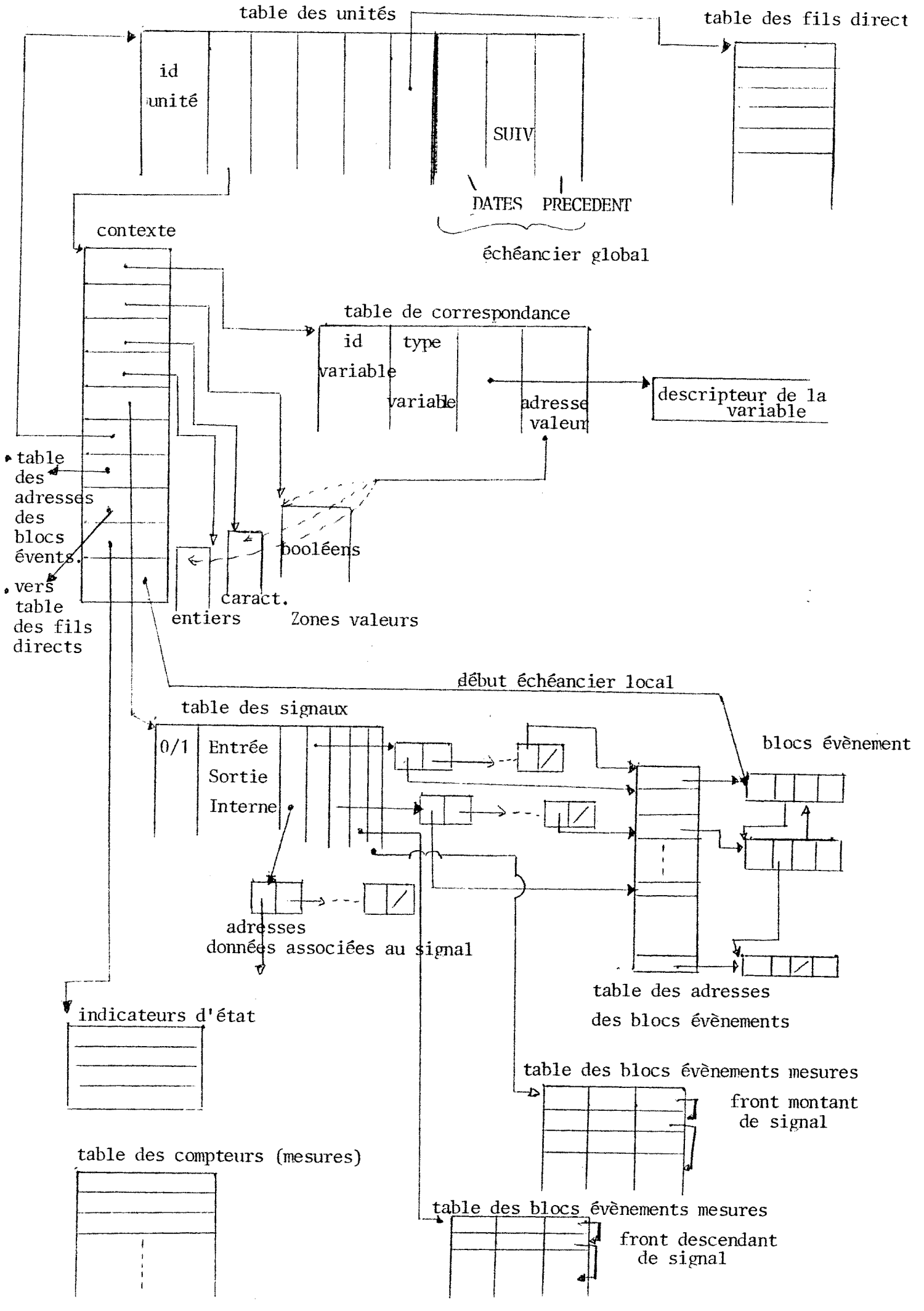


table des unités
table des fils direct
id unité
SUIV
DATES
PRECEDENT
échéancier global
contexte
table de correspondance
id variable
type variable
adresse valeur
descripteur de la variable
table des adresses des blocs évents.
vers table des fils directs
booléens
caract.
entiers
Zones valeurs
début échéancier local
table des signaux
0/1
Entrée
Sortie
Interne
blocs évènement
adresses données associées au signal
table des adresses des blocs évènements
indicateurs d'état
table des blocs évènements mesures
front montant de signal
table des compteurs (mesures)
table des blocs évènements mesures
front descendant de signal

III - L'INTERPRETEUR DE SIMULATION

Il rassemble les fonctions d'interprétation

1) des primitives du graphe de contrôle,
2) des connexions (avec et sans retard),
3) des mesures.

A chaque type de bloc évènement est associée une procédure d'interprétation particulière.

III-1 Les procédures d'interprétation des primitives du graphe de contrôle

Les procédures VALIDSELON, VALINDEX, CHOIXP, ET, OPER et CHOIX interprètent les 6 primitives de synchronisation du langage LASSO. Lorsqu'une procédure est appelée pour interpréter un bloc évènement, celui-ci répond à toutes les conditions pour être traité, c'est-à-dire :

a) tous les signaux de sortie du bloc sont à 0
b) en fonction du type de bloc, la configuration des signaux d'entrée permet de lancer l'interprétation.

Dans une première étape, chaque procédure lance la validation des signaux de sortie du bloc éventuellement en fonction de la configuration des signaux en entrée de ce bloc évènement. Dans une deuxième étape, tous les signaux en entrée du bloc sont positionnés à 0. La validation d'un signal de sortie de bloc comme l'invalidation d'un signal d'entrée de bloc, entraîne le calcul des nouveaux états des blocs dépendant de ce ou ces signaux. Des blocs évènements peuvent :

a) devenir activables car ils répondent à toutes les conditions pour être interprétés et sont alors rangés dans l'échéancier local.
b) devenir bloqués ; ils étaient activables mais l'interprétation du bloc courant a conduit à une validation d'un signal de sortie qui les bloque. Ils sont alors ôtés de l'échéancier local.
c) rester dans le même état.
d) rester dans le même état, mais une erreur a été détectée, résultant d'une mauvaise synchronisation du graphe de contrôle. (Un bloc bloqué ou activable voit un de ses signaux d'entrée passer à 0).

III-2 Les procédures d'interprétation des connexions

A chaque connexion de signal, correspond un seul bloc évènement connexion.
Son interprétation peut entraîner plusieurs changements de contexte d'unités suivant qu'il s'agit de connexions entre unités englobées dans une autre ou de connexions entre une unité et son unité englobante. Après validation d'un signal de sortie connecté à une unité (voir procédure VALIDESORTIE), le bloc évènement associé est rangé chronologiquement dans l'échéancier local de l'unité où est déclarée la connexion. Le signal de sortie peut être transmis instantanément, ou bien être retardé (présence de la primitive RETARD).
L'interprétation d'un bloc évènement connexion entraîne la mise à jour du contexte de l'unité réceptrice du signal, la validation d'un signal d'entrée pouvant rendre actives une ou plusieurs transitions de cette unité.
Deux procédures assurent l'interprétation de ces blocs évènements :

. La procédure CONNEX traite les blocs connexions non retardés
. La procédure CONNEXRETARD traite les blocs connexions retardés.

III-2-1 La procédure CONNEX

Elle est paramétrée par l'adresse du bloc évènement connexion sans retard à interpréter. Après chargement du contexte de l'unité réceptrice (si cette unité n'est pas l'unité active), on vérifie que le signal récepteur est à 0, c'est-à-dire que le message précédent a été pris en compte. Si non, un message d'erreur de synchronisation est envoyé à l'utilisateur, lui précisant les noms des unités mises en cause. Si le signal récepteur est à 0, il est mis à 1 ;
les données associées sont transférées vers les données associées réceptrices, le signal émetteur est invalidé, le contexte de l'unité réceptrice est mis à jour,en particulier le pointeur sur le début de l'échéancier local de l'unité. Cette dernière mise à jour, peut à son tour entraîner la mise à jour de l'échéancier global du modèle (l'unité réceptrice ne pouvant plus être activée, dû à un blocage, ou au contraire un évènement immédiat étant déclenché).

III-2-2 La procédure CONNEXRETARD

(paramètre : adresse du bloc évènement à interpréter).
De même que dans la procédure CONNEX, un chargement du contexte de l'unité réceptrice est fait (si l'unité réceptrice n'est pas l'unité active). On vé-

rifie que le signal récepteur est à 0 ; si non, un message d'erreur de synchronisation est envoyé à l'utilisateur, précisant les noms des unités concernées. Si oui, le signal récepteur de l'unité est validé. Du fait de la présence d'un retard dans la connexion, les données associées, si elles existent, transitent par une liste intermédiaire, précédées de la date de leur transfert (voir structure des blocs connexions décrite précédemment). Après transfert des données intermédiaires vers l'unité réceptrice, l'espace mémoire acquis dynamiquement pour créer cette liste, est libéré. Si un prochain transfert est prévu (le dernier élément de la liste libérée pointe sur la vague suivante des données à transmettre), l'échéance du bloc connexion est initialisée avec la date de transfert effective de la vague suivante de données intermédiaires, et le bloc est rangé chronologiquement dans l'échéancier local ; sinon, le bloc connexion est ôté de l'échéancier local.
Compte-tenu du fait qu'un signal d'entrée de l'unité réceptrice a été validé, des transitions peuvent devenir activables ou devenir bloquées dans cette unité. Un calcul du nouvel état de l'unité réceptrice est fait qui impose la mise à jour éventuelle du pointeur sur l'échéancier local. Cette mise à jour entraîne, comme dans la procédure CONNEX, la modification éventuelle de l'échéancier global du modèle.

III-3 Les procédures d'interprétation des mesures

III-3-1 Principe

Dès qu'une date de prise de mesure a été déterminée, un bloc évènement mesure est placé dans l'échéancier local.
L'activation d'un bloc évènement mesure conduit à l'exécution de deux types d'actions :

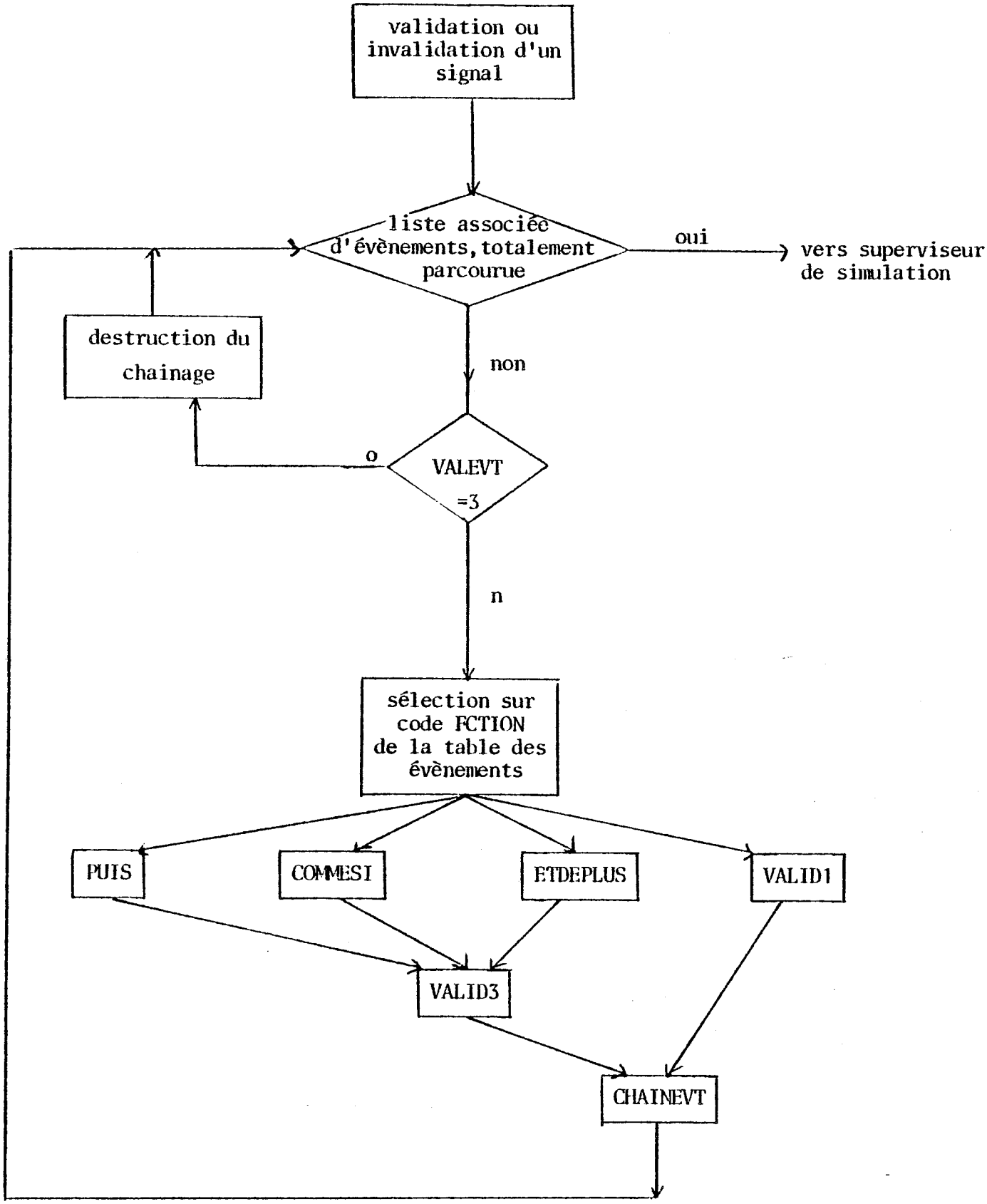
1) Les prises de mesures effectives (avec l'évaluation de leurs conditions restrictives, un SI par exemple),
2) l'évaluation de l'instant d'observabilité suivant, qui conduit à replacer le bloc évènement dans l'échéancier local ou bien l'ôter de l'échéancier.

Ces actions sont lancées par appel à la procédure FCTMESURE, qui a la structure d'un choix. Le numéro de l'alternative du choix est fourni par le champ ADFONCTION du bloc évènement mesure actif ou le champ FCTION d'une entrée de la table des évènements mesures. Trois procédures d'interprétation des blocs mesures sont présentes dans le simulateur; ce sont les procédures PUIS, COMMESI,

ETDEPLUS correspondant aux 3 primitives du langage de mesure de même nom. Elles font appel à des procédures d'évaluation d'évènement de mesure (VALID1, VALID3, CHAINEVT, VALID2, FORCER) détaillées dans [29 (TOME 2)].

III-3-2 Schéma général d'évaluation d'un évènement mesure

Après la validation ou l'invalidation d'un signal par le simulateur, les actions suivantes sont réalisées :


validation ou invalidation d'un signal
liste associée d'évènements, totalement parcourue
oui
vers superviseur de simulation
non
destruction du chainage
o
VALEVT =3
n
sélection sur code FCTION de la table des évènements
PUIS
COMMESI
ETDEPLUS
VALID1
VALID3
CHAINEVT

III-4 La validation des signaux

A chaque signal sont associées 4 listes, qui sont :

. la liste des numéros de blocs évènements ayant le signal en entrée.
. la liste des numéros de blocs évènements ayant le signal en sortie.
. la liste des actions mesures à exécuter si le signal transite de 1 à 0.
. la liste des actions mesures à exécuter si le signal transite de 0 à 1.

Une ou plusieurs des listes peuvent être vides.

III-4-1 Validation d'un signal de sortie d'interface

L'opération de validation d'un signal de sortie est réalisée par la procédure VALIDESORTIE (S), où S est l'indice du signal, dans la table des signaux, de l'unité en cours de traitement.
Si le signal S n'est ni connecté ni en entrée d'une transition dans l'unité père, cette procédure est sans action. Sinon, la procédure range dans l'échéancier de l'unité père un ou plusieurs blocs évènements ayant comme signal d'entrée S.

Si S est connecté, et si des données associées existent, celles-ci sont transférées dans la liste des données intermédiaires, repérée par le champ INTERDONNEE du bloc connexion au niveau du père.
Dans le cas où la connexion de signal est non retardée, on vérifie que le bloc connexion correspondant n'est pas déjà dans l'échéancier ; s'il y est, une erreur de synchronisation est détectée entre les unités, et un message est envoyé à l'utilisateur, précisant les unités concernées. Dans le cas contraire, le bloc connexion est rangé dans l'échéancier local de l'unité père avec, comme échéance d'exécution, la date courante T.
Dans le cas où la connexion de signal est retardée, plusieurs trains de données en transit peuvent être présents, dont le premier est repéré par le champ INTERDONNEE du bloc connexion concerné (voir paragraphe II.1.3.B).
L'arrivée d'une nouvelle vague de données à transmettre entraîne la création dynamique d'une nouvelle liste intermédiaire, qui sera chaînée à la suite de celles déjà présentes.
L'échéancier local de l'unité père est mis à jour, ce qui peut entraîner une mise à jour de l'échéancier global.
Puis la procédure restaure le contexte de l'unité en cours de traitement.

III-4-2 Validation et invalidation d'un signal interne

Deux procédures réalisent les fonctions de validation et d'invalidation de signaux :

. La procédure MAJO1 est appelée par les procédures d'interprétation des blocs évènement correspondant aux transitions et aux connexions, pour chacun de leurs signaux d'entrée et de sortie à modifier.
Un message d'erreur est envoyé à la simulation si on essaie de valider un signal d'entrée d'interface ou d'invalider un signal de sortie d'interface (test rendu nécessaire par l'existence des "nomsig").

. La procédure REPERCUTE est appelée par la procédure MAJO1.

Dans un premier temps, on vérifie que le signal à valider ou invalider n'est pas déjà respectivement à 1 ou 0 ; si oui, un message d'erreur est envoyé à l'utilisateur.
Puis cette procédure examine tour à tour les listes pointées par les champs TRANSIN et TRANSOUT de l'entrée I de la table des signaux.

En parcourant la liste pointée par TRANSIN, elle examine tous les blocs évènements ayant ce signal en entrée ; elle détermine pour chacun en fonction de son type (bloc ET, SELON ...), son nouvel état (activable, bloqué, sans changement) ou bien détecte une erreur (invalidation d'un signal d'entrée d'un bloc précédemment activable ou bloqué).
Si un bloc devient activable, il est mis dans l'échéancier local par appel à la procédure AJOUTE ; s'il devient bloqué, il est ôté de l'échéancier par appel à la procédure ENLEVE. En parcourant la liste pointée par TRANSOUT, elle détermine les états des blocs évènements ayant ce signal en sortie. Le travail est similaire à celui explicité pour TRANSIN.

III-4-3 Calcul des états des blocs évènements

Le calcul d'un état d'un bloc évènement est assuré par la procédure ECHEANCIER, qui a en paramètre, l'adresse du bloc à traiter. Cette procédure est appelée par REPERCUTE à chaque validation ou invalidation de signal. En fonction du type de bloc, certains compteurs sont mis à jour et des vérifications de cohérence sont faites. Si aucune erreur n'a été détectée, le nouvel état du bloc est déterminé ; en cas de changement d'état, la procédure AJOUTE ou ENLEVE est appelée.

IV - LE NOYAU DE SYNCHRONISATION : GESTION DES ECHEANCIERS

Le noyau de synchronisation assure la gestion :
- de l'échéancier global au modèle,
- des échéanciers locaux aux unités.

L'échéancier global est pointé par la variable COURANT. C'est une liste à double chainage des unités du modèle dont chaque élément contient l'index de l'unité considérée dans la table des unités, ce qui permet un accès direct aux éléments du contexte de l'unité que l'on active.
L'échéancier local de l'unité en cours de simulation est pointé par la variable globale PTRECH. C'est une liste à double chainage des blocs évènements activables et prévus. Si plusieurs évènements doivent se réaliser à la même date t, leur position dans l'échéancier correspond à leur ordre d'arrivée (fonctionnement FIFO pour une date donnée). Ceci assure un parcours du graphe de contrôle en largeur plutôt qu'en profondeur. Les gestions des 2 types d'échéanciers sont très dépendantes les unes des autres. L'échéancier global est mis à jour (si nécessaire) après chaque fin d'activation d'une unité ou après validation d'un signal de sortie d'interface qui entraîne une modification de l'échéancier local. L'échéancier global est géré par les 2 procédures ENLEGLOBAL et AJOUTEGLOBAL.
L'échéancier local est mis à jour par les 2 procédures ENLEVE et AJOUTE [30 (tome 2)].

V - LE SUPERVISEUR DE SIMULATION

La fonction du superviseur est double :

1) il assure l'intialisation et l'exécution de l'algorithme de simulation des unités du modèle.
2) il assure la gestion de l'interface entre le modèle simulé et l'utilisateur, via le langage de commandes.

Schéma général du dialogue

→ en début de simulation

. La variable globale T représentant le temps absolu de simulation, est initialisée à 0.

. Le contexte courant est celui de l'unité la plus englobante.

. L'échéancier global est vide.

. Le simulateur est en attente de commande.

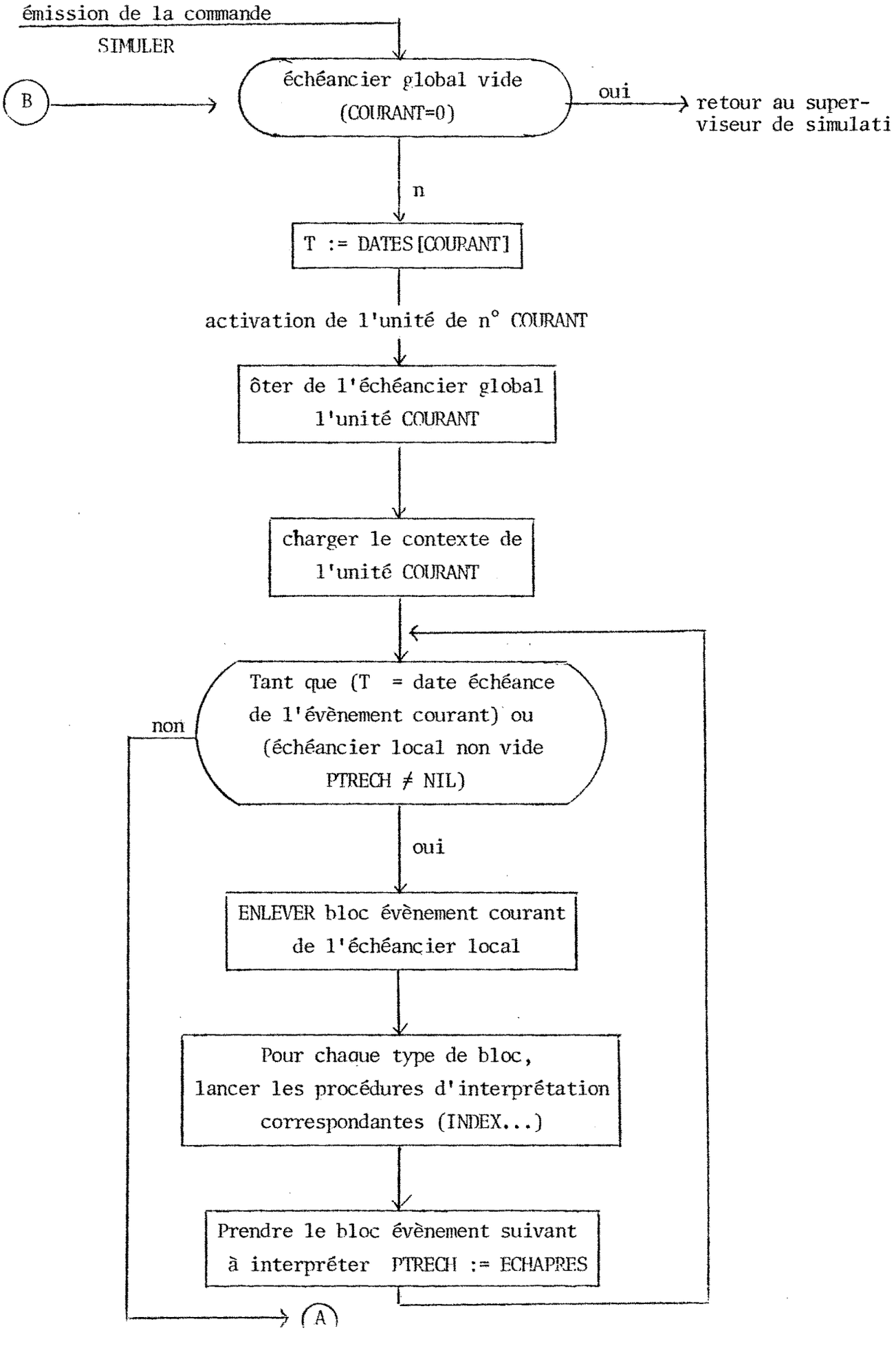
- émission d'une commande

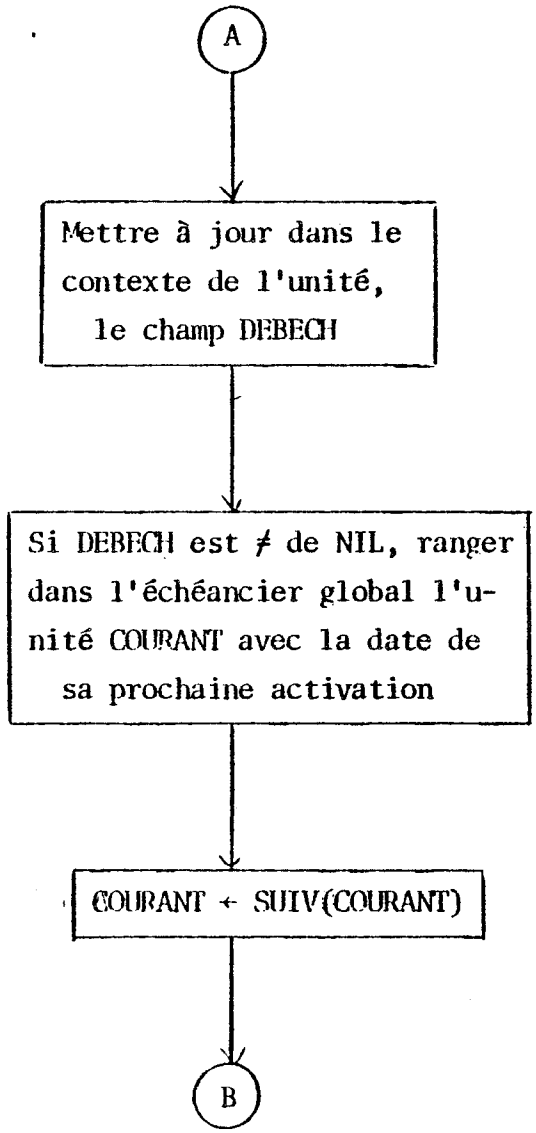
Elle est décodée, vérifiée syntaxiquement et ses paramètres éventuels traités. Une fois la commande exécutée, le simulateur se remet dans l'état attente de commandes.

- fin de simulation

Elle est atteinte par émission de la commande FINSIM. Un message de fin de simulation est imprimé, ainsi que la valeur du temps courant.

L'algorithme global de simulation




A
Mettre à jour dans le contexte de l'unité, le champ DEBECH
Si DEBECH est ≠ de NIL, ranger dans l'échéancier global l'unité COURANT avec la date de sa prochaine activation
COURANT ← SUIV(COURANT)
B

Cet algorithme est exécuté lorsque l'utilisateur émet à partir du langage de commande, l'ordre de simuler le modèle pendant un certain nombre d'unités de temps, lorsque le simulateur doit décider de la prochaine échéance à exécuter, ou bien lorsque l'utilisateur modifie ou initialise manuellement des variables de type signal. L'unité qui sera activée est toujours l'unité qui se trouve en tête de l'échéancier global ; l'élément de tête contient le numéro de l'unité et sa date d'activation. Une fois activée, l'unité est enlevée de l'échéancier global.
La procédure GLOBAL assure le lancement de l'exécution des unités. Le modèle sera activé tant que l'échéancier global n'est pas vide (le modèle s'est alors stabilité) ou bien tant qu'une date limite de simulation (fournie par l'utilisateur) n'a pas été atteinte. La procédure ACTIVERUNITE, est paramétrée par le numéro de l'unité à activer. Elle assure le chargement du contexte de l'unité à partir de la table des unités et active successivement tous les blocs évènements présents dans l'échéancier local, en appelant les procédures d'interprétation correspondantes. L'interprétation d'un bloc transition terminée, le calcul de son nouvel état est fait en appelant la procédure ECHEANCIER. Ceci n'est pas fait pour les blocs connexion (avec ou sans retard), ni pour les blocs mesures. Une fois tous les blocs évènements exécutés pour une échéance T, le pointeur sur la tête de l'échéancier local est mis à jour dans le contexte de l'unité active (champ DEBECH de la table de contexte) et l'unité est replacée (éventuellement) dans l'échéancier global (appel de la procédure AJOUTEGLOBAL).

VI - OUTIL D'AIDE A LA MISE AU POINT DU SIMULATEUR

Dans le but d'offrir une aide à la mise au point du simulateur, un environnement particulier a été défini. Il permet à chaque instant observable de la simulation, de connaître l'état de la structure interne de données, représentant le modèle ainsi que l'état des variables système. L'émission de la commande !SYSTEST permet de passer dans cet environnement.
Le contrôle est donné alors à la procédure principale PSYSTEST, qui se met en attente de commandes (voir plus loin la liste). Après exécution d'une commande, le contrôle lui est toujours rendu (sauf pour la commande FIN).
Il est possible alors d'imprimer en clair tout ou partie du modèle interne en émettant l'une des commandes suivantes :
(chaque commande sera suivie entre parenthèses par le nom de la procédure principale qui la réalise).

EVS : imprime l'état des variables globales système (appel ECRVS).

ETI : imprime le contenu de la table des indicateurs (ECRTI).

ECT : impression du contexte de l'unité courante (appel ECRTXT).

EUN : impression du contenu de la table des unités (appel ECRUNIT).

ETC : impression de la table de correspondance de l'unité courante (appel ECRTC).

ETS : impression de la table des signaux de l'unité courante (appel ECRTS).

EBL <i> : impression du bloc évènement de numéro i de l'unité courante.

ONS : positionne la variable globale SYSTEST à vrai pour signaler le mode test.

OFS : positionne la variable globale SYSTEST à faux pour signaler la fin du mode test.

FIN : permet de rendre le contrôle au superviseur de simulation.

Remarque

Le positionnement de la variable SYSTEST permet de tracer l'exécution du simulateur, en imprimant la valeur de certaines variables et/ou les identificateurs des procédures exécutées.
Les procédures dont les noms ont été cités dans les diverses commandes, ne sont que des procédures d'impression de tableau et de listes, et sont décrites dans le dossier de programmation.

VII - ANNEXES DU CHAPITRE 3

Nous donnons ici un exemple d'utilisation de la commande !SYSTEST pour l'aide à la mise au point du simulateur.
L'exemple d'une description LASSO avec 3 unités, est listé. Il est suivi, en annexes 2 et 3 de deux simulations :

- une permettant d'imprimer la structure de donnée interne d'une unité,
- une permettant de tracer l'exécution du simulateur.

Conventions pour la présentation des simulations

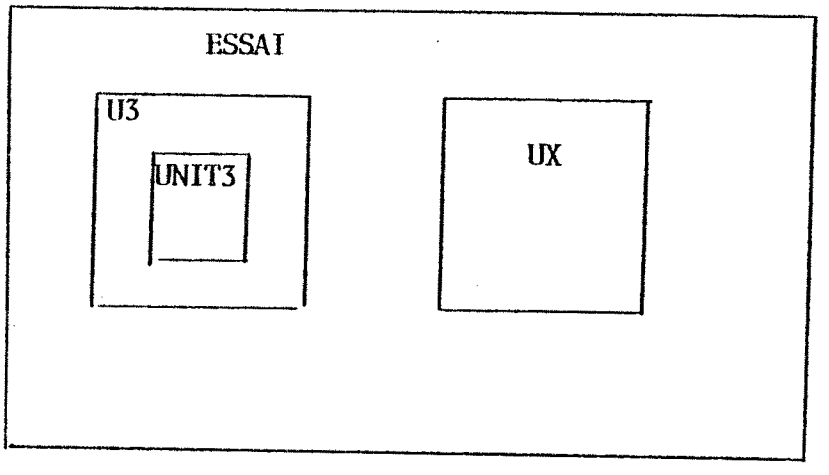
. Le ? indique que le simulateur est en attente de commande.
. Les champs des blocs évènements sont listés dans l'ordre de leur présentation au paragraphe II-1-3.
. Dans la présentation de la table de correspondance, l'entier entre parenthèses (1) fait référence au descripteur de la variable portant la même valeur (1).
. Dans la présentation de la table des signaux, l'entier entre parenthèses (2) fait référence aux 3 listes pointées par les champs DATA, TRANSIN, TRANSOUT, et dont le contenu est listé à raison d'un entier par élément de liste.

ANNEXE 1

EXEMPLE DE DESCRIPTION LASSO

La liste qui suit décrit 3 entités ENT3, ENT4 et ESSAI.

L'entité ESSAI contient 2 unités fils directs U3 et UX.
L'unité U3 contient 1 unité fils direct UNIT3.
Le schéma d'imbrication des unités est le suivant :



Les 2 unités UNIT3 et UX sont 2 exemplaires de l'entité ENT3. (cf lignes 75 et 122 de la liste).
L'unité U3 est l'exemplaire unique de l'entité ENT4. (cf ligne 121).

Dans un but de simplicité de la description aucune connexion n'est établie entre les unités du modèle.

LISTE DE LA DESCRIPTION

```
version compilateur lasso nov 1979..
debut compilation
          1---    entite ent3; <----------------------------
          2---  interfac
          3---  entier ad,data;  %donnees d'interface%
          4---  entree read(ad),write(ad,data),ready; %signaux d'entree%
          5---  sortie oklec(data),okecr;  %signaux de sortie%
          6---  corps
          7---
          8---     %variables internes%
          9---     %..................%
         10---
         11---  entier table,c,d,k[1:5];  %variable locale%
         12---  entier ss;
         13---  nomsig seul[1:2];
         14---  signal pret[1:3],lec,ecr;
         15---  signal finread,finwrite;
         16---
         17---      %procedures externes%
         18---      %..................%
         19---
         20---  fonction externe f(entier x) resultat : entier;
         21---  procedur externe p(entier p1,p2 );
         22---
         23---     %transitions%
         24---     %...........%
         25---
         26---  ?ready? choix(read,write) valider(lec,ecr);
         27---  ?lec? debut
         28---  c := 0;
         29---  d:= 1;
         30---  k[1]:= c;
         31---  d:=k[d];
         32---  k[2]:=d;
         33---  k[5] := c;
         34---  k[4]:=22;
         35---  fin delai(1) %temps de lecture%
         36---  valider(finread);
         37---  ?ecr? debut
         38---  c:=2;
         39---  k[c]:=5;
         40---  c:=c+1;
         41---  d:=12;
         42---  d:=d*c;
         43---  c:=c*d/2;
```

```
44---  d := c * (d /2);
45---  p(c,d);
46---  c:=f(c);
47---  p(12,1);
48---  fin delai(2)
49---         valider(finwrite);
50---  ?finread? debut
51---  c:=2;
52---  si (c > 1) alors d:=0  sinon c:=-2 finsi;
53---  si (d <= 12) alors d :=33; c:=30; finsi;
54---  tantque (d < c) faire c:=c*41;
55---  repeter debut d:=d*c; c:=c+12 fin;
56---  jusqua (d<247); fin valider(oklec);
57---  ?finwrite? valider(okecr);
58---
59---
60---    fin;
61---
62---    entite ent4; <----------------------------------------
63---  interfac
64---  entier a1,b,c1;
65---  entier  d ;
66---  caract f[0:5,0:6];
67---  entree e1(a1),e2(b,c1);
68---  sortie s1(f),s2;
69---  corps
70---
71---     %variables internes%
72---     %.................%
73---
74---  entite externe ent3;
75---  unite unit3:ent3;<-------------------déclaration unité fils
76---  entier j,k init 1;
77---  entier l[3:5] init (12,9,3),
78---         m[0:4,0:7];
79---  signal s3,s4,s5,s6,s7,s8,s9,s10;
80---  nomsig n[0:2] init (s3,s4,s5);
81---  caract o,p[0:3] init (k,b,c,d);
82---  bool q init (1),r;
83---
84---     %transitions%
85---     %..........%
86---
87---  procedur externe proc1(caract p5);
88---  procedur externe proc2(entier p6;bool p7);
89---  fonction externe fonc1(entier p8) resultat entier :res1;
90---  fonction externe fonc2(bool p9) resultat bool:res2;
91---  ?s3? debut fin valider(s4);
92---  ?s4? debut fin delai(4) valider(s5);
93---  ?s5 et s6? debut fin valider(s3);
94---  ?s7 et s8 et s9? debut fin valider (s3,s4);
```

```
 95---  fin;
 96---
 97---
 98---       %les 3 entites ne sont pas connectees
 99---           entre elles%
100---
101---
102---
103---  entite essai; <----------------------------------
104---  interfac
105---
106---    %variables d'interface%
107---
108---
109---  entier a1,b,c1;
110---  entier  d ;
111---  entier c[1:3], e;
112---  caract f[0:5,0:6];
113---  bool g[0:1,0:2,0:3] ,h,i;
114---
115---      %signaux d'interface%
116---
117---  entree e1(a1),e2(b,c1);
118---  sortie s1(f,i),s2;
119---  corps
120---  entite externe ent3,ent4;
121---  unite u3:ent4;<------------------------ déclarations 2 unités
122---  unite ux:ent3;<------------------------ fils directs
123---
124---     %variables locales%
125---
126---  entier j,k init 1;
127---  entier l[3:5] init (12,9,3),
128---         m[0:4,0:7];
129---
130---      %signaux locaux%
131---
132---  signal s3,s4,s5,s6,s7,s8,s9,s10;
133---  nomsig n[0:2] init (s3,s4,s5);
134---  caract o,p[0:3] init (k,b,c,d);
135---  bool q init (1),r;
136---
137---      %procedures externes%
138---
139---  procedur externe proc1(caract p5);
140---  procedur externe proc2(entier p6;bool p7);
141---  fonction externe fonc1(entier p8) resultat entier :res1;
142---  fonction externe fonc2(bool p9) resultat bool:res2;
143---
144---    %.......................%
145---    %   transitions         %
146---    %.......................%
147---
148---  ?s3? debut fin valider(s4);
149---  ?s4? debut fin delai(4) valider(s5);
150---  ?s5 et s6? debut fin valider(s3);
151---  ?s7 et s8 et s9? debut fin valider (s3,s4);
152---  fin;
153---
description sans erreur: fin compilation
```

ANNEXE 2

SIMULATION AVEC IMPRESSION DE LA STRUCTURE INTERNE DU MODÈLE

```
 PASCEXEC
CORE?7#
FICHE?MES66
BINAIRE?SIM6
- SFER/PASCAL-SYSTEM,VERS. 1/11/76
?
?*
?*-----FACILITES OFFERTRES A L'ECRIVAIN DU
?*       SYSYTEME
?*
?UNITE
NOM UNITE COURANTE: ESSAI
?*
?*
?*---APRES PASSAGE EN MODE 'MISE AU POINT'
?---NOUS IMPRIMERONS LA STRUCTURE INTERNE
COMMANDE INCONNUE
?*---SIMULABLE DE L'UNITE COURANTE
?*
?ISYSTEST
SUPERVISEUR EN MODE TEST...
COMMANDES DISPONIBLES: EVS,ETC,ETS,ETI,ECT,EUN,EBL,FIN,ONS,OFS
?EVS
VARIABLES SYSTEME..
RANG=          # SEUL=          # REPONSE=       F T=          # CODERREUR=          #
SYSTEST=       F ECHEANCIER=NIL-ECH  COURANT=         -1 DATELUE=          #
?
 ETC
TABLE DE CORRESPONDANCE
NOM IDF,TYPE,PTEUR VERS DESCR,INDEX DS TABLE DES VALEURS
A1                ENTIER   NIL-TYP            1 (          1 )
B                 ENTIER   NIL-TYP            2 (          2 )
C1                ENTIER   NIL-TYP            3 (          3 )
D                 ENTIER   NIL-TYP            4 (          4 )
C                 COMPLEXE PTDESCR            5 (          5 )
E                 ENTIER   NIL-TYP            8 (          6 )
F                 COMPLEXE PTDESCR            1 (          7 )
G                 COMPLEXE PTDESCR            1 (          8 )
H                 BOOLEEN  NIL-TYP           25 (          9 )
I                 BOOLEEN  NIL-TYP           26 (         1# )
E1                SIGNAL   NIL-TYP            1 (         11 )
E2                SIGNAL   NIL-TYP            2 (         12 )
S1                SIGNAL   NIL-TYP            3 (         13 )
S2                SIGNAL   NIL-TYP            4 (         14 )
J                 ENTIER   NIL-TYP            9 (         15 )
K                 ENTIER   NIL-TYP           1# (         16 )
```

Commande lancement du simulateur

→ (*) cf page suivante

→ pointeur vers UN descripteur de Tableau

→ pas de descripteur pour un scalaire

```
L                COMPLEXE PTDESCR          11 (         17 )
N                COMPLEXE PTDESCR          14 (         18 )
S3               SIGNAL   NIL-TYP           5 (         19 )
S4               SIGNAL   NIL-TYP           6 (         20 )
S5               SIGNAL   NIL-TYP           7 (         21 )
S6               SIGNAL   NIL-TYP           8 (         22 )
S7               SIGNAL   NIL-TYP           9 (         23 )
S8               SIGNAL   NIL-TYP          10 (         24 )
S9               SIGNAL   NIL-TYP          11 (         25 )
S10              SIGNAL   NIL-TYP          12 (         26 )
N                COMPLEXE PTDESCR          54 (         27 )
O                CARACTER NIL-TYP          43 (         28 )
P                COMPLEXE PTDESCR          44 (         29 )
Q                BOOLEEN  NIL-TYP          27 (         30 )
R                BOOLEEN  NIL-TYP          28 (         31 )
PAS DE DESCRIPTEUR.......(          1 )
PAS DE DESCRIPTEUR.......(          2 )
PAS DE DESCRIPTEUR.......(          3 )
PAS DE DESCRIPTEUR.......(          4 )
D E S C R I P T E U R  (          5 )
DEBUT DU DESCRIPTEUR.....
DEBUT DESCRIPTEUR DE TABLEAU..
NBDIM          1 NBELEM          3 LOGELEM          1
DEBUT BORNES DU TABLEAU..
          1..          3
FIN BORNES DU TABLEAU..
D E S C R I P T E U R  (          5 )
DEBUT DU DESCRIPTEUR.....
 ..ENTIER..
FIN DU DESCRIPTEUR.....
FIN DESCRIPTEUR DE TABLEAU..
FIN DU DESCRIPTEUR.....
PAS DE DESCRIPTEUR.......(          6 )
D E S C R I P T E U R  (          7 )
DEBUT DU DESCRIPTEUR.....
DEBUT DESCRIPTEUR DE TABLEAU..
NBDIM          2 NBELEM         42 LOGELEM          1
DEBUT BORNES DU TABLEAU..
          0..          5
          0..          6
FIN BORNES DU TABLEAU..
D E S C R I P T E U R  (          7 )
DEBUT DU DESCRIPTEUR.....
 ..CARACTERE..
FIN DU DESCRIPTEUR.....
FIN DESCRIPTEUR DE TABLEAU..
FIN DU DESCRIPTEUR.....
D E S C R I P T E U R  (          8 )
DEBUT DU DESCRIPTEUR.....
DEBUT DESCRIPTEUR DE TABLEAU..
NBDIM          3 NBELEM         24 LOGELEM          1
```

(*) variable A1 de l'entité ESSAI

descripteur du tableau F de ESSAI

```
DEBUT BORNES DU TABLEAU..
          0..          1
          0..          2
          0..          3
FIN BORNES DU TABLEAU..
D E S C R I P T E U R (          8 )
DEBUT DU DESCRIPTEUR.....
 ..BOOLEEN..
FIN DU DESCRIPTEUR.....
FIN DESCRIPTEUR DE TABLEAU..
FIN DU DESCRIPTEUR.....
PAS DE DESCRIPTEUR.....(          9 )
PAS DE DESCRIPTEUR.....(         10 )
PAS DE DESCRIPTEUR.....(         11 )
PAS DE DESCRIPTEUR.....(         12 )
PAS DE DESCRIPTEUR.....(         13 )
PAS DE DESCRIPTEUR.....(         14 )
PAS DE DESCRIPTEUR.....(         15 )
PAS DE DESCRIPTEUR.....(         16 )
D E S C R I P T E U R (         17 )
DEBUT DU DESCRIPTEUR.....
DEBUT DESCRIPTEUR DE TABLEAU..
NBDIM          1 NBELEM          3 LOGELEN          1
DEBUT BORNES DU TABLEAU..
          3..          5
FIN BORNES DU TABLEAU..
D E S C R I P T E U R (         17 )
DEBUT DU DESCRIPTEUR.....
 ..ENTIER..
FIN DU DESCRIPTEUR.....
FIN DESCRIPTEUR DE TABLEAU..
FIN DU DESCRIPTEUR.....
D E S C R I P T E U R (         18 )
DEBUT DU DESCRIPTEUR.....
DEBUT DESCRIPTEUR DE TABLEAU..
NBDIM          2 NBELEM         40 LOGELEN          1
DEBUT BORNES DU TABLEAU..
          0..          4
          0..          7
FIN BORNES DU TABLEAU..
D E S C R I P T E U R (         18 )
DEBUT DU DESCRIPTEUR.....
 ..ENTIER..
FIN DU DESCRIPTEUR.....
FIN DESCRIPTEUR DE TABLEAU..
FIN DU DESCRIPTEUR.....
PAS DE DESCRIPTEUR.....(         19 )
PAS DE DESCRIPTEUR.....(         20 )
PAS DE DESCRIPTEUR.....(         21 )
PAS DE DESCRIPTEUR.....(         22 )
PAS DE DESCRIPTEUR.....(         23 )
```

```
 PAS DE DESCRIPTEUR.......(         24 )
 PAS DE DESCRIPTEUR.......(         25 )
 PAS DE DESCRIPTEUR.......(         26 )
 D E S C R I P T E U R  (          27 )
 DEBUT DU DESCRIPTEUR...........
 DEBUT DESCRIPTEUR DE TABLEAU..
 NBDIM          1 NBELEM          3 LOGELEM          1
 DEBUT BORNES DU TABLEAU..
          0..          2
 FIN BORNES DU TABLEAU..
 D E S C R I P T E U R  (          27 )
 DEBUT DU DESCRIPTEUR...........
  ..NOMSIG..
 FIN DU DESCRIPTEUR...........
 FIN DESCRIPTEUR DE TABLEAU..
 FIN DU DESCRIPTEUR...........
 PAS DE DESCRIPTEUR.......(         28 )
 D E S C R I P T E U R  (          29 )
 DEBUT DU DESCRIPTEUR...........
 DEBUT DESCRIPTEUR DE TABLEAU..
 NBDIM          1 NBELEM          4 LOGELEM          1
 DEBUT BORNES DU TABLEAU..
          0..          3
 FIN BORNES DU TABLEAU..
 D E S C R I P T E U R  (          29 )
 DEBUT DU DESCRIPTEUR...........
  ..CARACTERE..
 FIN DU DESCRIPTEUR...........
 FIN DESCRIPTEUR DE TABLEAU..
 FIN DU DESCRIPTEUR...........
 PAS DE DESCRIPTEUR.......(         30 )
 PAS DE DESCRIPTEUR.......(         31 )
 ?
  ETS
 TABLE DES SIGNAUX
 VALEUR,GENRE,DONNEES ASSOC.,TRANSIN,TRANSOUT,EVT0,EVT1
        F ENTREE   PTLDATA  NILTRANS NILTROUT          -1          -1 (          1 )
        F ENTREE   PTLDATA  NILTRANS NILTROUT          -1          -1 (          2 )
        F SORTIE   PTLDATA  NILTRANS NILTROUT          -1          -1 (          3 )
        F SORTIE   NILDATA  NILTRANS NILTROUT          -1          -1 (          4 )
        F INTERNE  NILDATA  TRANSIN  TRANSOUT          -1          -1 (          5 )
        F INTERNE  NILDATA  TRANSIN  TRANSOUT          -1          -1 (          6 )
        F INTERNE  NILDATA  TRANSIN  TRANSOUT          -1          -1 (          7 )
        F INTERNE  NILDATA  TRANSIN  NILTROUT          -1          -1 (          8 )
        F INTERNE  NILDATA  TRANSIN  NILTROUT          -1          -1 (          9 )
        F INTERNE  NILDATA  TRANSIN  NILTROUT          -1          -1 (         10 )
        F INTERNE  NILDATA  TRANSIN  NILTROUT          -1          -1 (         11 )
        F INTERNE  NILDATA  NILTRANS NILTROUT          -1          -1 (         12 )
          1..          4
 LISTE VIDE...........
 LISTE VIDE...........
```

```
---------------------------- (          1 )
          2..          4
          3..          4
LISTE VIDE.....
LISTE VIDE.....
---------------------------- (          2 )
          1..          8
         26..          6
LISTE VIDE.....
LISTE VIDE.....
---------------------------- (          3 )
PAS DE BORNES...
LISTE VIDE.....
LISTE VIDE.....
---------------------------- (          4 )
PAS DE BORNES...
          1..
          3..          4..
---------------------------- (          5 )
PAS DE BORNES...
          2..
          1..          4..
---------------------------- (          6 )
PAS DE BORNES...
          3..
          2..
---------------------------- (          7 )
PAS DE BORNES...
          3..
LISTE VIDE.....
---------------------------- (          8 )
PAS DE BORNES...
          4..
LISTE VIDE.....
---------------------------- (          9 )
PAS DE BORNES...
          4..
LISTE VIDE.....
---------------------------- (         10 )
PAS DE BORNES...
          4..
LISTE VIDE.....
---------------------------- (         11 )
PAS DE BORNES...
LISTE VIDE.....
LISTE VIDE.....
---------------------------- (         12 )
?
 ETI
TABLE DES INDICATEURS
TEMPS,MOI,NBSIG,NBUNIT,NBIDC
```

```
        0          4          12          4          31
?
 ECT
TABLE DE CONTEXT
ADRESSES TABLES: T.C.,ENTIERS,BOOLEENS,CARACT.,SIGNAUX,UNITES,EVENTS.,FILS,ECHEANCIER LOCAL,INDICATEURS
T.C.     T.E.     T.B.     T.CHAR   T.SIG    T.U.     T.EVENT  T.FILS   NIL-ECH  INDIC
?EUN
TABLE DES UNITES
NOM UNITE,PTCONTEXT,AD.PROG,PERE,NBFILS,FILS,DATES,SUIVANT,PRECEDENT
UNIT3            PTCONTXT          1          2          0          0          0         -1         -1
LES FILS..
U3               PTCONTXT          2          4          1          3          0         -1         -1
LES FILS..          1...
UX               PTCONTXT          1          4          0          0          0         -1         -1
LES FILS..
ESSAI            PTCONTXT          3          0          2          1          0         -1         -1
LES FILS..          2...       3...
?
 EBL 1
IMPRESSION D'UN BLOC EVENEMENT
         1          0         -1 NIL-AV   NIL-APRE       F          0          1          1          1          0
 1
OPER
         5..
         6..
?
 EBL 2
IMPRESSION D'UN BLOC EVENEMENT
         2          4         -1 NIL-AV   NIL-APRE       F          0          2          1          1          0
 1
OPER
         6..
         7..
?
```

```
 EBL 3
IMPRESSION D'UN BLOC EVENEMENT
          3            0          -1 NIL-AV   NIL-APRE        F            0            3            2            1            0
 1
OPER
          7..          8..
          5..
?
 EBL 4
IMPRESSION D'UN BLOC EVENEMENT
          4            0          -1 NIL-AV   NIL-APRE        F            0            4            3            2            0
 2
OPER
          9..          10..         11..
          5..          6..
?FIN
FIN MODE TEST
?*
?*
?*-------------------------
?*
?ENVIRON U3
?UNITE
NOM UNITE COURANTE: U3
?*
```

Suite de la simulation

ANNEXE 3

SIMULATION AVEC IMPRESSION DE LA TRACE D'EXÉCUTION DU SIMULATEUR

```
 PASCEXEC
CORE?7Ø
FICHE?DFG
BINAIRE?SIM6
- SFER/PASCAL-SYSTEM,VERS. 1/11/76
?
?*
?*DEROULEMENT D'UNE SIMULATION
?* AVEC TRACE D'EXECUTION DEMANDEE
?*
?UNITE
NOM UNITE COURANTE: ESSAI
?!SYSTEST
SUPERVISEUR EN MODE TEST...
COMMANDES DISPONIBLES: EVS,ETC,ETS,ETI,ECT,EUN,EBL,FIN,ONS,OFS
?ONS
?FIN
FIN MODE TEST
?*
?*ONS' PERMET DE DEMANDER LA TRACE DE L'EXECUTION
?*
?RANGER S6=1
REPERCUTE....
ECHØ....          3
ECH1.....
ECHEANCIER...
SORTIE ECHEANCIER...
SORTIE REPERCUTE.....
ENLEVEGLOBAL          -1
SORTIE ENLEVGLOBAL        -1
?RANGER S3=1
REPERCUTE....
ECHØ....          1
ECH1.....
ECHEANCIER...
AJOUTE.....
PTRECH ETAIT NIL......
SORTIE AJOUTE...
SORTIE ECHEANCIER...
ECH3....          3
ECH4....
ECHEANCIER...
SORTIE ECHEANCIER...
ECH3....          4
ECH4....
ECHEANCIER...
SORTIE ECHEANCIER...
SORTIE REPERCUTE.....
```

Commande Siris 8

trace d'exécution pour la commande RANGER S6 = 1

début trace d'exécution du simulateur

```
AJOUTEGLOBAL
P1=          4          -1          -1
,,,,,         -1          -1
DATE AJOUT          0
ENLEVEGLOBAL LE MEME...
SORTIE AJOUTEGLOBAL          4
?SINULE 0
DATE MAX DE SIMULATION DEMANDEE          0
ACTIVE UNITE.....          4
ENLEVEGLOBAL          4
SORTIE ENLEVGLOBAL          -1
INITCONTXT.....          4
.......          0          0          0
CALL ENLEVE.....
NUMBLOC          1
BLOC TROUVE
SORTIE ENLEVE.....
CALL OPER...
VAL SORTIE ENTREE
MAJ01....
I-DUMMY          6
REPERCUTE....
ECH0....          2
ECH1.....
ECHEANCIER...
AJOUTE.....
PTRECH ETAIT NIL.....
SORTIE AJOUTE...
SORTIE ECHEANCIER...
ECH3....          1
ECH4....
ECHEANCIER...
MEME BLOC...
SORTIE ECHEANCIER...
ECH3....          4
ECH4....
ECHEANCIER...
SORTIE ECHEANCIER...
SORTIE REPERCUTE.....
SORTIEMAJ01...
VAL SORTIE FIN..
MAJ01....
I-DUMMY          5
REPERCUTE....
ECH0....          1
ECH2....
ECHEANCIER...
MEME BLOC...
SORTIE ECHEANCIER...
ECH3....          3
ECH5.....
```

```
ÉCHEANCIER...
SORTIE ECHEANCIER...
ECH3....          4
ECH5.....
ECHEANCIER...
SORTIE ECHEANCIER...
SORTIE REPERCUTE.....
SORTIEMAJ01...
ECHEANCIER...
SORTIE ECHEANCIER...
FIN INTERP. OPER.....
AJOUTEGLOBAL
P1=          4          -1          -1
,,,,,         -1          -1
DATE AJOUT          4
ENLEVEGLOBAL LE MEME...
SORTIE AJOUTEGLOBAL          4
...          4......
SIMULATION JUSQU'A LA DATE          0 TERMINEE
INITCONTXT.....          4
?*
?*ON CONTINUE LA SIMULATION
?*JUSQU'A LA DATE
?*
 .
 .
 .
 .
 .
 .
 .
 .
ENLEVEGLOBAL LE MEME...
SORTIE AJOUTEGLOBAL          4
...          4......
SIMULATION JUSQU'A LA DATE          4 TERMINEE
INITCONTXT.....          4
?*
?*ETAT DES SIGNAUX
?SIGNAUX
E1                       0
E2                       0
S1                       0
S2                       0
S3                       0
S4                       1
S5                       0
S6                       0
S7                       0
S8                       0
S9                       0
S10                      0
?*
?*
?*--ARRET DU MODE TRACE D'EXECUTION
?SYSTEST
SUPERVISEUR EN MODE TEST...
COMMANDES DISPONIBLES: EVS,ETC,ETS,ETI,ECT,EUN,EBL,FIN,ONS,OFS
?OFS
?EBL 2
 .
 .
 .
```

fin trace d'exécution pour la commande RANGER S3 = 1

Fin de simulation

CHAPITRE QUATRE

LE COMPILATEUR LASSO

I - INTRODUCTION

I-1 Organisation globale du compilateur

Le compilateur LASSO admet en entrée une description écrite en LASSO. Il effectue sur elle les vérifications syntaxiques et sémantiques et produit un module sous la forme d'un texte en langage PASCAL.

La compilation se fait en 2 phases :

1) La première phase comprend :

- L'analyse lexicographique et syntaxique d'une description contenant une ou plusieurs entités (éventuellement vides). Le résultat est la construction d'une structure de données interne pour le compilateur.

- La génération en ligne du code directement exécutable correspondant soit aux parties algorithmiques des "actions" ou de la partie condition de la primitive "selon", soit aux parties "prises de mesures".

2) La deuxième phase consiste à :

- compléter la table des unités dans la description analysée

- produire le code qui construira, à l'initialisation d'une simulation, la structure de données interne interprétable (cf paragraphe II, chapître 3) du modèle.

I-2 Outil d'aide à la construction du compilateur

Il existe des outils utilisant des techniques bien connues [16] qui permettent d'automatiser la construction d'une partie des compilateurs. Divers systèmes opérationnels existent tels que le système APARSE [12] qui traite des grammaires à attributs ou le système GSA (General Syntax Analyser) [14] qui dispose d'un langage de spécification de la syntaxe et de la lexicographie. Toutefois leur disponibilité au Laboratoire n'était pas immédiate. Une tentative a été faite

de transporter le système d'aide à l'écriture de compilateurs de la Faculté des Arts et des Sciences de l'Université de Montréal [20], [21] sur l'ordinateur IRIS80 du Laboratoire; cependant devant les difficultés rencontrées (délai d'obtention du système, particularités du système) nous avons abandonné ce projet.

Le seul système immédiatement disponible était le transformateur de Messieurs GRIFFITHS et PELLETIER. Ce système grandement amélioré par M. DELAUNAY a été adopté pour un traitement interprétatif en langage PASCAL.

I-3 Description et traitement de la syntaxe du langage LASSO

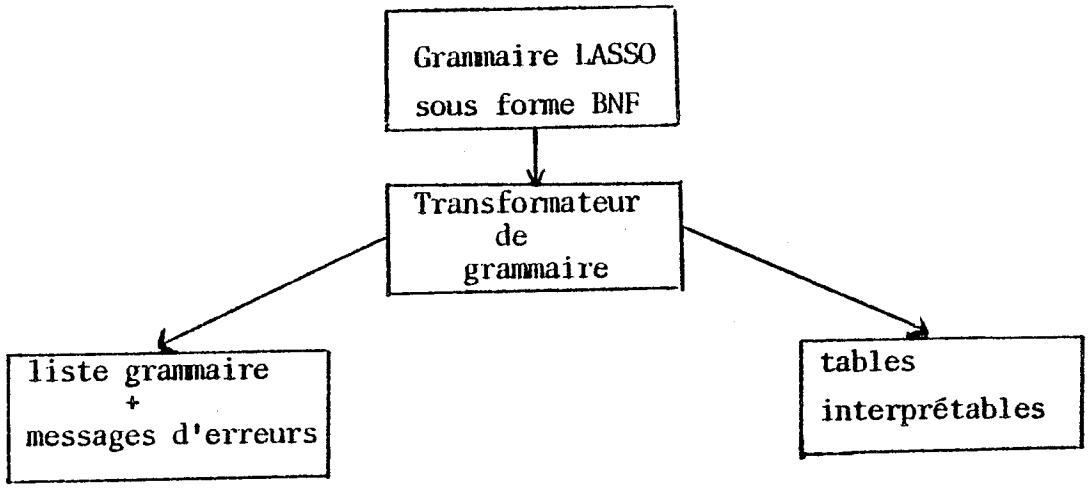
La syntaxe du langage LASSO est décrite par une grammaire hors-contexte LL(1). Cette grammaire, écrite suivant certaines conventions, est fournie à un transformateur de grammaire qui vérifie que la grammaire d'entrée obéit aux règles de définition d'une grammaire LL(1). Le transformateur fournit en sortie soit des tables interprétables soit un analyseur exécutable (écrit en PL/1 ou FORTRAN).

Dans le compilateur LASSO présent ce sont les tables interprétables qui sont utilisées. Cette méthode offre plus de souplesse dans la mise au point du compilateur et l'ensemble compilateur plus tables interprétables est moins encombrant en mémoire qu'un analyseur exécutable. Cependant, l'interprétation est plus coûteuse en temps d'exécution que l'analyseur directement exécutable.

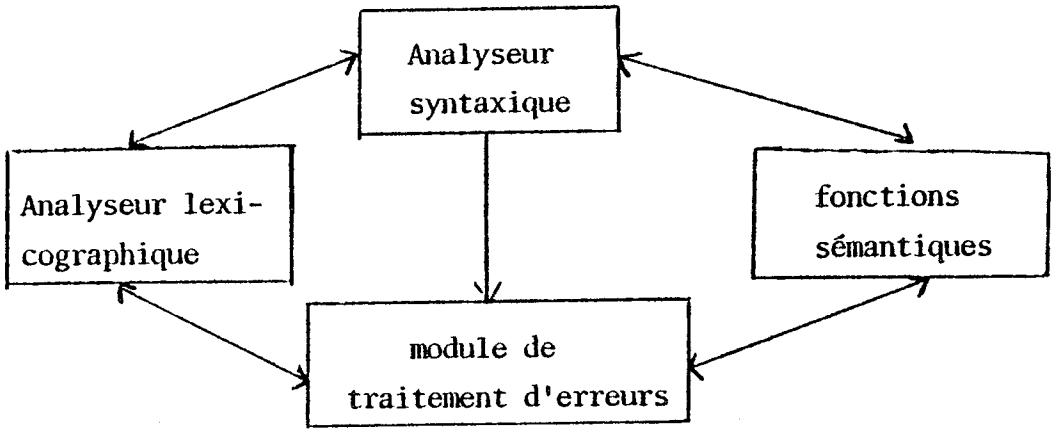
Nous rappelons brièvement les principes de fonctionnement du transformateur de grammaire utilisé.

I-4 Le transformateur de grammaire

Ce système accepte en entrée une description du langage sous forme d'un ensemble de règles de grammaire écrites, aux caractères de contrôle près, sous forme normale de Bakus. Le transformateur de grammaire vérifie que la grammaire d'entrée obéit aux règles de définition d'une grammaire LL(1) et fournit en sortie un ensemble de tables interprétables, les messages d'erreurs éventuels et une liste formatée de la grammaire (cf annexe du chapitre).



Ces tables, ainsi que leur interpréteur (qui sera décrit plus loin), constituent l'analyseur syntaxique descendant qui utilise les parties lexicale et sémantique comme des sous-programmes. Le schéma général d'un tel système est le suivant :



Cependant, la syntaxe ne spécifie que la forme et doit être complétée par la sémantique qui spécifie l'interprétation que l'on doit donner au langage. La façon la plus courante d'introduire cette interprétation est d'associer une ou plusieurs fonctions sémantiques à chaque règle syntaxique.

Pour éviter d'arrêter la compilation dès qu'une erreur a été détectée dans l'un des trois modules (syntaxique, sémantique ou lexical), il a été prévu un module de traitement des erreurs qui puisse être appelé par l'un des trois autres modules.

L'ensemble des 4 modules constitue la passe d'analyse syntaxique du compilateur.

II - PREMIERE PHASE DE COMPILATION

II-1 Analyseur lexicographique

L'analyseur lexicographique lit le texte source d'une description LASSO et reconnaît les unités lexicales; chaque unité lexicale reçoit un code interne. La correspondance unité lexicale - code interne est établie grâce aux 2 tables TMORES et TMOCODE décrites en annexes 1 de [30 (tome 1)]. Lorsqu'un terminal a été identifié par l'analyseur lexical, celui-ci transmet son code à l'analyseur syntaxique par l'intermédiaire de la variable globale CURRCHAR. Si le terminal est un identificateur, ce dernier est également transmis à l'analyseur syntaxique dans la variable globale INTERM.

L'ensemble des terminaux peut être divisé en quatre classes :

- la classe des mots réservés; la taille d'un mot ne peut dépasser 8 caractères
- la classe des symboles spéciaux simples (1 caractère) (+, -, / ...)
- la classe des symboles spéciaux doubles (= <, := ...)
- la classe des identificateurs; la longueur maximum d'un identificateur est paramétrable et a été fixée dans la version actuelle du compilateur à 16 caractères.

Il est nécessaire, pour effectuer des vérifications sémantiques, de distinguer les identificateurs d'unité, les identificateurs de procédures et de fonctions les autres identificateurs. Pour cela si un identificateur est reconnu 3 alternatives se présentent :

1) l'identificateur est suivi d'un caractère "souligné" dans un ordre de connexion, le code transmis dans CURRCHAR est 131 (identificateur d'unité)

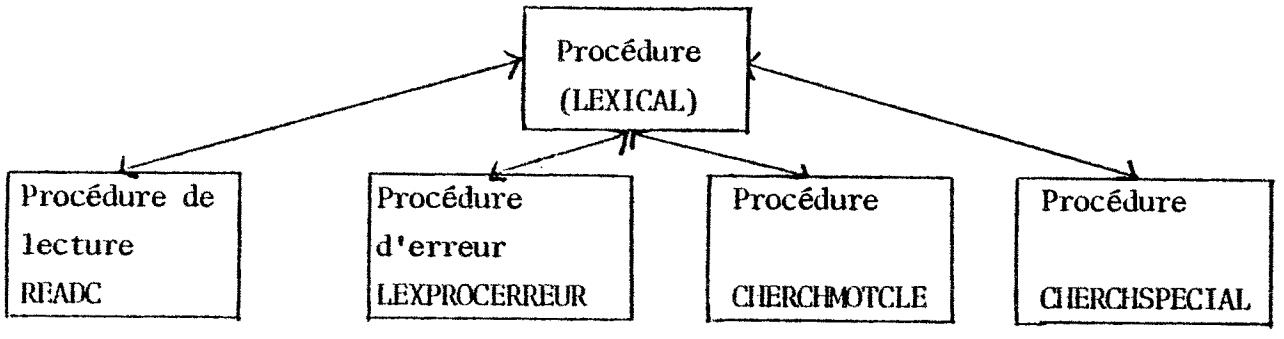
2) l'identificateur est suivi d'une parenthèse ouvrante, le code transmis dans CURRCHAR est 132 (identificateur de procédure ou fonction)

3) l'identificateur est suivi de tout autre caractère, le code transmis dans CURRCHAR est 133.

II-2 Organisation de l'analyseur lexicographique

Il comprend une procédure principale (LEXICAL) et 4 procédures réalisant les fonctions de :

- lecture du fichier source à analyser (READC)
- sortie de messages d'erreurs (LEXPROCERREUR)
- recherche de mots-clés dans la table TMORES (CHERCHMOTCLE)
- recherche d'un caractère spécial dans la table TMORES (CHERCH-SPECIAL).



Seule la procédure principale LEXICAL est décrite; les autres procédures sont détaillées dans [30 (tome 1)].

La procédure LEXICAL a pour but de reconnaître dans le texte à compiler une unité lexicale et d'en fournir le code à l'analyseur syntaxique. Cette reconnaissance se fait en examinant successivement les alternatives suivantes :

1) Test si la fin du fichier texte source a été atteinte; si oui, la variable CURRCHAR reçoit le code fin de fichier EOFILE (code 134) et le contrôle est rendu à la procédure ayant appelé LEXICAL.

2) Elimination des commentaires compris entre deux caractères "%". Le caractère % ne peut pas être utilisé dans la partie commentaire.

3) Elimination des caractères "blancs".

4) Identification, dans cet ordre, des unités lexicales suivantes :

   * ou *1 ou *0
   < ou <- ou <= ou <>
   > ou >=
   => ou =<
   : ou :=
   - ou →

5) Test si l'unité lexicale est un identificateur ou un mot-clé, le premier caractère étant l'un des caractères de l'alphabet.

6) Si l'unité lexicale est un chiffre décimal rend dans la variable CURRCHAR sa valeur.

7) A ce point, l'unité lexicale est soit un caractère spécial, soit un caractère invalide.

Au cours du traitement de l'une des 7 alternatives citées (sauf la première), une des trois erreurs suivantes peut être détectée :

- la rencontre d'une fin de fichier signifiant que la description est incomplète,
- la longueur d'un identificateur est supérieure à la longueur maximum autorisée,
- la rencontre d'un caractère invalide.

L'indicateur LEXERREUR est chargé avec le code de l'erreur, un message est imprimé (appel de LEXPROCERREUR) et le contrôle est rendu à l'analyseur syntaxique.

II-3 Analyseur syntaxique

II-3-1 Sa fonction

Le noyau de l'analyseur syntaxique est l'analyseur interprétatif descendant des tables fournies par le transformateur de grammaire LL(1). Il guide le déroulement de la compilation en assurant la séquence des appels à l'analyseur lexicographique et aux fonctions sémantiques. Il a les fonctions suivantes :

1) Il appelle l'analyseur lexical pour obtenir le code de la prochaine unité lexicale à traiter,
2) Il invoque (par l'intermédiaire de la procédure SEMFCT) les fonctions sémantiques associées à la règle de grammaire en cours de traitement.

Ces fonctions :

a) initialisent en début de compilation d'une description, les variables globales au modèle,

b) initialisent les variables associées à la compilation d'une nouvelle entité,

c) construisent la structure de donnée interne du compilateur pour le modèle,

d) génèrent le code objet en fin de compilation,

e) impriment les messages d'erreurs détectés (autres que les messages d'erreurs de syntaxe).

3) Il signale les erreurs syntaxiques rencontrées dans le texte source et essaie de poursuivre l'analyse de la description.

Nous nous intéressons uniquement à deux procédures qui sont :

. la procédure d'erreur LL1ERROR
. la procédure d'appel des actions sémantiques SEMFCT

et à la table des symboles "importants" TBLSYMB. Ce sont en effet les seuls objets qui doivent être connus de l'écrivain de compilateur et qui sont susceptibles d'être modifiés après une modification de la grammaire du langage.

II-3-2 Procédure LL1ERROR

Un système de détection et de rattrapage d'erreurs simple est intégré dans l'interpréteur et permet par appel de la procédure LL1ERROR, d'imprimer l'un des messages suivants :

1) "pile vide" la pile de récursivité est vide
2) "erreur de syntaxe n° X, symbole Y".
3) "analyse abandonnée"
   Le mécanisme est défaillant
4) "Symbole erroné; symbole attendu X, symbole courant Y".

L'analyseur est dans une alternative et attend un symbole précis qui, dans ce cas, n'est pas le bon symbole.

II-3-3 Procédure SEMFCT

Cette procédure gère les appels aux fonctions sémantiques. Elle est appelée avec en paramètre le numéro de la fonction sémantique à exécuter, la numérotation des fonctions étant assurée par le transformateur de grammaire.
Lors d'ajout ou de suppression de fonctions sémantiques dans la grammaire, cette procédure doit être mise à jour.

II-3-4 Table TBLSYMB

Cette table fait partie de l'interpréteur descendant et n'est pas générée par le transformateur de grammaire. Elle est initialisée au début de l'interpréteur avec la liste des symboles "importants" fournie par l'écrivain de compilateur. Ces symboles importants sont les symboles à partir desquels l'analyseur peut reprendre l'analyse d'une description lorsqu'il vient de rencontrer une erreur.

II-4 La structure de données du compilateur

Une description en LASSO est composée d'unités imbriquées ou de même niveau, toutes englobées dans l'entité la plus externe. Le compilateur traite cette description entité par entité et indépendamment les unes des autres. Ce principe de compilation induit que chaque composante de la structure de données soit propre à chaque entité.

L'organisation des données est très similaire à celle que nous avons décrite dans le chapître 3 du simulateur. Cette structure est construite lors de la compilation; elle est utilisée dans la phase 2 pour générer la structure de données interne interprétable (cf paragraphe II, chapître 3) du modèle compilé.

II-4-1 Structure de données propre à chaque entité

Cette structure construite lors de la compilation d'une entité comprend :

1) la table des identificateurs d'une entité
2) la table des blocs évènements construite à partir des transitions
3) la table des blocs connexions (une entrée par connexion déclarée).

A) La table des identificateurs

Elle est destinée à stocker tous les identificateurs apparaissant dans une entité à l'exception des identificateurs de compteurs de la partie mesure éventuellement associée. La structure d'une entrée de la table est la suivante :

. partie identificateur et type

NOMID ..... Nom de l'identificateur

TYPEID ..... Code du type de l'identificateur [30 (TOME 1) annexe 1]

PROCFONC ..... Code de la catégorie de l'identificateur :
0 : identificateur
1 : identificateur de procédure
2 : identificateur de fonction

. partie descripteur

PTDESCRIPT ..... Pointeur sur le descripteur de l'identificateur dans le cas d'un tableau ou d'une structure.
Dans le cas d'une procédure ou fonction pointe sur la liste des paramètres formels.

NBDIM ..... Nombre de dimensions d'un tableau

NBELEM ..... Nombre total d'éléments d'un tableau

. partie initialisation à la compilation

INITIALE ..... VRAI si une directive INIT est associée à l'identificateurs
FAUX sinon; dans ce cas les 2 champs INDICEDEBUT, LONGINIT sont sans signification.

INDICEDEBUT ..... Indice sur la table des valeurs d'initialisation d'une variable

LONGINIT ..... Nombre de valeurs d'initialisation pour cette variable

DEPLACT ..... Adresse (déplacement) de la zone valeur de la variable

. Les 5 champs suivants n'ont de signification que pour des identificateurs de type signal

ENTREEBLOC ..... Pointe la liste des blocs évènement ayant ce signal en entrée

SORTIEBLOC ..... Pointe la liste des blocs évènement ayant ce signal en sortie

DONNEEASS ..... Pointe la liste des adresses des données associées au signal

PTEVTO ..... Si une mesure est associée à ce signal, ce champ est l'indice sur la première action mesure à réaliser lorsque la valeur du signal passe de 1 à 0. Sinon ce champ a la valeur -1.

PTEVT1 ..... Si une mesure est associée à ce signal, ce champ est l'indice sur la première action mesure à exécuter lorsque la valeur du signal passe de 0 à 1. Sinon ce champ a la valeur -1.

B) La table des blocs évènements

Pour chaque primitive de synchronisation déclarée dans une entité ou pour chaque bloc mesure présent dans la partie mesure est construite une entrée de cette table. La structure d'une entrée est variable suivant le type de la primitive traitée; chaque entrée comprend 2 parties :

une partie commune à toutes les primitives contenant :

. TEMPSEXEC ..... Valeur du délai de la transition

. NBENTREE ..... Nombre de signaux déclarés en entrée principale de la transition

. NBSORTIE ..... Nombre de signaux déclarés en sortie de la transition

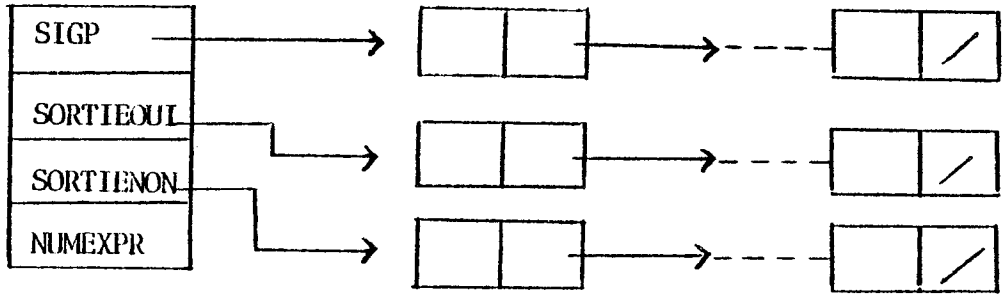
. FONCALG ..... Adresse dans la procédure ALGORITHME du code objet directement exécutable correspondant au corps de la primitive "action" ou "selon"

. TYPEBLOC ..... Code [30 (tome 1) annexe 3] du type du bloc évènement en cours de traitement qui détermine le format de la partie variable en fonction de la primitive ou du bloc mesure.

une partie variable

Nous donnons le nom de la primitive et la structure correspondante :

. Bloc SELON



SIGP ..... Pointeur sur la liste des signaux en entrée principale de la transition

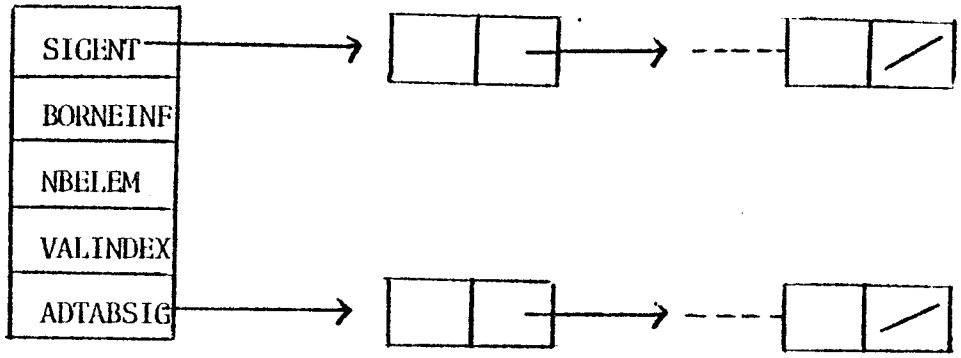
SORTIEOUI ..... Pointeur sur la liste des signaux à valider si l'expression booléenne est vraie.

SORTIENON ..... Pointeur sur la liste des signaux qui doivent être validés si l'expression booléenne est fausse.

Pour les deux listes citées ci-dessus, chaque élément de la liste contient le déplacement du signal dans la table des signaux.

NUMEXPR ..... Adresse du code programme dans la procédure ALGORITHME permettant l'évaluation de l'expression booléenne.

. Bloc INDEX



SIGENT ..... Pointeur sur la liste des signaux en entrée principale de la transition

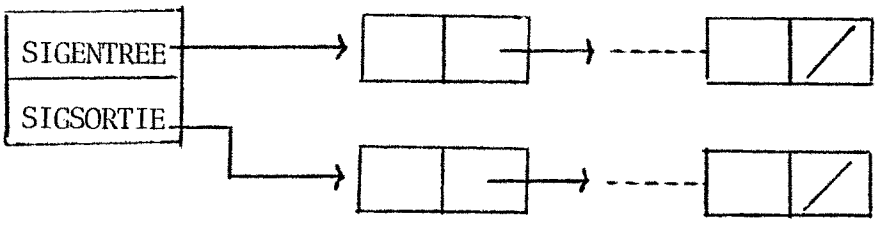
BORNEINF ..... Valeur de la borne inférieure de l'index

NBELEM ..... Indique le nombre de signaux qui sont en sortie de la transition et qui sont indéxables par l'index de la primitive

VALINDEX ..... Déplacement dans la table des valeurs entières de la zone valeur de la variable d'index

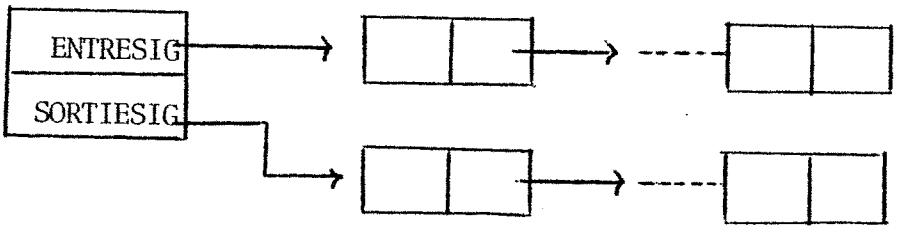
ADTABSIG ..... Pointeur sur la liste des signaux en sortie de la transition qui sont indexables.

. Bloc ACTION



SIGENTREE et SIGSORTIE sont deux pointeurs sur respectivement la liste des signaux en entrée, et la liste des signaux en sortie de la transition.

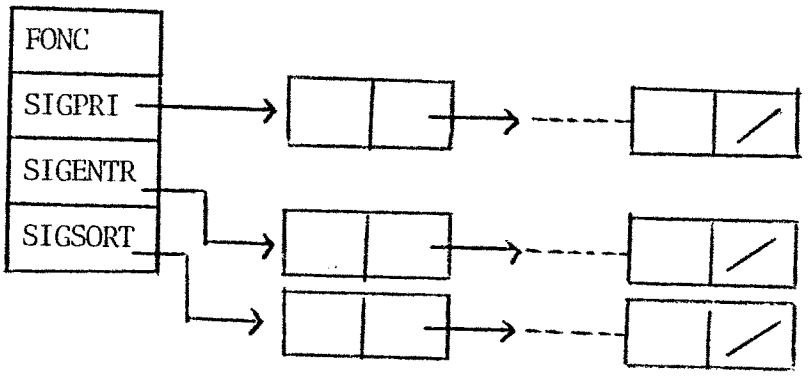
. Bloc ET



ENTRESIG et SORTIESIG sont deux pointeurs sur respectivement la liste des signaux en entrée, et la liste des signaux en sortie de transition. Chaque élément de la liste contient le déplacement d'un signal dans la table des signaux de l'unité.

. Bloc CHOIX ou CHOIXP

La même structure de données peut être utilisée pour ces deux primitives. La distinction se fera au niveau du code fonction.



FONC ..... -1 code pour CHOIX
1 code pour CHOIXP

SIGPRI ..... Pointeur sur la liste des signaux en entrée principale de la transition

SIGENTR ..... Pointeur sur la liste des signaux en paramètre de CHOIX ou CHOIXP

SIGSORT ..... Pointeur sur la liste des signaux en sortie de la transition

. Bloc MESURE

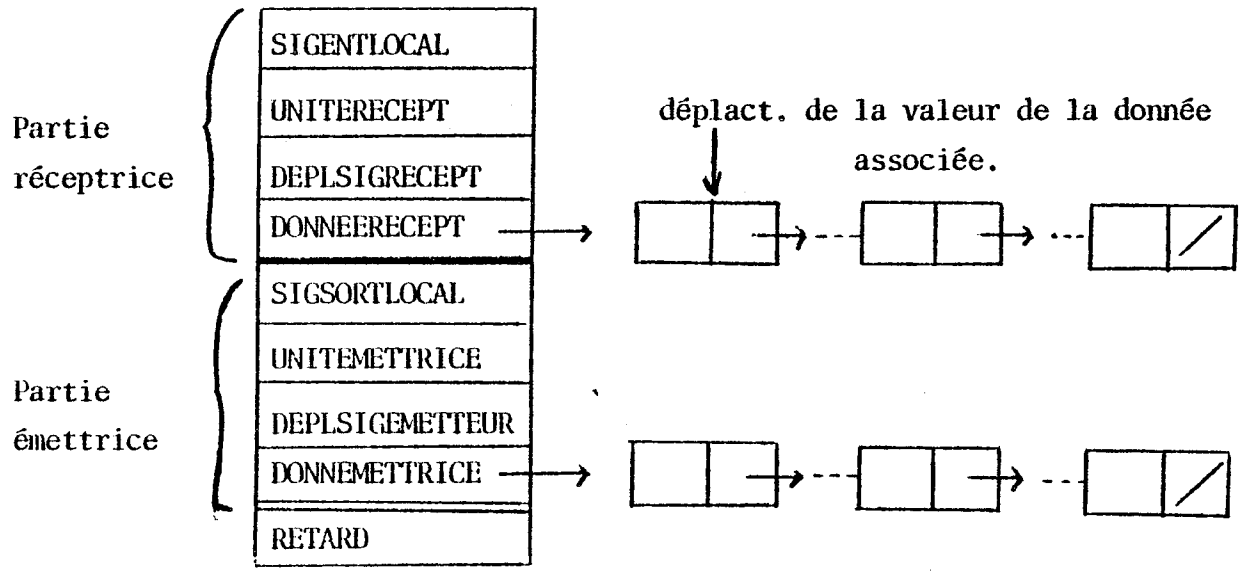
La partie variable est vide pour les blocs de type mesure.

C) Table des blocs connexion

Cette table est construite par le compilateur lorsque la déclaration de connexions d'unités est rencontrée. Toute unité ou signal doit être déclaré avant d'être réfférencée dans la connexion. Deux types de signaux peuvent apparaître dans les connexions :

- des signaux locaux à l'entité en cours de traitement auxquels peuvent être associées des données
- des signaux d'interface d'une autre unité qui sont dans ce cas qualifiés par le nom de l'unité à laquelle ils appartiennent.

Chaque entrée comprend une partie réception et une partie émission.



Partie réceptrice

SIGENTLOCAL ..... A la valeur VRAI si le signal est un signal local à l'entité en cours de compilation, FAUX sinon.

UNITERECEPT ..... Est le numéro de l'unité contenant le signal récepteur. C'est également l'indice de l'unité dans la table des unités.

DEPLSIGRECEPT ..... Déplacement dans la table des signaux du signal récepteur (en partie gauche de la connexion).

DONNEERECEPT ..... Dans le cas d'un signal local, est le pointeur sur la liste des données associées au signal. Ces données associées sont déclarées par leur occurrence dans la déclaration de connexion.

Partie émission

SIGSORTLOCAL ..... Si le signal émetteur est local à l'unité qui contient la déclaration de connexion, ce champ est positionné à la valeur VRAI, sinon il a la valeur FAUX.

UNITEMETTRICE ..... Est le numéro de l'unité émettrice. Ce numéro correspond à son indice dans la table des unités.

DEPLSIGEMETTEUR .... Déplacement du signal dans la table des signaux de l'unité de numéro UNITEMETTRICE.

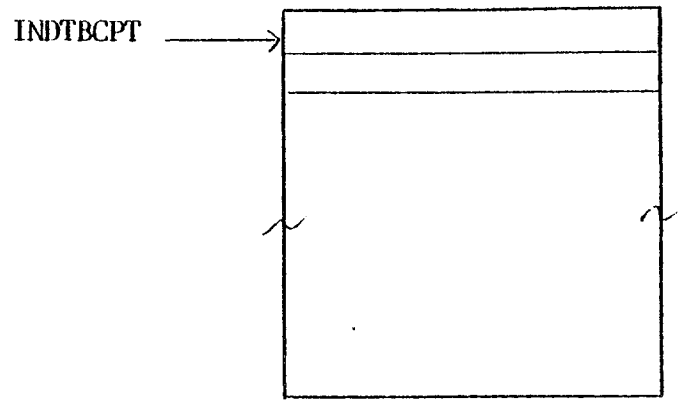
DONNEMETTRICE ..... Est le pointeur sur la liste des données associées au signal local. Cette zone n'a de signification que si SIGSORTLOCAL est VRAI.

RETARD ..... Est la valeur du retard sur la ligne de transmission.

II-4-2 Structure de données de la partie mesure

A) La table des déclarations de compteurs

La partie mesure peut contenir des déclarations locales d'identificateurs de compteurs. Ces identificateurs, non accessibles interactivement en cours de simulation, sont stockés dans la table des compteurs; les compteurs sont des variables scalaires à valeur entière.



TBNIDCPT
Table des identificateurs
de compteurs

B) Table d'évaluation des évènements mesure

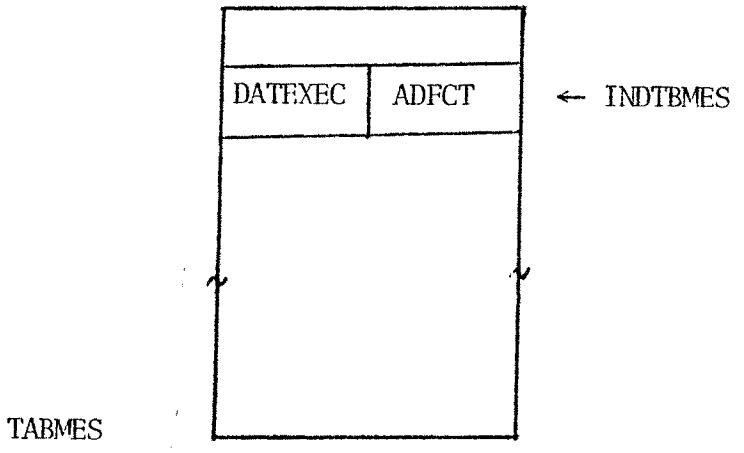
Elle est en tout point semblable à celle décrite au niveau du simulateur chapître 3, paragraphe II-1-4. Elle est intégralement recopiée à la phase génération du code objet pour être transmise au simulateur.

On trouve les algorithmes détaillés de sa construction dans [27]. Nous ne donnons ici que les propriétés que nous voulons exprimer dans cette structure :

. La représentation des évènements mesures doit permettre leur évaluation dans le temps.
. La structure d'un évènement est non-réentrante : lorsqu'un évènement est en cours d'évaluation, il faut attendre la validation finale avant de recommencer à évaluer cet évènement. Ainsi, si un évènement simple, composant cet évènement final, reçoit plusieurs validations au cours de l'évaluation de l'évènement, seule la première validation sera enregistrée, les autres étant perdues. La structure doit donc être "sans mémoire".
. On doit pouvoir invalider temporairement ou définitivement tous les évènements simples composant un évènement après la validation de ce dernier. Ceci parce que certains évènements ne sont plus réévalués durant la simulation, ou qu'ils ne peuvent être évalués qu'entre certaines dates. (cf primitives tousles et lafoisno).
. Pour la même raison que précédemment, on doit pouvoir réévaluer un évènement qui a été invalidé temporairement.
. On doit avoir une structure suffisamment simple et performante pour ne pas accroître fortement les temps de simulation.

C) Table des blocs évènements mesures

Pour les ordres de mesure dont les activations peuvent être connues dès la compilation (par exemple tousles x ...) une table des évènements est créée. Elle a la structure suivante :



DATEXEC : date de la première activation du bloc.

ADFCT : numéro du choix dans la procédure FCTMESURE qui indique le code à éxécuter à la simulation lors de l'activation du bloc évènement mesure.

Les évènements représentés dans cette table ne nécessitent aucune évaluation en cours de simulation et ne sont pas représentés dans la table d'évaluation des évènements mesures;

D) Structure de données de travail

Deux piles sont utilisées pour la compilation des mesures. La pile PILE sert à la traduction du texte source des évènements écrits sous forme infixée en expressions postfixées dans la table des évènements. La pile PILE2 sert à mémoriser toutes les primitives apparaissant dans le bloc interne à une fenêtre [27] pour générer le code correspondant.

II-4-3 Structure de données globale à une description

Ces structures sont construites à la compilation d'un modèle. Elles permettent d'accéder en cours de compilation d'une entité à des informations appartenant à toutes les entités et unités précédemment compilées ou déclarées respectivement.

A) Table des entités

Pour chaque définition d'entité on réserve une entrée de la table. Elle permet lors de la compilation d'une entité, l'accès aux informations d'autres entités. Chaque entrée a la structure suivante :

. partie identification

NOMENTITE est l'identificateur de l'entité.

NINTEX si VRAI le texte de l'entité en cours de compilation est interne à une autre entité. Si FAUX le texte de l'entité compilée n'est englobée dans aucune entité.

ENTPERE dans le cas d'une entité interne est le numéro de l'entité englobante.

. partie contexte

ORIGINETBIDENT → adresse de la table des identifications pour l'entité

LONGTBIDENT → longueur de la table des identificateurs

ORIGTBCONEX → adresse début de la table des blocs connexion

LGTBCONEX → longueur de la table des connexions

ORIGINEVENT → adresse début de la table des blocs transitions

LGTBEVENT → longueur de la table des transitions

LONGENTIER → taille de la zone valeur pour les entiers

LONGBOOL → taille de la zone valeur pour les booléens

LONGCARACT → taille de la zone pour les caractères

LONGSIG → taille de la table des signaux

. partie "génératrice"

NBUNITGENEREE nombre d'unités générées à partir de cette entié

PTUNITES pointeur sur la liste des numéros d'unités générées par l'entité

NPTYPARAM pointeur sur la liste des paramètres formels d'entité.
Chaque élément de la liste comprend :
. le type du paramètre formel (codage [30 (tome 1) annexe 3])
. l'identificateur du paramètre formel
. le pointeur sur l'élément suivant de la liste.

B) Table des unités

Elle est construite au fur et à mesure de la déclaration des unités dans la description. Une entrée est réservée par unité et chaque entrée à la structure suivante :

NUNITE est l'identificateur de l'unité

NNUMPERE c'est le numéro de l'entité père dans laquelle est déclarée l'unité

ENTGENERIQUE est le numéro (l'indice dans la table des entités) de l'entité générique

LEPERE numéro de l'unité père dans le graphe des unité du modèle LASSO.

PTPAREFFECTIF Pointeur sur la liste des paramètres effectifs de l'unité.

II-5 Génération en ligne du code objet directement exécutable

La partie code objet directement exécutable peut être le résultat de la compilation :

- d'une primitive action. Elle correspond à l'expression d'un algorithme à l'aide d'un langage de type PASCAL (cf chapître premier) et est délimité par début et fin. On y trouve :

  . des instructions de contrôle (tantque, si ...)
  . des appels de procédures
  . des expressions et des affectations.

- d'une action de prise effective de mesure (inc, date ...)

- de la primitive selon qui comprend en paramètre l'évaluation d'une expression booléenne.

Le code objet est produit par des actions de compilation appelées lors de l'analyse syntaxique d'une description. Il est rangé dans le fichier code généré (cf chapître deux) et se présente sous forme d'instructions PASCAL correctes syntaxiquement et sémantiquement.
Dans les expressions arithmétiques et logiques, on vérifie que tous les identificateurs référencés ont été déclarés. Des vérifications de compatibilités entre les types de variables, de compatibilités dimensionnelles, de compatibilités de type de résultat sont faites.
En ce qui concerne les instructions de contrôle, un traitement complet est fait et le parenthésage (déterminé par début et fin) est vérifié.

II-5-1 Traitement des déclarations des procédures et fonctions externes

Lors de l'appel du compilateur LASSO (version IRIS80), l'utilisateur a la possibilité de préciser le nom de la bibliothèque où se trouvent les procédures ou fonctions externes référencées dans sa description. L'ensemble de la bibliothèque est alors recopié dans le fichier code généré et sera compilé avec l'ensemble du fichier.
Dans l'état actuel du compilateur prototype :

. les procédures et fonctions de la bibliothèque sont écrites en PASCAL et sont sous forme de programmes source (éventuellement paramétrés).
. l'utilisateur ne peut déclarer qu'une seule bibliothèque de programmes.

Dans une version ultérieure, il est prévu d'avoir :

. une bibliothèque précompilée de programmes
. la possibilité de déclarer plusieurs bibliothèques utilisateur
. la possibilité d'écrire les procédures et fonctions de la bibliothèque dans le même langage que celui de la partie algorithmique de la transition "action"; ceci évitera à l'utilisateur d'apprendre deux langages différents.

II-5-2 Construction de la procédure ALGORITHME

Cette procédure (qui peut être vide), comprend toutes les parties algorithmiques de toutes les entités d'une description LASSO. Elle est unique pour tout le modèle.
Sa structure correspond à deux instructions CAS imbriquées. Le premier niveau porte sur le numéro d'entité dont dérivent la ou les unités, le second niveau porte sur le numéro de transition; les 2 niveaux sont fournis en paramètres de la procédure.

```
PROCEDURE ALGORITHME (numéro entité, numéro transition);
CAS % numéro entité %
      1 : DEBUT % entité numéro 1 %
          CAS % numéro transition %
             1 : DEBUT
                  % partie algorithmique de la transition
                    numéro 1 de l'entité numéro 1 %
                 FIN
             2 : DEBUT
                  :
                 FIN
          FINCAS % sur numéro transition %
      . FIN % entité numéro 1 %
      2 : DEBUT % entité numéro 2 %
          CAS % numéro transition %
             1 : DEBUT
                  % partie algorithmique de la transition
                    numéro 1 de l'entité numéro 2 %
                 FIN
          FINCAS % numéro transition %
        FIN % entité numéro 2 %
     FINCAS % numéro entité %
FIN % Procédure %
```

Si plusieurs exemplaires de la même entité ont été déclarés dans la description, le code objet directement exécutable correspondant à cette entité n'existe qu'en un seul exemplaire.

Lors de la simulation, tout appel correspondant à l'exécution d'une primitive "action" ou "selon" est transformé en un appel à la procédure ALGORITHME avec en paramètre le numéro de l'entité dont dérive l'unité courante et le numéro de la transition en cours d'interprétation.

II-5-3 Construction de la procédure FCTMESURE

Cette procédure, qui peut être vide, contient l'ensemble des ordres de prise de mesures qui sont :

- des appels à des procédures prédéfinies dans le simulateur
- des instructions PASCAL correspondant aux conditions restrictives sur les variables de l'entité à laquelle sont associées les mesures et aux instructions de mesures directement exécutables par le simulateur.

Elle possède un paramètre qui est le numéro du bloc mesure à activer. Dans l'état actuel du compilateur, sa structure est celle d'une instruction CAS à un seul niveau. Chaque alternative correspond à un bloc prise de mesure.

La procédure est unique pour l'ensemble de la description. Deux types de variables sont référencées dans la partie mesure :

- les variables déclarées au niveau de l'entité et qui sont transformées en références à des zones valeurs,

- les compteurs qui sont des objets locaux à la partie mesure d'une entité. Les références à ces compteurs sont également traduites en référence à une zone des valeurs.

Remarque : Dans la version actuelle IRIS80 du compilateur LASSO, l'expression des mesures n'est possible que pour la dernière entité du modèle. Ceci est dû à une limitation du nombre de fichiers dans le compilateur PASCAL, qui oblige à générer le code objet correspondant aux mesures à la suite de la partie algorithmique. Dans une version prochaine du compilateur LASSO, il sera possible d'associer des mesures à toute entité d'une description. Ceci implique :

1) de créer un fichier supplémentaire contenant la procédure FCTMESURE précédée de l'entête standard identique à celle du fichier code généré,

2) d'organiser la procédure FCTMESURE sous forme de 2 instructions CAS imbriquées.
Le premier niveau correspond au numéro de l'entité, le second niveau correspond au numéro du bloc mesure à activer.

Lors de la simulation, l'exécution des ordres de prises de mesures pour l'unité active sera lancée par appel à la procédure FCTMESURE, qui aura pour paramètres le numéro de l'entité dont dérive l'unité active et le numéro du bloc mesure à activer.
Chaque partie mesure est spécifique d'une entité et est compilée avec cette entité. Il y a actuellement duplication du code objet des prises de mesures pour chaque exemplaire d'une unité issue d'une même entité, possédant une partie mesure. Avec la nouvelle version proposée, il n'existera plus qu'un seul exemplaire du code des mesures pour tous les exemplaires d'unités issues de la même entité.
La procédure FCTMESURE construit dynamiquement à la simulation un fichier "résultats de mesures" accessible et exploitable hors simulation.

```
PROCEDURE FCTMESURE(numéro entité, numéro bloc mesure);
    CAS numéro entité
      1 : % entité 1 %
        DEBUT
          CAS % numéro bloc mesure %
              DEBUT
                1 : % bloc mesure 1 %
                    DEBUT
                    : % ordres de prises de mesures %
                    FIN
                2 : % bloc mesure 2 %
                    DEBUT
                    :
                    FIN
              FIN
          FINCAS % numéro bloc mesure %
    FIN % entité 1 %
      2 : % entité 2 %
          DEBUT
          :
          FIN
    FINCAS % numéro entité %
FIN % Procédure %
```

III - DEUXIEME PHASE DE COMPILATION

Cette passe est appelée lorsque toute la description LASSO a été analysée et si aucune erreur fatale n'a été détectée au cours de la première passe. Elle réalise les traitements suivants dans l'ordre chronologique :

1) complète la table des unités du modèle compilé
2) génère pour chaque unité déclarée des ordres de réservation de la structure interne interprétable de l'unité (cf chapitre 3)
3) génère pour chaque unité, la table de ses unités fils
4) génère le graphe d'imbrication des unités du modèle
5) génère la suite des appels aux procédures d'initialisation de la structure de données interne interprétable du modèle
6) imprimer des informations statistiques sur le modèle compilé.

III-1 Compléter la construction de la table des unités du modèle compilé

Toutes les entités du modèle doivent être compilées dans un ordre qui évite de référencer une entité inconnue; ceci revient à ce que la compilation aille depuis les feuilles jusqu'à la racine de l'arbre d'imbrication des unités qui sont des exemplaires de ces entités.
Un modèle simulable ne doit contenir que des imbrications d'unités dérivées d'entités pour lesquelles tous les paramètres (optionnels) ont reçu une valeur. Afin d'éviter à l'utilisateur une déclaration artificielle d'unité au niveau du modèle, par convention, nous considèrerons que l'entité la plus englobante est une unité. Il est donc nécessaire de compléter la table des unités produite pour cette entité par une unité racine qui est cette entité elle-même. C'est le rôle de la procédure ENTENGLOBE, qui teste si la description comporte une seule entité; si oui, la table des unités est complétée par le transfert de l'identificateur de l'entité unique vers l'entrée de la table des unités.
Sinon, une vérification de la cohérence du graphe d'imbrication des unités est faite, et permet de détecter que toutes les unités (sauf la plus englobante ont une unité père. Dans le cas contraire, un message d'erreur est imprimé.

III-2 Génération des ordres de réservation de la structure interne interprétable du modèle

Les ordres sont une séquence d'appels à des procédures externes défines dans le simulateur. Ils sont générés unité par unité et sont activés une seule fois en début de simulation pour construire les différentes tables représentant le modèle interne interprétable.

Le volume du code généré est fonction de la description LASSO. Pour des raisons de limitations de taille de prodédure en PASCAL (version IRIS 80), nous avons été conduits à faire un découpage arbitraire du code en générant dynamiquement des procédures qui regroupent des séquences d'ordres.

Pour chaque unité, l'ensemble des ordres est composé de 9 groupes d'appels de procédures, qui sont dans cette séquence :

1) les ordres de construction des 2 tables propres à chaque unité (table des indicateurs et table de contexte)
2) les ordres de construction de la table des signaux
3) les ordres de construction de la table de correspondance
4) les ordres de construction de la table des adresses des blocs évènements construits pour les connexions.
5) les ordres de construction de la table des adresses des blocs évènements construits pour les transitions
6) les ordres d'initialisation de la table de contexte
7) les ordres d'initialisation de la table des indicateurs
8) les ordres de construction d'une entrée de la table des unités
9) les ordres de calcul pour les nouvelles bases des tables des signaux, correspondance ... pour l'unité suivante.

On trouvera dans [30 (tome 1)] le détail des différents points énumérés appuyés d'un exemple.

III-3 Génération d'informations statistiques

En fin de compilation d'une description LASSO, le compilateur écrit dans le fichier code généré (sous forme de commentaires) des informations sur les tailles des différentes tables. Ces informations pourront, par la suite, faire partie de la liste de la description sortie par le compilateur LASSO et permettant de connaître grossièrement l'encombrement du modèle en mémoire.

Exemple :

```
%GENERATION DES CONSTANTES DE LONGUEUR DES TABLES
 CONST
 MAXID=66;TAILLE TABLE CORRESPONDANCE
 MAXENT=32;TAILLE TABLE DES ENTREES
 MAXBOOL=0;TAILLE TABLE DES BOOLEENS
 MAXCHAR=0;TAILLE TABLE CARACTERES
 MAXSIG=43;TAILLE TABLE DES SIGNAUX
 MAXUNIT=3;TAILLE TABLE UNITES
 MAEVENT=40;TAILLE TABLE EVENTS.
 MAXFILS=2;TAILLE TABLE DES FILS%
     %FIN INITIALISATION DU MODELE%
```

IV - LES ACTIONS DE COMPILATION (aussi appelées fonctions sémantiques)

Elles permettent de donner une interprétation aux différents objets du langage. Elles sont invoquées au cours des 2 phases de compilation soit automatiquement par l'interpréteur descendant de la grammaire (cf paragraphe II-3) soit par d'autres actions appelées automatiquement. Le compilateur comporte 216 actions réparties en 9 classes. On trouvera une présentation détaillée des actions dans [30 (tome 1)].

IV-1 Les actions de compilation générales

Invoquées au cours de la compilation d'une des parties d'une entité elles positionnent des variables globales, consultant des tables, initialisent des variables globales, écrivent des symboles et mots clefs dans le fichier code généré.

IV-2 Les actions de traitement des déclarations d'entités

Elles assurent le traitement :

1) des entités internes ou externes et de leurs paramètres formels
2) la mise à jour de la table des entités.

IV-3 Les actions de traitement des déclarations d'unités

La déclaration d'une nouvelle unité entraîne :

1) la mise à jour de la table des unités
2) la construction de la liste des paramètres effectifs
3) la vérification de la compatibilité entre paramètres formels et effectifs.

IV-4 Les actions de traitements des déclarations d'objets LASSO

Elles manipulent la table des identificateurs et déterminent si l'identificateur traité est un scalaire ou un tableau. L'initialisation d'une variable à la compilation implique que les actions :

1) sauvegardent les valeurs d'initialisation
2) contrôlent leurs compatibilités de types et dimensionnelles avec l'identificateur déclaré.

IV-5 Les actions de traitement des expressions arithmétiques et logiques

La compilation des expressions arithmétiques et logiques ainsi que celle des instructions de contrôle (tantque, si ...) sont des mécanismes connus. Pour l'instant les opérateurs de type APL portant sur des tableaux ne sont pas traités. Notons qu'un traitement particulier doit être fait pour les variables de type nomsig apparaissant comme opérandes de l'opérateur d'affectation (:=). Le code objet PASCAL est généré au fur et à mesure de l'analyse de la partie algorithmique.

IV-6 Les actions de traitement des transitions

Elles construisent la table des blocs évènements à raison d'une entrée par transition. En fin de compilation cette table est interprétée pour permettre la génération de code de réservation des blocs évènements du simulateur.

IV-7 Les actions de traitement des connexions

Elles analysent les déclarations des connexions simples et construisent la table des blocs évènements connexion à raison d'une entrée par connexion déclarée. Une vérification est faite pour déterminer :

1) si toutes les unités et les signaux sont définis
2) les compatibilités des connexions (sens des signaux et type des données).

IV-8 Les actions de génération de code PASCAL

Les rôles de ces actions sont présentés aux paragraphes II-5 et III-2.

IV-9 Les actions de traitement des mesures

Elles analysent les expressions des mesures et construisent les tables des compteurs et des évènements mesures.

V- ANNEXE DU CHAPITRE QUATRE

La grammaire formatée du langage LASSO

Les conventions suivantes sont utilisées :

. Les identificateurs entre ' ' sont les symboles terminaux de la grammaire

. Les identificateurs entre < > sont les identificateurs des fonctions sémantiques

. ( ) signifie une alternative vide

. Les autres identificateurs sont des non terminaux de la grammaire.

INPUT SYNTAX=

358	axiome	<initcompil> <liresta> <lirebib> <debalgo> prog 'eof' <findescr>
359	prog	descr 'ptvirg' suite+prog
360	suite+prog	termentite termentite prog pmesure termentite pmesure termentite prog
361	termentite	<ecrend> <ecrend> <ecrptvirg> <majtabentite> <finentite>
362	descr	entete 'ptvirg' interf pgcorps 'fin'
363	entete	'entite' <initentite> x+entete
364	x+entete	<dejaentite> <stortabentite> <entexterne> <casentite> 'id' opt+lstdonparm
365	interf	() 'interface' lstdeclinf
366	lstdeclinf	<synerr1> declinf 'ptvirg' <moderr1> <synerr1> declinf 'ptvirg' <moderr1> lstdeclinf
367	declinf	type lstdonnee declentree declsortie
368	lstdonnee	donnee 'virg' lstdonnee donnee
369	declentree	'entree' <typentree> lstsiginter
370	declsortie	'sortie' <typesortie> lstsiginter
371	lstsiginter	siginter 'virg' lstsiginter siginter
372	siginter	<dejalu> <rangtbident> 'id' suite+siginter
373	suite+siginter	() 'parouv' lstdata 'parferm'

		groupaireborne <deplcalcule> groupaireborne <deplcalcule> 'parouv' lstdata 'parferm'
374	type	'entier' <typentier> 'caract' <typecaract> 'bool' <typebool>
375	donnee	<dejalu> <rangtbident> <azero> 'id' <aun> <dejalu> <rangtbident> <azero> 'id' groupaireborne <deplcalcule>
376	groupaireborne	'crochouv' <typetablo> lstpaireborne 'croferm' <tablomaj>
377	lstpaireborne	x+paireborne 'virg' lstpaireborne x+paireborne
378	x+paireborne	paireborne <nbdimension> <tnbelem> <testbornes> <trangebornes>
379	paireborne	entiersansigne <dimun> 'deuxpt' entiersansigne <dimdeux>
380	lstdata	data 'virg' lstdata data
381	data	<dejalu> <chainedonnee> 'id'
382	pgcorps	'corps' <resetinterf> opt+suitecorps
383	opt+suitecorps	() x+corps 'ptvirg' opt+suitecorps
384	x+corps	decl connexion transition
385	decl	declentite declunite declvar declprocfonc

```
386     declentite          'entite' entetentite corpsentite
                            'entite' 'externe' lexentete

387     entetentite         <dejaentite> <stortabentite> <internentite> 'id' opt+lstdonparm 'ptvirg'

388     opt+lstdonparm      ()
                            'parouv' lstdon 'parferm'

389     corpsentite         'interface' 'fin'

390     lstdon              type ldon 'ptvirg' lstdon
                            type ldon

391     ldon                <typparam> <chaineparam> 'id' 'virg' ldon
                            <typparam> <chaineparam> 'id'

392     lexentete           <dejaentite> <entexiste> 'id' 'virg' lexentete
                            <dejaentite> <entexiste> 'id'

393     declunite           'unite' <lstunite> lstidfunite 'deuxpt' nomentite

394     lstidfunite         <testunite> <nunite> <chaineunite> 'id' 'virg' lstidfunite
                            <testunite> <nunite> <chaineunite> 'id'

395     nomentite           <dejaentite> <rventite> 'id'
                            <dejaentite> <rventite> 'id' 'parouv' <posparam> listconst <touslesparam> <idemunit>
                                'parferm'

396     listconst           tpentier <testype> 'deuxpt' lconstent opt+listconst
                            tpcaract <testype> 'deuxpt' lconscar opt+listconst
                            tpbool <testype> 'deuxpt' lconsbool opt+listconst

397     opt+listconst       ()
                            'ptvirg' listconst
```

```
398     lconstent           entiersansigne <rval> 'virg' lconstent
                            entiersansigne <rval>

399     lconscar            lcaractere <rval> 'virg' lconscar
                            lcaractere <rval>

400     lcaractere          'id'

401     lconsbool           cstebool <rval> 'virg' lconsbool
                            cstebool <rval>

402     declvar             tpentier lstvar1
                            tpcaract lstvar2
                            tpbool lstvar3
                            tpsig lstvar4
                            tpnomsig lstvar5

403     lstvar1             variable 'virg' lstvar1
                            variable
                            variable <debindice> initent <longindexrange> <ideminit> <intrue> 'virg' lstvar1
                            variable <debindice> initent <longindexrange> <ideminit> <intrue>

404     lstvar2             variable 'virg' lstvar2
                            variable
                            variable <debutindice> initcar <rangeinit> <ideminit> <intrue> 'virg' lstvar2
                            variable <debutindice> initcar <rangeinit> <ideminit> <intrue>

405     lstvar3             variable 'virg' lstvar3
                            variable
                            variable <debindice> initbool <longindexrange> <ideminit> <intrue> 'virg' lstvar3
                            variable <debindice> initbool <longindexrange> <ideminit> <intrue>

406     lstvar4             variable 'virg' lstvar4
                            variable
                            variable <debindice> initsig <longindexrange> <ideminit> <intrue> 'virg' lstvar4
                            variable <debindice> initsig <longindexrange> <ideminit> <intrue>
```

407	lstvar5	variable 'virg' lstvar5 variable variable <debutindice> initnomsig <rangeinit> <ideminit> <intrue> 'virg' lstvar5 variable <debutindice> initnomsig <rangeinit> <ideminit> <intrue>
408	initent	'init' 'parouv' lesentiers 'parferm' 'init' lentier
409	lentier	entiersansigne <overinit> <irange>
410	lesentiers	lentier+opt 'virg' lesentiers lentier+opt
411	lentier+opt	<intvidentier> lentier
412	initcar	'init' 'parouv' lescars 'parferm'
413	lescars	lecar+opt 'virg' lescars lecar+opt
414	lecar+opt	<intvdecar> lecar
415	lecar	<uncar> <overinit> <chrange> 'id'
416	initbool	'init' 'parouv' lesbools 'parferm' 'init' lebool
417	lebool	boolcste <overinit> <irange>
418	boolcste	<chiffreun> 'c0' <chiffreun> 'c1'
419	lesbools	lebool+opt 'virg' lesbools lebool+opt
420	lebool+opt	<intvidentier> lebool

```
421     initsig                 initbool

422     initnomsig              'init' 'parouv' lesnomsigs 'parferm'
                                'init' lenomsig

423     lenomsig                <overinit> <chrange> 'id'

424     lesnomsigs              lenomsig+opt 'virg' lesnomsigs
                                lenomsig+opt

425     lenomsig+opt            <intvidnomsig>
                                lenomsig

426     tpentier                'entier' <typentier>

427     tpcaract                'caract' <typecaract>

428     tpbool                  'bool' <typebool>

429     tpsig                   'signal' <typeinterne>

430     tpnomsig                'nomsig' <typnsig>

431     declprocfonc            'procedure' 'externe' <typrocext> foncproc 'parouv' procparam 'parferm'
                                'fonction' 'externe' <typfctext> foncproc 'parouv' foncparam 'parferm' 'resultat' decresultat

432     decresultat             tpentier 'deuxpt' varesul
                                tpcaract 'deuxpt' varesul
                                tpbool 'deuxpt' varesul
                                'deuxpt' tpentier <rangtypresultat>
                                'deuxpt' tpcaract <rangtypresultat>

                                'deuxpt' tpbool <rangtypresultat>

433     varesul                 'id'
```

434	foncproc	<dejalu> <rangtbident> 'idfoncproc'
435	procparam	<pwtypedec> lpar1 'ptvirg' procparam <pwtypedec> lpar1
436	lpar1	lpar tpvar
437	tpvar	'var' 'deuxpt' <typevar> lpar
438	foncparam	lpar 'ptvirg' foncparam lpar
439	lpar	tpentier lparvar tpcaract lparvar tpbool lparvar
440	lparvar	idparam 'virg' lparvar idparam
441	idparam	<lstparamformel> 'id'
442	variable	<dejalu> <rangtbident> <azero> 'id' groupaireborne <deplcalcule> <dejalu> <rangtbident> <azero> 'id' <aun>
443	connexion	connexsimple
444	connexsimple	partiegauche 'flechegauche' partiedroite opt+expretard
445	partiegauche	idfunite 'souligne' idfsigentree idfsiglocalent opt+listvarparam <pdonasrec>
446	partiedroite	uniteidf 'souligne' idfsigsortie idfsiglocalsort opt+listvarparam <pdonasem>

```
447     opt+listvarparam        ()
                                'parouv' lstdata 'parferm'

448     opt+expretard           ()
                                'retard' 'parouv' exprentiere <rangeretard> 'parferm'

449     exprentiere             entiersansigne

450     entiersansigne          <chiffreun> chiffre suitentier
                                <chiffreun> chiffre

451     suitentier              <chiffresuivant> chiffre suitentier
                                <chiffresuivant> chiffre

452     chiffre                 'c0'
                                'c1'
                                'c2'

                                'c3'
                                'c4'
                                'c5'
                                'c6'
                                'c7'
                                'c8'
                                'c9'

453     idfunite                <testunite> <creebloconnex> 'idspecial'

454     uniteidf                <testunite> <suitebloconnex> 'idspecial'

455     idfsigentree            <idcherche> <signalent> 'id'

456     idfsigsortie            <idcherche> <signalsortie> 'id'

457     idfsiglocalsort         <dejalu> <lstblocentree> <poseindexsig> <verifdeclare> <positionerrindex> <suitebloconnex>
                                    <depsigemetteur> 'idfoncproc'

458     idfsiglocalent          <dejalu> <poseindexsig> <verifdeclare> <positionerrindex> <creebloconnex> <depsigrecept>
                                    'idfoncproc'
```

```
459     variab              'id'

460     transition          'question' <initrans> <tetedelistenil> condentree 'question' action
                            'question' <initrans> <tetedelistenil> condentree 'question' opt+delet primcontrol

461     condentree          <dejalu> <rangesignal> <lstsignaux> <lstblocentree> idsignal 'et' condentree
                            <dejalu> <rangesignal> <lstsignaux> <lstblocentree> idsignal

462     idsignal            'id' <tablocomplet>
                            'id' 'crochouv' <debtab> ref+table <vertoutdim> 'croferm' <tablstsignaux>

463     ref+table           lbornes 'virg' ref+table
                            lbornes

464     lbornes             entiersansigne <verdimun> 'deuxpt' entiersansigne <verdimdeux>
                            entiersansigne <verdimun> <verdimdeux>

465     valider+lst+sig     'valider' 'parouv' lstsig 'parferm' <lstsortietrans>
                            <nilsavgenerale> <lstsortietrans>

466     action              'debut' <blocaction> <lstcondentree> <ecrcas> liste+instructions 'fin' <ecrend> <ecrptvirg>
                               opt+delet <tetedelistenil> valider+lst+sig

467     primcontrol         <blocet> <lstcondentree> <tetedelistenil> valider+lst+sig
                            'selon' <blocselon> <lstcondentree> <tetedelistenil> <dejalu> <rangebool> exprbool 'valider'
                               'parouv' <dejalu> <rangesignal> <lstsignaux> <sortieoui> <tetedelistenil> idsignal 'virg'
                               <dejalu> <rangesignal> <lstsignaux> <sortienon> idsignal 'parferm'
                            'choix' <blochoix> <lstcondentree> <tetedelistenil> 'parouv' lstsigchoix <lstestchoix>
                               <tetedelistenil> 'parferm' valider+lst+sig
                            'choixp' <blochoixp> <lstcondentree> <tetedelistenil> 'parouv' lstsigchoix <lstestchoix>
                               <tetedelistenil> 'parferm' valider+lst+sig
                            'index' <blocindex> <lstcondentree> <tetedelistenil> <dejalu> <rangevarindex> variab 'de'
                               exprentiere <rangeborneinf> 'ja' exprentiere <bornesup> valider+lst+sig <rangenbelem>
                            'index' <blocindex> <lstcondentree> <tetedelistenil> <dejalu> <rangevarindex> variab

                               valider+lst+sig <rangenbsignaux>

468     lstsig              <dejalu> <rangesignal> <lstsignaux> <lstblocsortie> idsignal 'virg' lstsig
                            <dejalu> <rangesignal> <lstsignaux> <lstblocsortie> idsignal
                            <pdummy> <lstsignaux> 'virg' lstsig

469     lstsigchoix         <dejalu> <rangesignal> <lstsignaux> <lstblocentree> idsignal 'virg' lstsigchoix
```

```
                        <dejalu> <rangesignal> <lstsignaux> <lstblocentree> idsignal

470     opt+delet       ()
                        'delai' 'parouv' exprentiere <rangedelai> 'parferm'

471     exprbool        'id'

472     cstebool        'c0'
                        'c1'

473     liste+instructions  instructions <depilerd> <ecrptvirg> 'ptvirg' liste+instructions
                        instructions

474     instructions    ()
                        affectation
                        instruction+conditionnel
                        instruction+iteration
                        instruction+tantque
                        'debut' <ecrbegin> liste+instructions 'fin' <ecrend>

475     affectation     idfgauche 'dptegal' <empildptegal> expr+arithm <testcomp>
                        appel+de+procedure

476     appel+de+procedure  <dejalu> <existe> <testproc> 'idfoncproc' 'parouv' <procouv> <pointeparam>
                            liste+de+parametres+effe 'parferm' <pareffectif> <procferm>

477     idf+param+effect    entiersansigne <ptypvar> <tstyparam> <tstvarparam> <ecrentier>
                        <dejalu> <existe> <tstyparam> <transforme> 'id' <ferme>

478     instruction+conditionnel  'si' <ecrsi> expr+booleenne 'alors' <ecrthen> <ecrbegin> liste+instructions opt+sinon 'finsi'
                            <ecrend>

479     opt+sinon       ()
                        'sinon' <ecrend> <ecrelse> <ecrbegin> liste+instructions

480     instruction+iteration  iter idfiter <deuxptegal> 'de' expr+arithm pas+entier 'ja' <ecrto> expr+arithm 'faire'
                            <ecrdo> instructions
```

```
481     pas+entier              'pas' entier+iter
                                ()

482     iter                    'pour'
                                'pourtout'

483     idfiter                 <dejalu> <existe> <tstentier> <ecrloop> <transforme> <ferme> <ecrptvirg> <ecrfor> 'id'

484     entier+iter             entiersansigne <ecrentier>

485     instruction+tantque     'tantque' <ecrwhile> expr+booleenne 'faire' <ecrdo> instructions

                                'repeter' <ecrrepeat> liste+instructions 'jusqua' <ecruntil> expr+booleenne

486     priorite+3              'plus' <empilplus>
                                'moins' <empilmoins>

487     priorite+4              'mult' <empilmult>
                                'div' <empildiv>
                                'mod' <empilmod>

488     priorite+5              'plus' <empilplus>
                                'moins' <empilmoins>
                                'non' <empilnot>
                                'abs' <empilabs>

489     priorite+2              'sup' <empilsup>
                                'inf' <empilinf>
                                'egal' <empilegal>
                                'infegal' <empiliegal>
                                'supegal' <empilsegal>
                                'different' <empildiff>

490     priorite+0              'ou' <empilou>

491     priorite+1              'et' <empilet>

492     expr+0                  expr+1 fin+expr+0
```

493	fin+expr+0	() priorite+0 expr+1 <testcomp> fin+expr+0
494	expr+1	expr+2 fin+expr+1
495	fin+expr+1	() priorite+1 expr+2 <testcomp> fin+expr+1
496	expr+2	expr+3 fin+expr+2
497	fin+expr+2	() priorite+2 expr+3 <testcomp> fin+expr+2
498	expr+3	expr+4 fin+expr+3
499	fin+expr+3	() priorite+3 expr+4 <testcomp> fin+expr+3
500	expr+4	expr+5 fin+expr+4
501	fin+expr+4	() priorite+4 expr+5 <testcomp> fin+expr+4
502	expr+5	expr+base priorite+5 <procouv> expr+5 <procferm> <opunaire>
503	expr+base	idfexpr entiersansigne <ptypvar> <ecrentier> 'parouv' <procouv> expr+0 'parferm' <procferm>
504	expr+booleenne	expr+0 <siexprbool>
505	expr+arithm	expr+0 <siexprentiere>
506	liste+de+parametres+effe	idf+param+effect 'virg' <ecrvirg> liste+de+parametres+effe idf+param+effect

```
507     idfexpr                 idfgauche
                                <dejalu> <existe> <empiletype> <transforme> 'idfoncproc' 'parouv' <procouv>0<toint!pcram>
                        'c%Mte+de+parametres+effe <pareffectif> 'parferm' <procferm>

508     tableau                 'crochouv' <debtab> suitetableau 'croferm' <verfin> <ferme>

509     suitetableau            lesbornes 'virg' <ecrvirg> suitetableau
                                lesbornes

510     lesbornes               entiersansigne <verdimun> <verdimdeux> <gendim>
                                <dejalu> <existe> <tstentier> <empiletype> <transforme> 'id' <ferme>

511     idfgauche               <dejalu> <existe> <empiletype> <transforme> 'id' <reftable> <ferme>
                                <dejalu> <existe> <empiletype> <transforme> 'id' tableau

512     pmesure                 <resetinterf> 'avec' mesure

513     mesure                  <intcompmes> dclcpts blocpm <fincompmes>

514     dclcpts                 'compteur' opt+listedclcpt 'ptvirg'
                                ()

515     opt+listedclcpt         ()
                                listedclcpt

516     listedclcpt             <traitidcpt> <declarecpt> 'id' 'virg' listedclcpt
                                <traitidcpt> <declarecpt> 'id'

517     blocpm                  pm 'ptvirg' blocpm
                                pm
                                ()

518     pm                      'depuis' suitedepuis1
                                'jusqua' suitejusqua1
                                'rep' suiterep1
                                <debsansfen> blocelem2

519     suitedepuis1            entiersansigne <depuisent> suitedepuis2
```

```
                                <initevt> evt <finevt> <depuisevt> suitedepuis3

520     suitedepuis2            'jusqua' suitejusqua2
                                'pendant' entiersansigne <pendent> bloc2 'findepuis' <dependat>
                                bloc2 'findepuis' <depdate>

521     suitedepuis3            'jusqua' suitejusqua3
                                'pendant' entiersansigne <pendent> bloc2 'findepuis' <depenevt>
                                bloc2 'findepuis' <depevt>

522     suitejusqua2            entiersansigne <jusqaent> bloc2 'findepuis' <fenentent>
                                <initevt> evt <finevt> <jusqaevt> bloc2 'findepuis' <fenentevt>

523     suitejusqua3            entiersansigne <jusqaent> bloc2 'findepuis' <fenevtent>
                                <initevt> evt <finevt> <jusqaevt> bloc2 'findepuis' <fenevtevt>

524     suitejusqua1            entiersansigne <jusdate> bloc2 'findepuis' <finfenent>
                                <initevt> evt <finevt> <jusevt> bloc2 'findepuis' <finfenevt>

525     suiterep1               entiersansigne <repete> 'depuis' <initevt> evt <finevt> <depuisevt> suiterep2
                                'depuis' <initevt> evt <finevt> <depuisevt> suiterep3

526     suiterep2               'jusqua' <initevt> evt <finevt> <jusqaevt> bloc3 'finrep' <nfinrejus>
                                'pendant' entiersansigne <pendent> bloc3 'finrep' <nfinrepend>

527     suiterep3               'jusqua' <initevt> evt <finevt> <jusqaevt> bloc3 'finrep' <finrepjus>
                                'pendant' entiersansigne <pendent> bloc3 'finrep' <finreppend>

528     bloc2                   blocelem2 'ptvirg' bloc2
                                blocelem2

529     blocelem2               'desque' suitedesque
                                'lafoisno' entiersansigne <lafoisent> 'que' <initevt> evt <finevt> <quevt> bloc10 'fin'
                                    <typevt2>
                                'chaquefoisque' <initevt> evt <finevt> <chaquevt> bloc10 'fin' <typevt2>
                                'tousles' entiersansigne <tousent> bloc10 'fin' <typdat2>
```

530	suitedesque	entiersansigne <desqent> bloc1Ø 'fin' <typdat2> <initevt> evt <finevt> <desqevt> bloc1Ø 'fin' <typevt2>
531	bloc3	blocelem3 bloc3 blocelem3
532	blocelem3	'desque' <initevt> evt <finevt> <desqevt> bloc1Ø 'fin' <typevt2> 'lafoisno' entiersansigne <replafois> 'que' <initevt> evt <finevt> <quevt> bloc1Ø 'fin' <typevt2> 'chaquefoisque' <initevt> evt <finevt> <chaquevt> bloc1Ø 'fin' <typevt2> 'tousles' entiersansigne <tousent> bloc1Ø 'fin' <typdat2>
533	bloc1Ø	blocelem1Ø 'ptvirg' bloc1Ø blocelem1Ø
534	blocelem1Ø	<ecrsi> 'si' expr+booleenne 'alors' <ecrthen> <ecrbegin> blocelem1Ø suitesi actionelempm ()
535	suitesi	'finsi' <ecrend> <ecrend> 'sinon' <ecrelse> <ecrbegin> blocelem1Ø 'finsi' <ecrend>
536	actionelempm	'inc' 'parouv' opt+listidcpt 'parferm' 'mes' 'parouv' opt+listidvar 'parferm' <relevedate> 'date' 'relever' 'parouv' opt+listidc 'parferm' 'init' 'parouv' opt+listcpt 'parferm'
537	opt+listidcpt	() listidcpt
538	opt+listidvar	() listidvar

```
539     opt+listidc             ()
                                listidc

540     opt+listcpt             ()
                                listcpt

541     listidc                 <traitidcpt> <relevecpt> 'id' 'virg' listidc
                                <traitidcpt> <relevecpt> 'id'

542     listcpt                 <traitidcpt> <initcpt> 'id' 'virg' listcpt
                                <traitidcpt> <initcpt> 'id'

543     listidcpt               <traitidcpt> <increment> 'id' 'virg' listidcpt
                                <traitidcpt> <increment> 'id'

544     listidvar               <dejalu> <mesvar> 'id' 'virg' listidvar
                                <dejalu> <mesvar> 'id'

545     evt                     evt1 suitevt

546     suitevt                 evt2
                                <depilevt>

547     evt2                    'puis' <depilevt> <empilpuis> evt
                                'commesi' <depilevt> <empilcomme> evt
                                'etdeplus' <depilevt> <empildeplus> evt

548     evt1                    'parouv' evt 'parferm' <depilevt>
                                'etoilezero' <dejalu> <listevt0> 'id'
                                'etoileun' <dejalu> <listevt1> 'id'
```

CONCLUSION ET BIBLIOGRAPHIE

CONCLUSION

Le présent mémoire a exposé la mise en oeuvre du langage LASSO au moyen d'un compilateur et d'un simulateur prototypes.

Le concepteur de circuits dispose actuellement d'un outil de description modulaire, offrant dans la structure du langage un moyen naturel de séparation de la partie contrôle de la partie opérative d'un circuit.

Les simulations interactives permettent d'agir à tout instant sur le fonctionnement interne du modèle.

Une partie mesure associée à une entité permet de réaliser automatiquement les prises de mesures et/ou l'espionnage d'un modèle en cours de fonctionnement. Ce dernier point est une caractéristique importante de LASSO; il est à notre avis un facteur important dans le gain de temps de conception d'un circuit.

Le prototype actuel a été testé sur des exemples de descriptions réels [29]. Il est cependant difficile d'estimer les performances du simulateur car les simulations effectuées n'ont jamais excédé deux minutes. Les descriptions ont cependant montré la nécessité d'apporter certaines améliorations et extensions à différents niveaux du système LASSO :

- au niveau du langage de descriptions

L'ensemble des primitives de synchronisation présentées n'est pas figé; ces primitives sont le fruit de notre propre expérience.

L'étude menée par D. DUBOIS [29] sur l'Application de LASSO à la Description Formelle de Circuits Intégrés a montré que de nouvelles primitives étaient nécessaires comme par exemple une primitive choicx, similaire à choixp (cf chapître 1) dans laquelle tous les signaux en entrée sont remis à 0 après la prise en compte du plus prioritaire; la présence de cette nouvelle primitive rendrait beaucoup plus aisée la description d'un système de traitement d'interruptions particulier.

- au niveau du compilateur

Dans le but d'améliorer les messages d'erreurs envoyés au concepteur lors du traitement de sa description, une attention toute particulière doit être portée au système de détection et de correction des erreurs syntaxiques et sémantiques. Nous pensons pour cela dans un premier temps exploiter toutes les possibilités potentielles offertes par le transformateur de grammaire actuel. Dans un deuxième temps nous pensons appliquer des stratégies plus globales telles que celles proposées dans [13] et [17] qui nécessitent cependant une modification de la structure de données de compilation.

- au niveau du simulateur

Le concepteur dispose pour dialoguer avec le simulateur d'un langage de commandes simple facilement extensible, auquel il est nécessaire d'ajouter la possibilité de définir et de traiter des macros-commandes.

La réalisation de ce système s'inscrit actuellement dans un projet beaucoup plus vaste de construction d'un système intégré d'aide à la conception de circuits logiques développé au sein de notre équipe. Les choix que nous avons pris dans LASSO et les enseignements que nous avons pu tirer de son implantation seront pour nous une aide précieuse dans la mise en oeuvre du système intégré.

BIBLIOGRAPHIE LANGAGE

[1] C.W. ROSE
'LOGOS and the software engineer'
Conference Proceedings, 1972 Fall Joint Computer Conference,
Vol. 41, Part 1, AFIPS.

[2] J. MERMET
'Etude méthodologique de la conception assistée par ordinateur des systèmes logiques : CASSANDRE'
Thèse Docteur ès-Sciences Mathématiques IMAG, Avril 1973.

[3] K. JENSEN, N. WIRTH
'PASCAL - User manual and report'
Springer Verlag, 1974.

[4] F.J. RAMMIG
'DIGITESTII : integrated structural and behavioral language'
International symposium on Computer Hardware Description Languages and their applications, New York, September 1975.

[5] SERVICE DE SYNTHESE ET D'ORIENTATION DE LA RECHERCHE EN INFORMATIQUE
'SFER-PASCAL - Le langage de programmation Pascal. Compilateur pour les ordinateurs CII 10070, IRIS 80. Système d'exploitation SIRIS 7-8'
Sept. 75.

[6] D. BORRIONE
'LASCAR : un langage pour la simulation et l'évaluation des architectures d'ordinateurs'
Thèse 3ème cycle spécialité informatique IMAG, Avril 1976.

[7] R. PERRET
'LASICO : langage de simulation et de commande'
Bulletin BIGRE n° 11, Novembre 1978.

[8] J. MERMET
'Définition d'un langage préfixe pour exprimer les conditions logiques et les évènements'
Rapport de Recherche n° 143 IMAG, Grenoble, Décembre 1978.

BIBLIOGRAPHIE COMPILATEUR

[9] M.K. DONEGAN, R.E. NOONAN, S. FEYOCK
'A code generator Generator language'
ACM SIGPLAN NOTICES, Vol. 14, Number 8, August 1979.

[10] R.G.G. CATTELL, J.M. NEW COMER, B.W. LEVERETT
'A code generation in a machine independent compiler'
ACM SIGPLAN NOTICES, Vol. 14, Number 8, August 1979.

[11] R.G.G. CATTELL
'Formalization and automatic derivation of Code Generators'
PHD Thesis, Carnegie Mellon University, April 1978.

[12] D.F. MILTON, L.W. KIRCHHOFF, B.R. ROWLAND
'An ALL(1) compiler Generator'
ACM SIGPLAN NOTICES, Vol. 14, Number 8, August 1979.

[13] A.B. PAI, R.B. KIEBURTZ
'Global context recovery : a new strategy for parser recovery from syntax errors'
ACM SIGPLAN NOTICES, Vol. 14, Number 8, August 1979.

[14] T.N. TURBA
'General syntax analyzer (GSA)'
SPERRY-UNIVAC
ACM SIGPLAN NOTICES, Vol. 14, Number 12, December 1979.

[15] P. KORNERUP, B.B. KRISTENSEN, OLE.LEHRMANN MADSEN
'Interpretation and code generation based on intermediate languages'
COMPUTER SCIENCE DEPARTMENT AARHUS UNIVERSITY DENMARK
DAIMI internal report PB-88, May 1978.

[16] M. GRIFFITHS
'Analyse déterministe et compilateurs'
Thèse d'Etat présentée à Grenoble, Octobre 1969.

[17] J.P. BANATRE, J.P. ROUTEAU, L. TRILLING
'An event-driven compiling technique'
CACM, Vol. 22, Number 1, January 1979.

[18] M. DELAUNAY, J.F. GRABOWIECKI
'Utilisation du transformateur de grammaire LL(1)
Interprétation de l'analyseur syntaxique
(système MULTICS) (production et optimisation des tables)'
Rapport de recherche IMAG n° 234, Septembre 1980.

[19] D. LEBARBIER
'Etude et réalisation d'un producteur de code paramétrable par les descriptions des langages source et cible'
Thèse 3ème cycle présentée à l'Université de Rennes I, Octobre 1979.

BIBLIOGRAPHIE COMPILATEUR

[20] P. WARD
'Un système d'écriture de compilateurs à analyse syntaxique descendante'
Manuel d'utilisation
Faculté des Arts et Sciences, Université de Montréal, octobre 1975.

[21] P. WARD
'Un système d'écriture de compilateurs à analyse syntaxique descendante'
Notes d'installation et de fonctionnement
Faculté des Arts et Sciences, Université de Montréal, octobre 1975.

[22] J. FELDMAN, D. GRIES
'Translator Writing Systems'
CADM, Vol. 11, number 2, February 1968.

BIBLIOGRAPHIE SIMULATEUR

[23] E.G. ULRICH
'Event manipulation for discrete simulations requiring large numbers of events'
CACM, Vol. 21, number 9, sept. 1978.

[24] J. LEROUDIER
'La simulation à évènements discrets'
Monographies d'Informatique de l'A.F.C.E.T.
Editions Hommes et Techniques, Suresnes, 1980, pp. 101:

BIBLIOGRAPHIE LASSO

[25] D. BORRIONE, J.F. GRABOWIECKI
'Etude d'un système intégré de simulation pour l'aide à la conception des systèmes logiques'
Rapport final, Contrat SESORI n° 75151, février 1978.

[26] D. BORRIONE, J.F. GRABOWIECKI
'Définition d'un langage pour la simulation des systèmes informatiques : LASSO'
Rapport de recherche n° 122, I.M.A.G., juin 1978.

[27] J.L. BAILLY, J.J. LEROUX
'Définition et implémentation d'un langage de prise de mesures automatiques sur le langage LASSO'
Projet 3ème année E.N.S.I.M.A.G., Grenoble, juin 1979.

[28] D. BORRIONE, J.F. GRABOWIECKI
'Informal introduction to LASSO : a language for asynchronous system specification and simulation'
Présenté à l'Euro IFIP 79 à Londres, septembre 1979.

[29] D. DUBOIS
'Evaluation du langage LASSO pour la description formelle de circuits intégrés'
Rapport de recherche I.M.A.G. n° 180, décembre 1979.

[30] D. BORRIONE, J.F. GRABOWIECKI
'Réalisation d'un prototype du compilateur et du simulateur du langage LASSO'
Rapport final contrat SESORI 78067, août 1980
TOME 1 : "LASSO : Le compilateur prototype"
TOME 2 : "LASSO : Le simulateur prototype"
TOME 3 : "LASSO : Le manuel d'utilisation".

[31] D. BORRIONE, J.F. GRABOWIECKI, J.C. REYNAUD
'Etude d'un système intégré de simulation pour l'aide à la conception des systèmes logiques'
Contrat SESORI n° 75151, lot n° 1, avril 1976.

BIBLIOGRAPHIE GENERALE

[32] COMPAGNIE INTERNATIONALE POUR L'INFORMATIQUE
'Ordinateur IRIS80 manuel d'utilisation'
Février 1973.

[33] S. MARIANI
'Simulation d'une machine multi-processeur'
Rapport D.E.A., Université de Grenoble, juin 1976.

[34] G. MAZARE
'Structure multi-processeurs - Problèmes de parallélisme, Définition et évaluation d'un système particulier'
Thèse Docteur ès-sciences présentée à Grenoble, juin 1978.

[35] J.F. GRABOWIECKI
'Simulation et évaluation des caches sur une structure multi-processeurs multi-caches'
(Paragraphe A.4 du rapport de Synthèse Finale. Contrat D.R.M.E. n° 75101 'Spécification et évaluation d'un système multimicro-processeurs'), juin 1977.