# Manuel SIGNAL

Patricia Bournai, Bruno Chéron, Bernard Houssais, Paul Le Guernic

INRIA

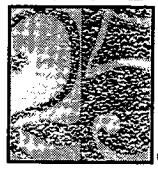
UNITÉ DE RECHERCHE
INRIA RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports Techniques

*1 9 9 2*



*ème anniversaire*

**N° 128**

***Programme 1***

*Architectures Parallèles, Bases de Données,*
*Réseaux et Systèmes*

# MANUEL SIGNAL

**Patricia BOURNAI**
**Bruno CHERON**
**Bernard HOUSSAIS**
**Paul LE GUERNIC**

**Avril 1991**

IRISA INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX
FRANCE
Téléphone : 99.36.20.00
Télex : UNIRISA 950 473F
Télécopie : 99.38.38.32

# MANUEL SIGNAL

I.R.I.S.A.

Campus de Beaulieu

35042 Rennes CEDEX

Patricia BOURNAI, Bruno CHERON, Bernard HOUSSAIS,Paul LE GUERNIC

4 février 1991

Publication Interne n° 575 - Février 1991 - 84 pages

Programme 1

**Résumé**

SIGNAL est un langage conçu pour la programmation synchrone de systèmes temps réel. Un programme SIGNAL définit un automate à partir d'un système d'équations dont les variables sont les signaux. Un signal est une suite de valeurs à laquelle est associée une horloge qui spécifie les instants auxquels les valeurs sont disponibles. Un programme SIGNAL exprime ainsi les relations fonctionnelles et temporelles qui existent entre tous les signaux mis en oeuvre.

Ce rapport est un manuel du **langage** SIGNAL; il présente la syntaxe du langage dans sa version H2, et introduit sa sémantique. Des exemples sont présentés tout au long du rapport. L'annexe A montre un exemple complet d'utilisation du compilateur.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
( U.R.A. 227 )
UNIVERSITE DE RENNES I I.N.S.A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
( UNITE DE RECHERCHE DE RENNES )

# SIGNAL MANUAL

**Abstract**

SIGNAL is a synchronous language designed to program real-time systems. A SIGNAL program defines an automaton whose variables are signals. A signal is a sequence of values to which a clock is associated. This clock defines the instants at which these values are available. A SIGNAL program expresses the functional and temporal relationships between all the involved signals.

This paper is a SIGNAL langage manual; it presents the language syntax, in its H2 version, and introduces its semantic. Examples are presented in the different chapters. The appendix A is a complet example of the compiler use.

# Table des matières

# Chapitre I

# Introduction

Le **langage** SIGNAL a été défini à l'INRIA/IRISA avec la collaboration et le soutien du CNET [3] [2]. Ce document contient une description sommaire des caractéristiques de la version H2 du langage. Il ne constitue ni un manuel de référence, ni un manuel de l'utilisateur.

## I-1 Principaux traits du langage

Un programme exprimé dans le **langage** SIGNAL définit un automate à partir d'un système d'équations dont les variables sont les identificateurs de signaux. Ces équations peuvent être organisées en sous-systèmes (ou processus). Un modèle de processus est un sous-système qui peut avoir plusieurs contextes d'utilisation. Pour ce faire un modèle est désigné par un identificateur.

### I-1.1 Signaux

**Un signal est une suite de valeurs, à laquelle est associée une horloge.** Les valeurs du signal appartiennent toutes à un même domaine prédéfini (Signaux purs, Booléens, Entiers, Réels, Complexes) ou défini dans le programme (Tableaux). L'horloge permet de définir l'ensemble des instants où un signal possède une valeur.

### I-1.2 Evénements

Une communication associe, à un instant logique du programme, une valeur à une variable.
Un événement est un ensemble de communications simultanées formant une transition de l'automate. Dans un événement, une variable peut ne pas avoir de valeur associée : on dira alors que le signal est absent et on notera alors $\perp$ sa valeur. Un événement comporte au moins une communication.

La détermination de la présence d'un signal dans un événement résulte de la résolution d'un système d'équations dans $\mathcal{F}_3$ le corps des entiers modulo 3 [1].

La valeur associée à une variable dans un événement est le résultat de l'évaluation de son expression de définition (qui ne doit donc pas être implicite : les définitions circulaires de signaux non booléens sont interdites).

### I-1.3 Modèles

Un modèle associe à un identificateur un système d'équations muni de variables locales, de sous-modèles et de variables externes (variables libres). Les paramètres d'un modèle sont des

constantes de taille de tableaux ou de valeurs initiales de signaux.
Un modèle peut être défini à l'extérieur du programme ; il n'est alors visible que par son interface. L'invocation d'un modèle défini dans un programme est équivalente au remplacement de cette invocation par le système d'équations associé (macro-substitution).

### I-1.4 Causalité

Un programme temps-réel doit respecter le principe de causalité, la valeur d'un événement ne peut dépendre de la valeur d'un événement futur. La garantie du respect de ce principe est obtenue dans le **langage** SIGNAL par la gestion implicite du temps : l'utilisateur dispose d'un ensemble de termes qui lui permettent de faire référence aux valeurs passées ou actuelles d'un signal, pas à une valeur future.

## I-2 Forme de présentation

Ce document présente la syntaxe, et introduit la sémantique du **langage** SIGNAL. On distingue deux classes de termes pour la description de la syntaxe du langage :

- les structures lexicales (ou terminaux) définies, à un niveau lexical, par des règles dans une grammaire dont le vocabulaire est un ensemble de caractères ; aucun caractère implicite (par exemple, séparateur) n'est autorisé dans les termes construits selon ces règles ;
- les stuctures syntaxiques définies, à un niveau syntaxique, par des règles dans une grammaire dont le vocabulaire est formé des terminaux ; entre deux terminaux, on peut toujours insérer des séparateurs.

Chaque unité du langage est introduite puis décrite, individuellement ou par catégorie, à l'aide de tout ou partie des alinéas suivants.

1. **Syntaxe Hors Contexte.** Elle donne la syntaxe hors-contexte de la structure considérée selon l'une des formes suivantes :

   ♠ **STRUCTURE** ::=
   - DERIVATION1 •
   - DERIVATION2 •
   - ... •

   ♠ **Terminal** ::=
   - DERIVATION1 •
   - DERIVATION2 •
   - ... •

   DERIVATION1, DERIVATION2 sont des réécritures de la variable **STRUCTURE** (respectivement, de la variable **Terminal**).
   Chaque DERIVATION est une suite d'*éléments* dont chacun peut être :

   - un **ensemble** de caractères, noté dans cette typographie (niveau lexical uniquement),
   - un symbole **terminal** (formé de lettres), dans cette typographie,
   - un **symbole** terminal (formé d'autres caractères admissibles), dans cette typographie,
   - un **Terminal**, dans cette typographie,

- une **STRUCTURE** syntaxique, dans cette typographie (niveau syntaxique uniquement),
- une suite d'*éléments* avec ou sans séparateur, respectivement sous les formes suivantes :
  - { *élément* **symbole** ... } ,
  - { *élément* ... }
- un *élément* optionnel, noté [ *élément* ] .

2. **Représentant.**
   ▷ EXPRESSION($E_1$,$E_2$,...) ◁
   est un terme générique dont les arguments formels sont dénotés par $E_1$, $E_2$, ... ; ce représentant est utilisé pour définir les propriétés générales d'un terme du **langage** SIGNAL dans les rubriques suivantes.

3. **Définition en SIGNAL.**
   TERME($E_1$,$E_2$,...)
   ▷ est défini par un terme du **langage** SIGNAL contenu dans ce paragraphe. ◁

4. **Profil.**
   Cette rubrique décrit l'ensemble des signaux d'entrée et de sortie de l'expression ; cette description est obtenue avec respectivement les notations $?(E)$ pour désigner la liste des signaux d'entrée de $E$, $!(E)$ pour désigner la liste des signaux de sortie de $E$ et enfin $!\{a_1,\ldots,a_n\}$ pour désigner l'ensemble explicite des ports $a_1,\ldots,a_n$. On utilisera les opérations ensemblistes $A \bigcap B$, $A \bigcup B$ et finalement la notation $A - B$ pour désigner l'ensemble des éléments de $A$ qui ne sont pas dans $B$.

5. **Types.**
   Cette rubrique décrit les propriétés des types des arguments sous la forme d'équations sur les domaines de valeur des signaux. La notation $\tau(E)$ est utilisée pour désigner le type de l'expression $E$

   (a) EQUATION1

6. **Horloges.**
   Cette rubrique décrit les propriétés de synchronisation des arguments (valeurs de booléens et horloges), informellement ou au moyen d'une liste d'équations dans l'espace de synchronisation. La notation $\omega(E)$ y est utilisée pour désigner l'horloge de l'expression $E$ et la notation $\hbar$ pour désigner l'horloge des expressions constantes. Une équation a la forme suivante :

   (a) $E_1 = E_2$

7. **Sémantique.**
   Lorsque le terme ne peut être redéfini dans le **langage** SIGNAL, sa sémantique est l'objet d'une présentation informelle.

8. **Propriétés.**
   On donne ici une liste de propriétés de la construction (telles que associativité, distributivité, etc.).

   (a) PROPRIETE

9. **Exemples.**

   (a) Un ou plusieurs **Exemples** dans le **langage** SIGNAL complètent la présentation de l'unité.

## I-3 Unités lexicales

Le texte d'un programme du **langage** SIGNAL est formé de mots du vocabulaire construits sur un ensemble de caractères.

### I-3.1 Caractères

L'ensemble des caractères (noté **caractère**) utilisés dans le **langage** SIGNAL est l'ensemble des caractères ASCII parmi lesquels on distingue les sous-ensembles disjoints suivants :

1. L'ensemble **carnom** des caractères utilisés pour construire les identificateurs ; cet ensemble est lui même constitué de
   - l'ensemble **lettre** des lettres de l'alphabet, lui-même union des sous-ensembles disjoints suivants :
     - l'ensemble **majuscule** des lettres majuscules de l'alphabet :
       A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
     - l'ensemble **minuscule** des lettres minuscules de l'alphabet :
       a b c d e f g h i j k l m n o p q r s t u v w x y z

     Les formes majuscule et minuscule d'une même lettre ne sont distinguées que pour les mots-cles ; ceux-ci sont nécessairement écrits en **minuscule**.
   - l'ensemble **chiffre** des chiffres :
     0 1 2 3 4 5 6 7 8 9
2. Le séparateur de suffixe ⊡
3. Les **séparateurs** : ␣ (caractère espace), fin de ligne,
4. Les caractères spéciaux utilisés dans les terminaux du langage ;
5. Les autres caractères éditables utilisables dans les commentaires.

### I-3.2 Vocabulaire

Un texte du **langage** SIGNAL est une suite d'éléments du vocabulaire **Terminal** du **langage** SIGNAL, entre lesquels peut apparaître un nombre quelconque de **séparateurs**. Un **Terminal** du **langage** SIGNAL est la plus longue suite de caractères contigus qui peut être formée selon les règles décrites ci-dessous.

#### I-3.2 A Noms

Un nom permet de désigner un signal, un paramètre ou un modèle, dans un contexte formé d'un enseble de déclarations. Deux occurrences du même nom dans des contextes distincts peuvent désigner des objets différents.

1. **Syntaxe Hors Contexte.** Un nom est formé d'une lettre suivie d'une suite de lettres ou de chiffres séparables par le caractère ⊡, suivi ou non et précédé ou non de **séparateurs**. Tous les caractères **carnom** et les occurrences de ⊡ sont significatifs (à la confusion minuscule-majuscule près).

♠ **Nom** ::=

- **lettre** [ { { **carnom** ... } ⊡ ... } ] •
- **lettre** ⊡ { { **carnom** ... } ⊡ ... } •

2. **Exemples.**

   (a) `a` et `A` sont des **Noms** identiques.

   (b) `X_25`, `Le_mot_de_passe_12Xs3` sont des **Noms**.

### I-3.2 B Constantes

Une dénotation de valeur constante est l'une des unités énumérées ci-dessous et décrites dans le chapitre II :

- **Cst-booléenne**
- **Cst-entière**

### I-3.2 C Commentaires

Un commentaire est une suite de caractères quelconques entre deux occurrences du caractère [%]. Un commentaire apparaît librement entre deux unités syntaxiques (termes lexicaux).

### I-3.3 Mots réservés

L'ensemble des mots réservés est constitué des terminaux apparaissant dans l'index.b sous le titre Terminal

## I-4 Codage des synchronisations

La vérification des propriétés d'un programme du **langage** SIGNAL se fait dans un espace à trois valeurs pour le codage du contrôle associé à un signal booléen qui peut être absent, ou s'il est présent posséder la valeur *false* ou *true*. Le corps des entiers modulo 3 ($\mathcal{F}_3$) est utilisé pour ce codage :

absent → 0
*false* → -1
*true* → 1

# Chapitre II

# Domaines de valeur des signaux

Un signal est une suite de valeurs associée à une horloge. Ces valeurs possèdent toutes le même type qui est, par abus de langage, considéré comme le type de la suite. L'objet de ce chapitre est de présenter les notations adoptées pour représenter ces types et les traitements qui leur sont appliqués. Un élément de l'ensemble des types du **langage** SIGNAL est noté *type*.

Soit E un terme du **langage** SIGNAL ; on note $\tau$(E) le type associé au terme E et, lorsque E est une expression constante, $\varphi$(E) la valeur de cette expression, élaborés dans le contexte où apparaît E.

## II-1 Types scalaires

Les types scalaires comprennent les types de synchronisation, le type entier, les types réels et le type complexe ; ces trois derniers forment l'ensemble des types numériques selon la syntaxe suivante :

1. **Syntaxe Hors Contexte.**

   ♠ **Type-scalaire** ::=

   - **Type-synchronisation** •
   - **Type-numérique** •

   ♠ **Type-numérique** ::=

   - **Type-entier** •
   - **Type-réel** •
   - **Type-complexe** •

### II-1.1 Types de synchronisation

Les types de synchronisation sont utilisés pour construire les horloges des signaux. Ce sont le type *event* (ou signal pur) et le type *logical*.

#### Dénotations de types

1. **Syntaxe Hors Contexte.**

♠ **Type-synchronisation** ::=

- • **event** •
- • **logical** •

2. **Types.**

(a) $\tau$(**event**) = *event*

(b) $\tau$(**logical**) = *logical*

### Dénotations de valeurs

- Un signal de type *event* prend ses valeurs dans un singleton : il n'y a pas de constante associée et un paramètre ne peut être de ce type.
- Les constantes de type *logical* sont les valeurs logiques dénotées selon la syntaxe d'une **Cst-booléenne**.

1. **Syntaxe Hors Contexte.**

♠ **Cst-booléenne** ::=

- • **true** •
- • **false** •

## II-1.2 Type entier

### Dénotations de types

1. **Syntaxe Hors Contexte.**

♠ **Type-entier** ::=

- • **integer** •

2. **Types.**

(a) $\tau$(**integer**) = *integer*

### Dénotations de valeurs

Les valeurs positives de type *integer*sont dénotées selon la syntaxe d'une **Cst-entière**. Une valeur négative n'a pas de représentation directe : elle est obtenue en utilisant l'opérateur **−** appliqué à une valeur positive.

1. **Syntaxe Hors Contexte.** Une **Cst-entière** est formée d'une suite de chiffres.

♠ **Cst-entière** ::=

- • { **chiffre** ... } •

2. **Types.**

(a) Le type d'une **Cst-entière** est le type *integer*.

3. **Sémantique.**

Une **Cst-entière** dénote une valeur entière positive ou nulle représentée en base 10.

## II-1.3 Types réels

Les valeurs réelles peuvent être en représentation simple précision (type *real*) ou double précision (type *long real*). Le générateur de code FORTRAN assimile les type *real* et *long real*.

### Dénotations de types

1. **Syntaxe Hors Contexte.**

   ♠ **Type-réel** ::=

   • [real] •
   • [dpreal] •

2. **Types.**

   (a) $\tau$(`real`) = *real*

   (b) $\tau$(`dpreal`) = *long real*

### Dénotations de valeurs

Une valeur de type *real*est dénotée selon la syntaxe d'une **CST-REELLE** identique à la représentation FORTRAN 77 à l'anomalie suivante près :
La marque d'exposant ([e] ou [d]) doit être impérativement suivie d'un séparateur (espace ou fin de ligne). Une **CST-REELLE** dénote la valeur approchée d'un nombre réel.

1. **Syntaxe Hors Contexte.** Il existe deux ensembles de réels : les réels simple précision et les réels double précision qui contiennent les précédents.

   ♠ **CST-REELLE** ::=

   • **CST-REELLE-SIMPLE-PRECISION** •
   • **CST-REELLE-DOUBLE-PRECISION** •

   ♠ **CST-REELLE-SIMPLE-PRECISION** ::=

   • **Cst-entière EXPOSANT-SIMPLE-PRECISION** •
   • **Partie-fraction [ EXPOSANT-SIMPLE-PRECISION ]** •

   Une **CST-REELLE-SIMPLE-PRECISION peut posséder** un exposant.

   ♠ **CST-REELLE-DOUBLE-PRECISION** ::=

   • **Cst-entière EXPOSANT-DOUBLE-PRECISION** •
   • **Partie-fraction EXPOSANT-DOUBLE-PRECISION** •

   Une **CST-REELLE-DOUBLE-PRECISION doit posséder** un exposant.

   ♠ **Partie-fraction** ::=

   • [.] **Cst-entière** •
   • **Cst-entière** [.] **[ Cst-entière ]** •

   Une **Partie-fraction** est composée d'un point et d'un entier au moins.

   ♠ **EXPOSANT-SIMPLE-PRECISION** ::=

   • [e] **CST-RELATIVE** •

   ♠ **EXPOSANT-DOUBLE-PRECISION** ::=

   • [d] **CST-RELATIVE** •

♠ **CST-RELATIVE** ::=

- **Cst-entière** •
- • [+] **Cst-entière** •
- • [−] **Cst-entière** •

2. **Représentant.**

▷ $E_1.E_2$e $E_3$ (simple précision) ou $E_1.E_2$d $E_3$ (double précision) ◁

3. **Types.**

(a) Une **CST-REELLE-SIMPLE-PRECISION** est de type *real*.

(b) Une **CST-REELLE-DOUBLE-PRECISION** est de type *long real*.

4. **Sémantique.**

La valeur $\varphi(E_i)$, lorsque $E_i$ est absent, est 0. Si $E_2$ possède $n$ chiffres, la valeur de la constante est la valeur approchée de $(\varphi(E_1) + \varphi(E_2) * 10^{-n}) * 10^{\varphi(E_3)}$.

5. **Exemples.**

(a) Les notations contenues dans les tableaux suivants sont des représentations simple précision équivalentes respectivement de la valeur unité et du centième de l'unité.

| | 1e 0 | 1e +0 | 10e -1 |
|---|---|---|---|
| 1. | 1.e 0 | 1.e +0 | 10.e -1 |
| 1.0 | 0.1e 1 | 0.1e +1 | 10.0e -1 |
| | .1e 1 | .1e +1 | |

| | | | 1e -2 |
|---|---|---|---|
| | | | 1.e -2 |
| 0.01 | 0.001e 1 | 0.001e +1 | 1.0e -2 |
| .01 | .001e 1 | .001e +1 | .1e -1 |

## II-1.4 Type complexe

### Dénotations de types

1. **Syntaxe Hors Contexte.**

♠ **Type-complexe** ::=

- • [complex] •

2. **Types.**

(a) $\tau$(complex) = *complex*

### Dénotations de valeurs

Une valeur de type *complex* est dénotée par un couple de réels synchrones dont le premier élément est la partie réelle et le second la partie imaginaire.

1. **Syntaxe Hors Contexte.**

♠ **CST-COMPLEXE** ::=

- • [(] **CST-REELLE-SIGNEE** [,] **CST-REELLE-SIGNEE** [)] •

♠ **CST-REELLE-SIGNEE** ::=

- **CST-RELATIVE** •
- [+] **CST-REELLE-SIMPLE-PRECISION** •
- [−] **CST-REELLE-SIMPLE-PRECISION** •

2. **Exemples.**

(a) (1.0,−1.0)

## II-2 Types tableau

Un tableau est une structure de regroupement d'éléments synchrones de même type. La description de cette structure et de l'accès à ses éléments est obtenue en utilisant des expressions constantes possédant la syntaxe générale des expressions sur signaux (**S-EXPR**).

### Dénotations de types

Un type tableau est défini selon la syntaxe de la première règle donnée ci-dessous ; le type des éléments d'un tableau est décrit par la variable **Type-élément**.

1. **Syntaxe Hors Contexte.**

♠ **TYPE-TABLEAU** ::=

- [[] { **S-EXPR** [,] ... } []] **Type-élément** •

♠ **Type-élément** ::=

- [**logical**] •
- **Type-numérique** •

2. **Représentant.**

$\triangleright\ [n_1, \ldots, n_m]\nu\ \triangleleft$

3. **Types.**

(a) Les valeurs élaborées de $n_1$ ($\varphi(n_1)$), ..., $n_m$ ($\varphi(n_m)$) sont des entiers positifs.

(b) Le type du tableau est :

$$\tau([n_1, \ldots, n_m]\nu) = [1..\varphi(n_1)] \times \ldots \times [1..\varphi(n_m)] \rightarrow \tau(\nu).$$

4. **Horloges.**

Les entiers $n_i$ sont définis par des expressions constantes.

(a) $\omega(n_i) = \hbar$

5. **Exemples.**

(a) `[10,10]integer T` déclare T comme tableau d'entiers à deux dimensions dont les bornes commencent à 1 dans chaque dimension.

### Dénotations de valeurs

Une constante tableau est une énumération d'expressions constantes de même type (cf Expressions sur tableaux).

## II-3 Structure de l'ensemble des types

Un ordre partiel est défini sur les types de telle sorte qu'il existe un plongement "naturel" d'un ensemble plus petit dans un ensemble plus grand.

### II-3.1 Ensemble des types

L'ensemble des types est formé des types dont les expressions dans le **langage** SIGNAL, décrites dans le récapitulatif ci-dessous, dérivent de la variable **TYPE-SIGNAL** :

**TYPE-SIGNAL**
- **TYPE-TABLEAU**
  [ { **S-EXPR** [,] ... } ] **Type-élément**
  Forme générique des types tableau : $[1..n_1] \times \ldots \times [1..n_m] \rightarrow \mu$
- **Type-élément**
  - [**logical**] dénote le type *logical*
  - **Type-numérique**
    - [**integer**] dénote le type *integer*
    - **Type-réel**
      - [**real**] dénote le type *real*
      - [**dpreal**] dénote le type *long real*
    - [**complex**] dénote le type *complex*
- [**event**] dénote le type *event*

1. **Syntaxe Hors Contexte.**

   ♠ **TYPE-SIGNAL** ::=
   - • **TYPE-TABLEAU** •
   - • **Type-élément** •
   - • [**event**] •

### II-3.2 Ordre sur les types

**Ordre sur les types de synchronisation**
Le type *event* est inférieur au type *logical*.

**Ordre sur les types numériques**
Le type *integer* est inférieur au type *real*. Le type *real* est inférieur au type *complex*.

**Ordre sur les tableaux**

L'ordre sur les types numériques est étendu aux tableaux :

- $[1..m_1] \times \ldots \times [1..m_k] \rightarrow \mu \sqsubseteq [1..n_1] \times \ldots \times [1..n_l] \rightarrow \nu$ si et seulement si
  - $k = l$
  - $\forall i, 0 < i \leq k \Rightarrow n_i = m_i$
  - et $\mu \sqsubseteq \nu$

**Notation** On utilise la notation $\mu \sqcup \nu$ pour désigner la borne supérieure de deux types $\mu$ et $\nu$ compatibles.

### II-3.3 Conversions

Dans une expression binaire, un des arguments de type $\mu$ peut être implicitement converti au type supérieur $\nu$ si le second argument possède directement ce type $\nu$.

## II-4 Déclarations d'identificateurs de signaux

Une suite de valeurs est munie d'un type (celui de ses éléments); ce type est associé à un identificateur dans une déclaration. Un identificateur peut désigner un signal formel (paramètre) ou un signal local selon la même forme de déclaration.

Les déclarations d'identificateurs de signaux respectent la syntaxe suivante :

1. **Syntaxe Hors Contexte.**

   ♠ **S-DECLARATION** ::=

   • **TYPE-SIGNAL** { **SIGNAL** [,] ... } •
   • **Type-inféré** { **SIGNAL** [,] ... } •

   ♠ **SIGNAL** ::=

   • **Nom-*signal*** [ [**init**] **S-EXPR** ] •

   ♠ **Type-inféré** ::=

   • •

2. **Représentant.**
   $\triangleright\ \mu\ ID_1, \ldots, ID_i$ init $E_i, \ldots, ID_n\ \triangleleft$

3. **Types.**

   (a) Les noms déclarés doivent être distincts deux à deux. Le même type $\tau(ID_1)$ est attribué aux identificateurs $ID_1, \ldots, ID_n$ dans le contexte de la déclaration.
   - si $\mu$ est un **TYPE-SIGNAL** alors le type attribué aux identificateurs est le type $\tau(\mu)$ ;
   - si $\mu$ est un **Type-inféré** (omission de la notation de type dans la déclaration), le type attribué aux identificateurs est le résultat, s'il existe, de l'inférence de types. Un signal externe a le type *real* par défaut.

   (b) le type $\tau(E_i)$ de l'expression $E_i$ est le type $\tau(ID_i)$ éventuellement augmenté d'une dimension à gauche (cf. Expressions dynamiques)

4. **Horloges.**
   Les expressions d'initialisation $E_i$ sont des expressions constantes.

   (a) $\omega(E_i) = \hbar$

5. **Sémantique.**
   La valeur $E_i$ est une valeur initiale pour le signal $ID_i$; elle doit être présente si $ID_i$ est un signal obtenu en appliquant un retard, en prenant une fenêtre, ou en mémorisant la valeur

courante d'un autre signal. Cette valeur n'a donc de sens ni pour les paramètres de type quelconque, ni pour les signaux purs (*event*).

6. **Exemples.**

   (a) `real X, Y, ZX init 3.14, T`

   (b) `[n]real T init [{i to n}:i]`

7. **Anomalie.**

   *Un vecteur passé en paramètre ne peut initialiser un retard*

# Chapitre III

# Expressions sur signaux

Les valeurs associées aux signaux sont déterminées par des équations sur signaux ; ces équations sont construites par composition de sous-equations à partir d'équations élémentaires. Dans ce chapitre, nous présentons les expressions permettant de définir un signal (**S-EXPR**); cette présentation est précédée d'une introduction aux expressions de composition de définitions (**P-EXPR**).

## III-1 Equations de définition de signaux

### III-1.1 Equations élémentaires

Une définition de signaux permet de définir un signal ou un ensemble de signaux selon la syntaxe suivante :

1. **Syntaxe Hors Contexte.**

   ♠ **DEFINITION-DE-SIGNAUX** ::=

   - **Nom-*signal*** [:=] **S-EXPR** •
   - [{] { **Nom-*signal*** [,] ... } [}] [:=] **S-EXPR** •

2. **Représentant.**

   ▷ $X := E$ ◁

3. **Représentant.**

   ▷ $\{X_1,\ldots,X_n\} := E$ avec $< v_1,\ldots,v_n >$ le tuple de signaux définis par $E$ ◁

4. **Horloges.**

   Un identificateur et le signal qui le définit sont synchrones

   (a) $\omega(X) = \omega(E)$

   (b) $\omega(X_i) = \omega(v_i)$

5. **Types.**

   (a) $\tau(X) = \tau(E)$

   (b) $\tau(X_i) = \tau(v_i)$

6. **Profil.**

Une équation de définition de signaux possède les entrées et les sorties définies par les règles suivantes :

- Les identificateurs de signaux définis constituent les sorties de l'équation :
  - $!(X := E)=\{X\}$
  - $!(\{X_1,\ldots,X_n\} := E) =\{X_1,\ldots,X_n\}$
- Les entrées de l'équation sont les entrées de $E$ : ce sont les identificateurs de signaux ayant au moins une occurrence dans $E$, directement ou comme entrées visibles des invocations de modèles :
  $?(X := E)=?(E)$

7. **Sémantique.**

- le signal $X$ est égal au signal résultant de l'évaluation de $E$,
- chaque signal $X_i$ est respectivement égal au signal $v_i$

8. **Exemples.**

(a) si `x`, `y`, `z` désignent des signaux :
`x := y + z` définit le signal désigné par `x`, égal à la somme des signaux respectivement désignés par `y` et `z`; cette expression a pour entrées `y` et `z` et pour sortie `x`;

(b) si `x`, `y`, `a` désignent des signaux, et `Q` un modèle possédant deux sorties et une entrée :
`{x,y} := Q{a-5}` définit les signaux désignés par `x` et `y` respectivement égaux à la première et à la seconde sortie du modèle `Q`; cette expression a pour entrée `a` et pour sorties `x` et `y`;

(c) si `x`, `y`, `z`, `a` désignent des signaux, `P` un modèle possédant trois sorties et quatre entrées et `Q` un modèle possédant deux sorties et une entrée :
`{x,y,z} := P{a,Q{a-5},a+5}` définit les signaux désignés par `x`, `y` et `z` respectivement égaux à la première, à la seconde et à la troisième sortie du modèle `P`; cette expression a pour entrée `a` et pour sorties `x`, `y` et `z`;

## III-1.2 Composition de définitions de signaux

Les équations de définition de signaux peuvent être composées par l'opérateur |. Une expression $E_1$ | $E_2$, dite expression sur équation, définit les signaux (a pour sorties les signaux) définis dans chacune des sous-expressions et a pour entrées les signaux d'entrée de chacune de ces sous-expressions qui ne sont pas des sorties de l'autre (qui ne sont pas des sorties de l'autre); un signal ne pouvant avoir une double définition, un même identificateur de signal ne peut être une sortie des deux sous-expressions. La valeur d'un signal d'entrée d'une sous-expression définie dans l'autre est la valeur associée par cette définition.

Une expression sur équations peut être parenthésée par (| à gauche et |) à droite.

Une partie des sorties d'une expression peut être rendue invisible au moyen de l'opérateur /. Une expression $E_1$ $/a_1,\ldots,$ $a_n$ a pour sorties les sorties de $E_1$ qui ne figurent pas dans la liste $a_1, \ldots, a_n$ et pour entrées les entrées de $E_1$.

# III-2 Expressions élémentaires

1. **Syntaxe Hors Contexte.**

♠ **S-EXPR-ELEMENTAIRE** ::=

- **CONSTANTE** •
- **Nom-*signal*** •
- **INDEXATION** •
- **PRODUCTION** •

♠ **CONSTANTE** ::=

- **Cst-booléenne** •
- **Cst-entière** •
- **CST-REELLE** •
- **CST-COMPLEXE** •

## III-2.1 Expressions constantes

Une expression constante est une **CONSTANTE**, une occurrence d' identificateur de paramètre ou l'une des expressions suivantes ayant récursivement pour arguments des expressions constantes :

- une **S-EXPR-TABLEAU**
- une **S-EXPR-BOOLEENNE**
- une **S-EXPR-ARITHMETIQUE**

Les expressions temporelles (**S-EXPR-TEMPORELLE**) et les appels de fonctions (**PRODUCTION**) ne peuvent appartenir à une expression constante. Une constante est une dénotation de valeur d'un type scalaire

1. **Syntaxe Hors Contexte.**

♠ **CONSTANTE** ::=

- **Cst-booléenne** •
- **Cst-entière** •
- **CST-REELLE** •
- **CST-COMPLEXE** •

2. **Profil.**

Une constante et par conséquent une expression constante ne possèdent ni entrée nommée ni sortie nommée.

Une expression constante et ses arguments ont tous $\hbar$ pour horloge.

Le type d'une expression constante est évalué conformément au type de la **S-EXPR** ayant la même syntaxe.

## III-2.2 Occurrence d'identificateur de signal

Une occurrence d'identificateur de signal a pour valeur le signal qui définit cet identificateur et pour type le type de sa déclaration la plus interne ; le profil qui lui est associé contient pour entrée cet unique identificateur et ne contient pas de sortie nommée.

### III-2.3 Indexation

1. **Syntaxe Hors Contexte.**

♠ **INDEXATION** ::=

• **Nom-*signal*** [ { **S-EXPR** , ... } ] •

2. **Profil.**

$?(T[E_1,\ldots,E_m]) = \{T\} \cup \cup_{i=1}^{m} ?(E_i)$

3. **Horloges.**

Les signaux apparaissant dans une indexation sont synchrones

(a) $\omega(T[E_1,\ldots,E_m]) = \omega(T)$

(b) $\omega(T[E_1,\ldots,E_m]) = \omega(E_i)$

4. **Types.**

(a) si T est un tableau de type
$[1..n_1] \times \ldots \times [1..n_m] \rightarrow \tau(\nu)$,
et si les valeurs résultant de l'évaluation des expressions $E_i$ sont respectivement comprises entre 1 et $n_i$ alors
T$[E_1,\ldots,E_m]$ est un scalaire de type $\tau(\nu)$

(b) si T est un tableau de type
$[1..n_1] \times \ldots \times [1..n_m] \times [1..n_{m+1}] \times \ldots \times [1..n_p] \rightarrow \tau(\nu)$,
et si les valeurs résultant de l'évaluation des expressions $E_i$ sont respectivement comprises entre 1 et $n_i$ alors
T$[i_1,\ldots,i_m]$ est un tableau de type $[1..n_{m+1}] \times \ldots \times [1..n_p] \rightarrow \tau(\nu)$

(c) dans les autres cas l'indexation est incorrecte.

5. **Sémantique.**

La valeur d'une indexation est définie récursivement par

- la valeur de $T[E_1, E_2, \ldots, i_m]$ est la valeur $TT[i_2, \ldots, i_m]$ où $TT$ est la valeur de $T[E_1]$.
- la valeur de $T[E_1]$ est la valeur du $k^{eme}$ élément de $T$ lorsque $k$ est la valeur entière produite par l'évaluation de $E_1$
- si le nombre d'indices est correct, mais la valeur d'un de ces indices n'est pas comprise dans les bornes correspondantes, alors le résultat est non défini.

### III-2.4 Invocation de modèle

La **PRODUCTION** d'un processus par invocation d'un modèle est réalisée par macro-expansion du texte du modèle. Les paramètres statiques sont parenthésés par ( et ) ; ces paramètres sont des expressions constantes utilisées comme valeurs initiales de signaux ou tailles de tableaux. Les signaux d'entrée peuvent être associés à un exemplaire de modèle

- soit en "en position" (liste des expressions les définissant entre les caractères { et }),
- soit par identité sur les noms de signaux (liste vide entre les caractères { et }).

1. **Syntaxe Hors Contexte.**

   ♠ **PRODUCTION** ::=

   • **REFERENCE-MODELE** [{] [ { **S-EXPR** [,] ... } ] [}] •

   ♠ **REFERENCE-MODELE** ::=

   • **APPEL** •
   • **modèle** •

   ♠ **APPEL** ::=

   • **Nom-*modèle*** [(] [ { **S-EXPR** [,] ... } ] [)] •

2. **Représentant.**

   ▷ $P(V_1,\ldots,V_m)\{E_1,\ldots,E_n\}$ ◁

3. **Types.**

   (a) Pour chaque paramètre formel $P_i$ (nom de l'identificateur de paramètre en position $i$ dans la déclaration de $P$) et sa valeur effective associée $V_i$, $\tau(V_i) = \tau(P_i)$.

   (b) Pour chaque entrée $SE_i$ (nom de l'identificateur de signal d'entrée en position $i$ dans la déclaration de $P$) et son expression correspondante $E_i$, $\tau(E_i) = \tau(SE_i)$.

   (c) Pour chaque sortie $SS_i$ et son résultat attendu $F_i$, $\tau(SS_i) = \tau(F_i)$.

4. **Profil.**

   • $?(P(V_1,\ldots,V_m)\{E_1,\ldots,E_n\}) = \bigcup_{i=1}^{n} ?(E_i)$
   • $!(P(V_1,\ldots,V_m)\{E_1,\ldots,E\_n\})$ est l'ensemble des noms des sorties de la déclaration de $P$.

## III-3 Expressions arithmétiques

Les expressions arithmétiques sont des expressions synchrones sur signaux : les signaux arguments, ainsi que le signal résultant, ont la même horloge. Les opérateurs définissant de telles expressions sont les opérateurs arithmétiques standards étendus aux suites d'éléments.

1. **Syntaxe Hors Contexte.**

   ♠ **S-EXPR-ARITHMETIQUE** ::=

   • **S-EXPR** [+] **S-EXPR** •
   • **S-EXPR** [−] **S-EXPR** •
   • **S-EXPR** [*] **S-EXPR** •
   • **S-EXPR** [/] **S-EXPR** •
   • **S-EXPR** [**] **S-EXPR** •
   • [+] **S-EXPR** •
   • [−] **S-EXPR** •

2. **Priorités.**

   Ces opérateurs sont ordonnés selon les priorités croissantes ci-dessous :

(a) + et − binaire ont la même priorité ;

(b) * et / ont la même priorité ;

(c) **

(d) −, + unaires ont la même priorité ;

3. **Sémantique.**
Les expressions sont définies avec leur sémantique habituelle. Lorsqu'une expression de division est de type *integer*, la division est la division entière.
Si le résultat d'une expression n'est pas représentable dans le type $\mu$ de cette expression, sa valeur est une valeur de type $\mu$ dépendant de l'implémentation.

### III-3.1 Addition, soustraction, multiplication, division

Les opérateurs binaires de même priorité sont évalués de gauche à droite.

1. **Représentant.**
$\triangleright\ E_1\ \mathbf{Op}\ E_2\ \triangleleft$

2. **Types.**

(a) $\tau(E_1)$ et $\tau(E_2)$ sont des types numériques

(b) $\tau(E_1\ \mathbf{Op}\ E_2) = \tau(E_1) \sqcup \tau(E_2)$

3. **Horloges.**

(a) $\omega(E_1) = \omega(E_2)$

(b) $\omega(E_1\ \mathbf{Op}\ E_2) = \omega(E_1)$

### III-3.2 Elévation à la puissance

1. **Représentant.**
$\triangleright\ E_1\ **\ E_2\ \triangleleft$

2. **Types.**

- $\tau(E_1)$ est un type numérique.
- $\tau(E_2)$ est le type *integer*
- $\tau(E_1\ **\ E_2) = \tau(E_1)$

3. **Horloges.**

(a) $\omega(E_1) = \omega(E_2)$

(b) $\omega(E_1\ **\ E_2) = \omega(E_1)$

### III-3.3 Opérateurs d'arité 1

1. **Représentant.**
   $\triangleright$ *op* $E$ $\triangleleft$

2. **Types.**

   (a) $\tau(E)$ est un type numérique.

   (b) $\tau(op\ E) = \tau(E)$

3. **Horloges.**

   (a) $\omega(op\ E) = \omega(E)$

## III-4 Expressions booléennes

Les expressions booléennes sont des expressions synchrones sur signaux : les signaux arguments, ainsi que le signal résultant, ont la même horloge. Les opérateurs définissant de telles expressions sont les opérateurs standards sur éléments booléens étendus aux suites d'éléments. Les expressions booléennes (ou expression à résultat booléen) sont soit des relations, soit des expressions du treillis.

### III-4.1 Expressions sur booléens

1. **Syntaxe Hors Contexte.**

   ♠ **S-EXPR-BOOLEENNE** ::=
   - **S-EXPR** and **S-EXPR** •
   - **S-EXPR** or **S-EXPR** •
   - not **S-EXPR** •
   - **RELATION** •

2. **Priorités.**
   Ces opérateurs sont ordonnés selon les priorités croissantes ci-dessous :

   (a) **or** : ou logique ;

   (b) **and** : et logique ;

   (c) **not** : non logique ;

   (d) les opérateurs de relation.

3. **Types.**

   (a) $\tau(E_1) \sqsubseteq logical$

   (b) $\tau(E_2) \sqsubseteq logical$

   (c) $\tau(E_1\ \mathbf{or}\ E_1) = logical$

   (d) $\tau(E_1\ \mathbf{and}\ E_1) = logical$

   (e) $\tau(\mathbf{not}\ E_1) = logical$

4. **Horloges.**
   Les horloges des arguments des expressions sur booléens sont identiques ; ainsi pour *op* dénotant **or** ou **and**

   (a) $\omega(E_1)$ *op* $E_2 = \omega(E_1)$
   (b) $\omega(E_1)$ *op* $E_2 = \omega(E_2)$
   (c) $\omega$(**not** $E_1$) $= \omega(E_1)$

5. **Sémantique.**
   Les opérateurs booléens ont leur sémantique habituelle

6. **Définition en SIGNAL.**
   La disjonction booléenne n'est pas un opérateur primitif du **langage** SIGNAL:
   $X$ := $E_1$ **or** $E_2$ est égal au processus défini ci-dessous.

```
▷
( | X := (when E1) default (when E2) default (not event E1)
  | synchro{E1,E2}
  | )
◁
```

7. **Définition en SIGNAL.**
   La conjonction booléenne n'est pas un opérateur primitif du **langage** SIGNAL:
   $X$ := $E_1$ **and** $E_2$ est égal au processus défini ci-dessous.

```
▷
( | X := (when E1 when E2) default (not event E1)
  | synchro {E1,E2}
  | )
◁
```

## III-4.2 Relations

1. **Syntaxe Hors Contexte.**

   ♠ **RELATION** ::=
   - **S-EXPR** [=] **S-EXPR** •
   - **S-EXPR** [/=] **S-EXPR** •
   - **S-EXPR** [>] **S-EXPR** •
   - **S-EXPR** [>=] **S-EXPR** •
   - **S-EXPR** [<] **S-EXPR** •
   - **S-EXPR** [=] **S-EXPR** •

2. **Représentant.**
   ▷ $E_1$ **Op** $E_2$ ◁

3. **Types.**

   (a) $E_1$ et $E_2$ possèdent un même type scalaire différent du type *complex*.

4. **Sémantique.**
   Dans l'ordre défini sur les valeurs de type *logical*, *false* est inférieur à *true*. Avec cette précision, les opérateurs de relation ont leur sémantique habituelle.

# III-5 Expressions temporelles

Les expressions temporelles sont construites sur des signaux d'horloges éventuellement différentes.

1. **Syntaxe Hors Contexte.**

   ♠ **S-EXPR-TEMPORELLE** ::=

   - **COMPTEUR**
   - **MELANGE**
   - **EXTRACTION**
   - **MEMORISATION**
   - **HORLOGE-SIGNAL**
   - **EXTRACTION-HORLOGE**

2. **Priorités.**

   Ces opérateurs sont ordonnés selon les priorités croissantes ci-dessous :

   (a) les opérateurs **COMPTEUR** (`#`)

   (b) l'opérateur de **MELANGE** (`default`)

   (c) l'opérateur d'**EXTRACTION** (`when`)

   (d) l'opérateur de **MEMORISATION** (`cell`)

   (e) l'opérateur d'**EXTRACTION-HORLOGE** (`when` unaire) et **HORLOGE-SIGNAL** (`event`) ont même priorité.

## III-5.1 Mélange

Le mélange déterministe (avec priorité) est défini selon la syntaxe suivante :

1. **Syntaxe Hors Contexte.**

   ♠ **MELANGE** ::=

   - **S-EXPR** `default` **S-EXPR**

2. **Représentant.**

   ▷ $E_1$ `default` $E_2$ ◁

3. **Types.**

   (a) $\tau(E_1 \texttt{ default } E_2) = \tau(E_1) \sqcup \tau(E_2)$

4. **Horloges.**

   (a) $h = \omega(E_1 \texttt{ default } E_2)$

   (b) $h = \omega(E_1) \cup h$

   (c) si $\omega(E_2)$ est différent de $\hbar$, $h = \omega(E_1) \cup \omega(E_2)$

5. **Propriétés.**

(a) $(E_1$ **default** $E_2)$ **default** $E_3 = E_1$ **default** $(E_2$ **default** $E_3)$

6. **Exemples.**

(a) les valeurs prises par **X default Y** sont décrites ci-dessous pour les valeurs correspondantes de **X** et **Y** en entrée :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **X** | = | 1 | 3 | $\perp$ | 5 | 7 | ... |
| **Y** | = | 2 | 4 | 6 | $\perp$ | 8 | ... |
| **X default Y** | = | 1 | 3 | 6 | 5 | 7 | ... |

## III-5.2 Extraction

Les valeurs d'un signal peuvent être produites par extraction des valeurs d'un autre signal lorsque les valeurs d'un signal booléen sont **true**.

1. **Syntaxe Hors Contexte.**

♠ **EXTRACTION** ::=

• **S-EXPR** when **S-EXPR** •

2. **Représentant.**

$\triangleright$ $E$ **when** $B$ $\triangleleft$

3. **Types.**

(a) $\tau(B) \sqsubseteq$ *logical*

(b) $\tau(E$ **when** $B) = \tau(E)$

4. **Horloges.**

(a) $h = \omega(E$ **when** $B)$

(b) si $\omega(E) = \hbar$, $h = \omega(B) * (-1 - B)$

(c) si $\omega(E)$ est différent de $\hbar$, $h = \omega(E) * \omega(B) * (-1 - B)$

5. **Propriétés.**

(a) $(E$ **when** $B)$ **when** $C = E$ **when** $(B$ **when** $C)$

(b) $E$ **when** $(B$ **when** $B) = E$ **when** $B$

(c) $(E_1$ **default** $E_2)$ **when** $B = (E_1$ **when** $B)$ **default** $(E_2$ **when** $B)$

6. **Exemples.**

(a) les valeurs prises par **X when C** sont décrites ci-dessous pour les valeurs correspondantes de **X** et **C** en entrée :

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **X** | = | 1 | 3 | $\perp$ | 5 | $\perp$ | 7 | ... |
| **C** | = | T | $\perp$ | T | F | F | T | ... |
| **X when C** | = | 1 | $\perp$ | $\perp$ | $\perp$ | $\perp$ | 7 | ... |

### III-5.3 Horloge d'un signal

L'horloge d'un signal est obtenue par application de l'opérateur **event** à ce signal de type quelconque.

1. **Syntaxe Hors Contexte.**

   ♠ **HORLOGE-SIGNAL** ::=
   • event **S-EXPR** •

2. **Représentant.**
   ▷ **event** *B* ◁

3. **Types.**

   (a) $\tau$(**event** *B*) = *event*

4. **Exemples.**

   (a) les valeurs prises par **event X** sont décrites ci-dessous pour les valeurs correspondantes de **X** en entrée :

| | | | | | | |
|---|---|---|---|---|---|---|
| **X** | = | 1 | 2 | 3 | 4 | ... |
| **event X** | = | T | T | T | T | ... |

### III-5.4 Extraction d'horloge

L'extraction des valeurs *true* d'une condition booléenne est obtenue par application de l'opérateur **when** unaire :

1. **Syntaxe Hors Contexte.**

   ♠ **EXTRACTION-HORLOGE** ::=
   • when **S-EXPR** •

2. **Représentant.**
   ▷ **when** *B* ◁

3. **Types.**

   (a) $\tau(B) \sqsubseteq$ *logical*

   (b) $\tau$(**when** *B*) = *event*

4. **Définition en SIGNAL.**

   ▷ **(event (***B***)) when** *B* ◁

5. **Horloges.**

   (a) $\omega$(**when** *B*) = $\omega(B) * (-1 - B)$

6. **Exemples.**

(a) les valeurs prises par **when** C sont décrites ci-dessous pour les valeurs correspondantes de C en entrée :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| C | = | T | T | F | F | T | ... |
| when C | = | T | T | ⊥ | ⊥ | T | ... |

### III-5.5 Equations sur horloges

Une **EQUATION-SUR-HORLOGE** participe à la construction du système d'équations d'horloges du programme. Elle est l'outil de programmation par contraintes.

1. **Syntaxe Hors Contexte.**

♠ **EQUATION-SUR-HORLOGE** ::=

• [synchro] [{] { **S-EXPR** [,] ... } [}] •

2. **Représentant.**

▷ synchro $\{E_1, E_2\}$ ◁

3. **Profil.**

Une équation sur horloges est un processus sans sortie avec
?(synchro $\{E_1, E_2\}$)= ?($E_1$) ⋃ ?($E_2$)

4. **Types.**

(a) Les arguments $E_i$ sont de types quelconques.

5. **Définition en SIGNAL.**

synchro $\{E_1, E_2\}$
contraint à l'égalité des horloges des expressions sur signaux $E_1$ et $E_2$ ; cette expression est égale au processus sans sortie défini ci-dessous.

▷
( | $X$ := (event $E_1$) = (event $E_2$)
| ) / $X$
◁

### III-5.6 Compteur

Les expressions de compteur permettent de numéroter les occurrences d'une horloge.

#### III-5.6 A Compteur complet

Le compteur complet d'un signal est défini selon la syntaxe suivante :

1. **Syntaxe Hors Contexte.**

♠ **COMPTEUR** ::=

• [#] **S-EXPR** •

2. **Représentant.**

▷ # $C_1$ ◁

3. **Types.**

(a) $\tau(C_1) \sqsubseteq$ *logical*

(b) $\tau(\#\ C_1) =$ *integer*

4. **Horloges.**

(a) $\omega(\#\ C_1) = \omega(\texttt{when}\ C_1)$

5. **Sémantique.**

Le signal $N$, défini à l'horloge $H$ (obtenue par extraction des valeurs *true* d'un signal *logical* $C_1$), par l'expression $N$ := # $C_1$, énumère successivement les entiers strictement positifs ; $N$ compte ainsi le nombre d'occurrences à *true* du signal $C_1$.

6. **Définition en SIGNAL.**

$N$ := # $C_1$

est égal au processus défini ci-dessous.

```
▷
( | N := ZN + 1
  | ZN := N $ 1
  | synchro{N,when C1}
  | ) / ZN
```

dans lequel $ZN$ possède la valeur initiale 0 ◁

7. **Exemples.**

(a) les valeurs prises par # $C_1$ sont décrites ci-dessous pour les valeurs correspondantes de $C_1$ en entrée :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $C_1$ | = | F | T | T | F | T | ... |
| # $C_1$ | = | ⊥ | 1 | 2 | ⊥ | 3 | ... |

### III-5.6 B Compteur relatif

Le compteur relatif d'un signal est défini sous deux formes, selon la syntaxe suivante :

1. **Syntaxe Hors Contexte.**

♠ **COMPTEUR** ::=

- # **S-EXPR** after **S-EXPR** •
- # **S-EXPR** from **S-EXPR** •

2. **Représentant.**

```
▷ # C1 after C2
# C1 from C2 ◁
```

3. **Types.**

(a) $\tau(C_1) \sqsubseteq$ *logical*

(b) $\tau(C_2) \sqsubseteq$ *logical*

(c) $\tau(\#\ C_1\ mode\ C_2) =$ *integer*

4. **Horloges.**

(a) $\omega$(# $C_1$ *mode* $C_2$ ) = $\omega$((when $C_1$) default (when $C_2$))

5. **Sémantique.**
Le signal $N$ défini, à l'horloge $H$ obtenue par extraction des valeurs *true* d'un signal *logical* $C_1$, par l'expression $N$ := # $C_1$ *mode* $C_2$, compte ainsi le nombre d'occurrences à *true* du signal $C_1$ ($o_1$) depuis la dernière occurrence à la valeur *true* du signal $C_2$ ($o_2$); dans le cas du mode **from**, les occurrences $o_1$ simultanées à des des occurrences $o_2$ sont comptées ; dans le cas du mode **after**, les occurrences $o_1$ simultanées à des des occurrences $o_2$ ne sont pas comptées.

6. **Définition en SIGNAL.**
$N$ := # $C_1$ from $C_2$
est égal au processus défini ci-dessous
▷
( | $N_{IN}$ := ( (1 when $C_1$) default 0) when $C_2$
| $N$ := $N_{IN}$ default ( ($ZN$ + 1 )when $C_1$)
| $ZN$ := $N$ $ 1
| synchro{$N$,(when $C_1$) default (when $C_2$)}
| ) / $ZN$, $N_{IN}$
dans lequel $ZN$ possède la valeur initiale 0 ◁

7. **Définition en SIGNAL.**
$N$ := # $C_1$ after $C_2$
est égal au processus défini ci-dessous
▷
( | $N_{IN}$ := 0 when $C_2$
| $N$ := $N_{IN}$ default ( ($ZN$ + 1 )when $C_1$)
| $ZN$ := $N$ $ 1
| synchro{$N$,(when $C_1$) default (when $C_2$)}
| ) / $ZN$, $N_{IN}$
dans lequel $ZN$ possède la valeur initiale 0 ◁

8. **Exemples.**

(a) les valeurs prises par # $C_1$ **from** $C_2$ et # $C_1$ **after** $C_2$ sont décrites ci-dessous pour les valeurs correspondantes de $C_1$ et $C_2$ en entrée :

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $C_1$ | = | F | F | T | T | T | T | F | T | ⊥ | F | T | ... |
| $C_2$ | = | T | ⊥ | F | ⊥ | T | ⊥ | F | F | ⊥ | T | F | ... |
| # $C_1$ **from** $C_2$ | = | 0 | ⊥ | 1 | 2 | 1 | 2 | ⊥ | 3 | ⊥ | 0 | 1 | ... |
| # $C_1$ **after** $C_2$ | = | 0 | ⊥ | 1 | 2 | 0 | 1 | ⊥ | 2 | ⊥ | 0 | 1 | ... |

## III-5.7 Mémorisation

La mémorisation d'un signal à l'horloge définie par les occurrences à *true*d'une condition booléenne respecte la syntaxe suivante :

1. **Syntaxe Hors Contexte.** Les deux arguments ont une priorité au plus égale à celle de l'opérateur d'extraction unaire.

♠ MEMORISATION ::=
• S-EXPR cell S-EXPR •

2. **Représentant.**
   ▷ $E$ `cell` $B$ ◁

3. **Types.**

   (a) $\tau(B) \sqsubseteq$ *logical*

   (b) $\tau(E$ `cell` $B) = \tau(E)$

4. **Définition en SIGNAL.**
   $X$ `:=` $E$ `cell` $B$
   dont la partie située à droite du := représente une expression de mémorisation de $E$ aux instants où $B$ possède la valeur **true**, est égal au processus défini ci-dessous.

```
▷
( | X := E default (X $ 1)
  | synchro{X,E default (when B)}
  | )
◁
```

5. **Horloges.**

   (a) $\omega($ $E$ `cell` $B) = \omega(E)+(1-\omega(E))*(-B-\omega(B))$

6. **Exemples.**

   (a) les valeurs prises par `Y := X cell C` sont décrites ci-dessous pour les valeurs correspondantes de `X` et `C` en entrée sous l'hypothèse que `Y` soit initialisé par `0` :

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| X | = | ⊥ | 1 | 3 | ⊥ | ⊥ | ⊥ | 5 | ⊥ | 7 | ... |
| C | = | T | ⊥ | T | T | F | T | F | T | ⊥ | ... |
| Y | = | 0 | 1 | 3 | 3 | ⊥ | 3 | 5 | 5 | 7 | ... |

# III-6 Expressions dynamiques

Les expressions dynamiques permettent de manipuler des valeurs de signaux de dates distinctes.

1. **Syntaxe Hors Contexte.**

   ♠ **S-EXPR-DYNAMIQUE** ::=

   • **RETARD** •
   • **FENETRE** •

## III-6.1 Retard

Une expression de retard est définie selon la syntaxe suivante :

1. **Syntaxe Hors Contexte.**

   ♠ **RETARD** ::=

   • Nom-*signal* **$** **S-EXPR** •

2. **Représentant.**
   ▷ $X$ `$` $N_1$ ◁

3. **Types.**

   (a) $N_1$ est un entier strictement positif,

   (b) $\tau(X \$ N_1) = \tau(X)$

4. **Horloges.**

   (a) $\omega(N_1) = \hbar$

   (b) $\omega(X \$ N_1) = \omega(X)$

5. **Sémantique.**

   L'initialisation du signal $ZX$, défini par $ZX := X \$ N_1$ doit être

   - si $N_1 = 1$, une constante de même type que $X$,
   - si $N_1 > 1$
     - si $X$ est un scalaire, un tableau de dimension $[N_1]$ dont les éléments ont le type $\tau(X)$
     - si $X$ est un tableau de dimension $[I_1, \ldots, I_2]$, un tableau de dimension $[N_1, I_1, \ldots, I_2]$ dont les éléments ont le type des éléments de $X$.

   La valeur du signal $ZX$ est à chaque instant $t$, la valeur du signal retardé $X$ à l'instant $t - N_1$. Si le retard apparait comme sous-expression d'une expression de définition d'un signal, sa valeur initiale est indéfinie.

6. **Exemples.**

   (a) les valeurs prises par **x $ 1**, avec la valeur initiale 0, sont décrites ci-dessous pour les valeurs correspondantes de **x** en entrée :

| | | | | | | |
|---|---|---|---|---|---|---|
| **x** | = | 1 | 2 | 3 | 4 | ... |
| **x $ 1** | = | 0 | 1 | 2 | 3 | ... |

## III-6.2 Fenêtre glissante

Une expression de fenêtre glissante sur un signal est définie selon la syntaxe suivante :

1. **Syntaxe Hors Contexte.**

   ♠ **FENETRE** ::=

   • **Nom-*signal*** [ **$** **S-EXPR** ] **window** **S-EXPR** •

2. **Représentant.**

   ▷ $X$ **$** $N_1$ **window** $N_2$ ◁

3. **Types.**

   (a) $N_1$ et $N_2$ sont deux entiers strictement positifs ; en l'absence de retard, $N_1 = 0$

   (b)• si $\tau(X)=\mu$ scalaire alors l'expression est un vecteur possédant $N_2$ éléments

   • si $X$ est un tableau de type $[I_1, \ldots, I_2]\mu$, le type de la fenêtre est $[N_2, I_1, \ldots, I_2]\mu$.

4. **Horloges.**

(a) $\omega(N_1) = \hbar$

(b) $\omega(N_2) = \hbar$

(c) $\omega(X \texttt{ \$ } N_1 \texttt{ window } N_2) = \omega(X)$

5. **Sémantique.**

Dans la déclaration du signal $ZX$ défini par $ZX$ `:=` $X$ `$` $N_1$ `window` $N_2$,

- si $N_1+N_2 = 1$ alors le signal $ZX$ n'est pas initialisé,
- si $N_1+N_2 > 1$ alors le signal $ZX$ doit être initialisé
  - si $X$ est un scalaire, par un vecteur de taille $N_1 + N_2$ - 1 ;
  - si $X$ est un tableau de dimension $[I_1, \ldots, I_2]$, par un tableau de dimension $[N_1 + N_2 - 1, I_1, \ldots, I$
- ANOMALIE : l'initialisation de $ZX$ est obtenue uniquement par une énumération explicite (comme par exemple `[{to 1}:v0]`)

Si la fenêtre apparait comme sous-expression d'une expression de définition d'un signal, sa valeur initiale est indéfinie.

Une fenêtre est un vecteur $ZX$ dont chaque élément $ZX[i]$ est, à l'instant $t$, la valeur de $X$ à l'instant $[t - N_1 - N_2 + i]$

6. **Exemples.**

(a) les valeurs prises par `x $ 1 window 2`, avec une initialisation `[{to 2}:0]` sont décrites ci-dessous pour les valeurs correspondantes de `x` en entrée :

| `x` | = | 1 | 2 | 3 | 4 | ... |
|---|---|---|---|---|---|---|
| `x $1 window 2` | = | [0,0] | [0,1] | [1,2] | [2,3] | ... |

# III-7 Expressions sur tableaux

La manipulation de tableaux est rendue possible par l'extension d'opérations sur scalaires, la concaténation et l'énumération d'éléments.

1. **Syntaxe Hors Contexte.**

♠ **S-EXPR-TABLEAU** ::=

- **CONCATENATION** •
- **ENUMERATION** •

## III-7.1 Concaténation

1. **Syntaxe Hors Contexte.**

♠ **CONCATENATION** ::=

- { **Nom-*signal*** $\boxed{||}$ ... } •

2. **Sémantique.**

La **CONCATENATION** de deux tableaux

- $T_1$ de type $[1..m_1] \times [1..n_2] \times \ldots \times [1..n_m] \rightarrow \mu$

- $T_2$ de type $[1..m_2] \times [1..n_2] \times \ldots \times [1..n_m] \rightarrow \mu$

construit le tableau $T$, de type $[1..(m_1 + m_2)] \times [1..n_2] \times \ldots \times [1..n_m] \rightarrow \mu$ vérifiant :

- si $i_1$ est inférieur ou égal à $m_1$, $T[i_1, i_2, \ldots, i_m]$=$T_1[i_1, i_2, \ldots, i_m]$
- si $i_1$ est strictement supérieur à $m_1$, $T[i_1, i_2, \ldots, i_m]$=$T_2[(i_1 - m_1), i_2, \ldots, i_m]$

## III-7.2 Expression d'itération

### 1. Syntaxe Hors Contexte.

♠ **ENUMERATION** ::=
- [ [ ] { **ITERATION** [ , ] ... } [ ] ]

♠ **ITERATION** ::=
- **ELEMENT-SIMPLE**
- **ELEMENT-MULTIPLE**

♠ **ELEMENT-SIMPLE** ::=
- [ [ ] { **S-EXPR** [ , ] ... } [ ] ] [ : ] **S-EXPR**

♠ **ELEMENT-MULTIPLE** ::=
- [ { ] { **ITERANT** [ , ] ... } [ } ] [ : ] **S-EXPR**
- [ { ] { **ITERANT** [ , ] ... } [ } ] [ : ] **ELEMENT-SIMPLE**

♠ **ITERANT** ::=
- [ Nom-*indice* ] [ **DEBUT** ] [ **FIN** ] [ **PAS** ]

♠ **DEBUT** ::=
- [ in ] **S-EXPR**

♠ **FIN** ::=
- [ to ] **S-EXPR**

♠ **PAS** ::=
- [ step ] **S-EXPR**

### 2. Sémantique.

Une **ENUMERATION** définit un tableau $T$ de type $[1..n_1] \times \ldots \times [1..n_m] \rightarrow \mu$ par une suite d'actions exécutées **séquentiellement.**
Chaque action est soit une définition d'un élément isolé (**ELEMENT-SIMPLE**), soit une **ITERATION** permettant de définir un ensemble de valeurs.

- Un **ELEMENT-SIMPLE** est défini par le couple de la liste $[I_1, \ldots, I_m]$ de ses indices (dans le tableau) et de la valeur $E_{[I_1, \ldots, I_m]}$ en ces indices ;
- une itération est définie par le couple d'une liste d'itérants et des expressions de la valeur en chacun des points atteints par le parcours ;
- les itérants constituent une suite de déclarations visibles dans le reste de la définition de l' **ELEMENT-MULTIPLE** qui les contient ; ils définissent un ensemble de boucles,

imbriquées de gauche à droite ; l'indice de chaque boucle prend successivement la valeur $\varphi$(**DEBUT**), puis la valeur de $\varphi$(**DEBUT**) incrémentée de la valeur $\varphi$(**PAS**),... jusqu'à la dernière valeur comprise entre $\varphi$(**FIN**) et $\varphi$(**DEBUT**)

- si un terme est omis, il est implicitement égal à 1 ;
- lorsque l'expression de définition d'une valeur en un point est une **S-EXPR**, elle est équivalente à un **ELEMENT-SIMPLE** dont la liste des indices est formée de la suite des identificateurs (au besoin créés) des itérants

3. **Exemples.**

(a) `T := [ {to 10}: 0 ]` définit un vecteur nul

(b) `T := [ {to 10 , to 10}: 0 , {i in 1 to 10 , j in i to 10}: (i+j) ]` définit une matrice triangulaire supérieure

### III-7.3 Extension des expressions scalaires

Les expressions sur scalaires, à l'exclusion des relations sont étendues terme à terme aux tableaux.

## III-8 Ensemble des expressions sur signaux

### III-8.1 Expressions scalaires

Les expressions arithmétiques (**S-EXPR-ARITHMETIQUE**), booléennes (**S-EXPR-BOOLEENNE**) temporelles (**S-EXPR-TEMPORELLE**) et dynamiques **S-EXPR-DYNAMIQUE** forment les expressions scalaires selon la grammaire ci-dessous :

1. **Syntaxe Hors Contexte.**

♠ **S-EXPR-SCALAIRE** ::=

- **S-EXPR-ARITHMETIQUE** •
- **S-EXPR-BOOLEENNE** •
- **S-EXPR-TEMPORELLE** •
- **S-EXPR-DYNAMIQUE** •

2. **Profil.**

Ces expressions ne produisent pas de sortie nommée ; elles ont pour entrées respectivement :

- $?(op\ E) = ?(E)$ pour un opérateur *op* d'arité 1
- $?(E_1\ op\ E_2) = ?(E_1) \bigcup ?(E_2)$ pour un opérateur *op* d'arité 2

Une **S-EXPR-SCALAIRE** est évaluée selon les priorités décroissantes définies ci-dessous :

1. **S-EXPR-ARITHMETIQUE**
2. **S-EXPR-BOOLEENNE**
3. **S-EXPR-TEMPORELLE**
4. **S-EXPR-DYNAMIQUE**

Les priorités des opérateurs de chaque structure syntaxique sont décrites dans les sections correspondantes. Toute expression moins prioritaire que l'expression dont elle est argument doit être parenthésée. Le parenthésage est possible mais non nécessaire dans les autres cas. Pour une même priorité, sauf indication contraire, l'ordre d'évaluation des expressions d'arité supérieure à 1 est de gauche à droite. Les seules expressions non parenthésées autorisées sont construites sur les opérateurs associatifs (ou pseudo associatifs comme **when**).

Les extensions aux tableaux des expressions scalaires respectent les règles énoncées ci-dessus.

## III-8.2 Imbrication des expressions sur signaux

Les expressions sur signaux sont organisées en expressions élémentaires, expressions à structure scalaire et expressions à structure tableau.

### 1. Syntaxe Hors Contexte.

♠ **S-EXPR** ::=

- • [ ( **S-EXPR** ) ] •
- • **S-EXPR-ELEMENTAIRE** •
- • **S-EXPR-SCALAIRE** •
- • **S-EXPR-TABLEAU** •

Sous réserve d'être correctement typée, une **S-EXPR-ELEMENTAIRE** peut être utilisée comme argument dans toute expression sur signaux. Une **S-EXPR-SCALAIRE** ne peut posséder une **S-EXPR-TABLEAU** comme argument et inversement.

# Chapitre IV

# Expressions sur processus

Les expressions sur processus permettent de composer des systèmes d'équations sur signaux selon la grammaire ci-dessous:

1. **Syntaxe Hors Contexte.**

   ♠ **P-EXPR** ::=

   - [ **(** ] **P-EXPR** [ **)** ] •
   - **PROCESSUS-ELEMENTAIRE** •
   - **COMPOSITION** •
   - **PROFIL** •
   - **MOTIF** •

## IV-1 Processus élémentaires

Un processus élémentaire est une définition de signaux, un exemplaire de processus ou une équation sur horloges.

1. **Syntaxe Hors Contexte.**

   ♠ **PROCESSUS-ELEMENTAIRE** ::=

   - **EXEMPLAIRE-DE-PROCESSUS** •
   - **DEFINITION-DE-SIGNAUX** •
   - **EQUATION-SUR-HORLOGE** •

   ♠ **EXEMPLAIRE-DE-PROCESSUS** ::=

   - **APPEL** •
   - **PRODUCTION** •

Les processus élémentaires sont définis dans le chapitre précédent.

## IV-2 Composition

La composition de processus est définie selon la syntaxe de la première règle ci-dessous :

1. **Syntaxe Hors Contexte.**

♠ **COMPOSITION** ::=

• [(|] { **P-EXPR** [|] ... } [|)] •

2. **Représentant.**

$\triangleright\ P_1 \mid P_2\ \triangleleft$

3. **Profil.**

- $!(P_1) \cap !(P_2)$ est vide,
- $!(P_1 \mid P_2) = !(P_1) \cup !(P_2)$
- $?(P_1 \mid P_2) = (?(P_1) - !(P_2)) \cup (?(P_2) - !(P_1))$

4. **Sémantique.**

Un signal en entrée de $P_1$ (respectivement de $P_2$) ayant pour nom le nom d'un signal en sortie de $P_2$ (respectivement de $P_1$) a pour définition dans $P_1$ (respectivement dans $P_2$) sa définition dans $P_2$ (respectivement dans $P_1$).

Les contraintes sur les types et les horloges sont celles de $P_1$ et $P_2$.

## IV-3 Profil

1. **Syntaxe Hors Contexte.**

♠ **PROFIL** ::=

- **P-EXPR** [?] { **MODIFICATION** [,] ... } •
- **P-EXPR** [!] { **MODIFICATION** [,] ... } •
- **P-EXPR** [/] { **Nom-*signal*** [,] ... } •
- **P-EXPR** [!!] { **Nom-*signal*** [,] ... } •
- **P-EXPR** [@] { **Nom-*signal*** [,] ... } •

Le premier argument d'une expression de **PROFIL** est une **P-EXPR** qui ne peut être directement une **DEFINITION-DE-SIGNAUX**

### IV-3.1 Renommages

Les expressions de renommage en entrée ou en sortie sont obtenues au moyen de **MODIFICATION**s.

1. **Syntaxe Hors Contexte.**

♠ **MODIFICATION** ::=

• **Nom-*signal*** [:] **Nom-*signal*** •

2. **Sémantique.**

Une expression de renommage est un changement de nom de signaux.

- en entrée, l'expression
  $P$ `?` $a_1 : b_1, \ldots, a_n : b_n$ est égale au processus `P` dans lequel les noms de signaux d'entrée $a_i$ ont été respectivement remplacés par les noms $b_i$
- en sortie, l'expression
  $P$ `!` $a_1 : b_1, \ldots, a_n : b_n$ est égale au processus `P` dans lequel les noms de signaux de sortie $a_i$ ont été respectivement remplacés par les noms $b_i$. Les noms $b_i$ sont tous distincts ; si un nom $b_j$ est déjà le nom d'une sortie de $P$, ce nom doit être renommé (il est l'un des $a_i$).

3. **Exemples.**

(a) Considérant un processus `P` ayant `a`, `b` et `c` comme entrées et `x` et `y` comme sorties

(b) `P ? a : b` ne possède plus que deux entrées `b` (à laquelle l'entrée `a` de `P` est identifiée) et `c`

(c) `P ! x : y` est interdit

(d) `P ! x : y , y: x` inverse les noms `x` et `y`

## IV-3.2 Confinement

1. **Sémantique.**

Les opérations de confinement permettent de masquer des sorties d'un processus $P$

- par oblitération $P$ `/` $b_1, \ldots, b_n$ : le processus résultant a pour sorties, les sorties de $P$ **qui ne figurent pas** dans la liste $b_1, \ldots, b_n$
- par validation $P$ `!!` $b_1, \ldots, b_n$ : le processus résultant a pour sorties, les sorties de $P$ **qui figurent** dans la liste $b_1, \ldots, b_n$

2. **Exemples.**

(a) Considérant un processus `P` ayant `a`, `b` et `c` comme entrées et `x` et `y` comme sorties

(b) `P !! x` ne possède plus que la sortie `x`

(c) `P / y` ne possède plus que la sortie `x`

(d) `P / z` est égal à `P`.

## IV-3.3 Fermeture

1. **Sémantique.**

La fermeture permet d'identifier une sortie d'un processus $P$ avec l'entrée de même **Nom-*signal*** ; par fermeture $P$ `@`$b_1, \ldots, b_n$ :

- le processus résultant a pour entrées, les entrées de $P$ **qui ne figurent pas** dans la liste $b_1, \ldots, b_n$ et pour sorties, les sorties de $P$.
- les entrées de $P$ **qui figurent** dans la liste $b_1, \ldots, b_n$ ont respectivement pour valeurs les valeurs de ces sorties

2. **Exemples.**

(a) Considérant un processus `P` ayant `a`, `b` et `c` comme entrées et `a` et `y` comme sorties

(b) `P @ a` ne possède plus que les entrées `b` et `c`

(c) `P @ y` est égal à `P`.

# IV-4 Motif

Un **MOTIF** construit un processus par itération d'une même cellule élémentaire.

1. **Syntaxe Hors Contexte.**

   ♠ **MOTIF** ::=

   • [array] **I-MOTIF** [of] **S-EXPR** [ [with] { **CONNEXION** [,] ... } ] [end] •

   ♠ **I-MOTIF** ::=

   • **Nom-*indice*** [to] **S-EXPR** •

   ♠ **CONNEXION** ::=

   • **Nom-*signal*** [0]: **Nom-*signal*** •
   • **Nom-*signal*** [.]: **Nom-*signal*** •
   • **Nom-*signal*** •

2. **Représentant.**

   ▷ `array` $I$ `to` $N$ `of` $P$ `with` $c_1$, $c_2$ `end` ◁

3. **Types.**

   (a) $\tau(N)$=*integer*

4. **Sémantique.**

   Le processus $PN$ défini par l'expression
   `array` $I$ `to` $N$ `of` $P$ `with` *indexation* `end`
   est constitué de la succession des processus $P_i$ construits sur le modèle de $P$.

   - Pour toute sortie $x$ de $P$ possédant le type $\mu$, $PN$ possède une sortie $X$ de même nom et de type $[1..N] \rightarrow \mu$, telle que $X[i]$ est la sortie $x$ de $P_i$.
   - Si $a$ est une entrée de $P$ qui n'est pas présente dans les *indexations*, $a$ est une entrée de $PN$ qui est diffusée aux $N$ cellules.
   - Si $a$ est une entrée de $P$ présente dans les *indexations* sous la forme $a$, chaque entrée de cellule $P_i$ possédant ce nom est indexée par $i$.
   - Si $a$ est une entrée de $P$ présente dans les *indexations* sous la forme `a[.]:b`, chaque entrée de cellule $P_i$ possédant ce nom est indexée par $i+1$ ; $P$ possède une sortie de nom $a$ ;
   - Si $a$ est une entrée de $P$ présente dans les *indexations* sous la forme `a[0]:b`, chaque entrée de cellule $P_i$ possédant ce nom est indexée par $i-1$ ; $P$ possède une sortie de nom $a$ ;
   - Chaque entrée indexée $a$ est connectée (est égale) à la sortie de même nom possédant le même indice.
   - Une entrée $a$ d'indice 0 (respectivement $N+1$) donnée par `a[0]:b` (respectivement `a[.]:b`) devient l'entrée $b$ du processus $PN$.
   - Pour toute entrée $x$ de $P$, indexée et non connectée, possédant le type $\mu$, $PN$ possède une entrée $X$ de même nom de type $[1..N] \rightarrow \mu$, telle que $X[i]$ est l'entrée $x$ de $P_i$.
   - Le processus $PN$ ne possède que les entrées et les sorties définies selon les règles ci-dessus.

- **Restriction** : on ne peut avoir à la fois des entrées indexées sous la forme `a[0]:b` et des entrées indexées sous la forme `a[.]:b`.

5. **Exemples.**

(a) `array i to 10 of s:=a+b with a,b end` est équivalent à `s:=a+b`

(b) `array i to 10 of s:=a[i]+b[i] end` est équivalent à `s:=a+b`

(c) `array i to 10 of (| s:=y+b[i] | y:=a[i] |) with y end / y` est équivalent à `s:=a+b`

(d)
```
( | s0:=0
  | array i to 10 of s:=s+a*b with a,b,s[0]:s0 end
  | ps:=s[10]
  | )
```
produit dans `ps` le produit scalaire des vecteurs `a` et `b`

6. **Anomalie.**

*Si un signal externe au motif est utilisé pour définir une horloge dans le motif, il est considéré comme entrée de la boucle, même quand ce n'est pas nécessaire*

7. **Anomalie.**

*Un retard dans un motif provoque une erreur du compilateur*

## IV-5 Modèle de processus

Un modèle de processus établit une désignation entre un nom et un ensemble d'équations paramétrables; toute référence à ce nom est formellement remplacée par les équations désignées. Si l'ensemble d'équations est vide (invocation d'une fonction externe), le remplacement reste évidemment partiel (limité aux propriétés externes du processus invoqué); l'effet de l'invocation d'un processus externe (qui pourrait être déjà compilé, ou non conforme à sa description) ne peut être que théoriquement décrit. Tout modèle de processus invoqué dans le programme doit avoir une déclaration visible dans le contexte syntaxique de l'invocation. Un modèle de processus est un terme dérivé de **MODELE** selon la grammaire ci-dessous.

1. **Syntaxe Hors Contexte.**

♠ **MODELE** ::=

- **MODELE-EXTERNE** •
- **MODELE-DECRIT** •

♠ **MODELE-EXTERNE** ::=

- [function] **Nom-*modèle*** [=] **INTERFACE** •

♠ **MODELE-DECRIT** ::=

- [process] **Nom-*modèle*** [=] **INTERFACE DESCRIPTION** [end] •

♠ **DESCRIPTION** ::=

- **P-EXPR** [ [where] **DECLARATIONS** ] •

♠ **DECLARATIONS** ::=

- [ { **S-DECLARATION** [;] ... } ] [ { **MODELE** [;] ... } ] •

## IV-5.1 Déclarations locales de modèles de processus

L'ensemble des sous-modèles $\mathcal{M}$ déclarés dans un modèle $P$ ne doit pas contenir deux modèles de même nom. Une déclaration locale d'un modèle $Q$ est visible (peut être l'objet d'une **REFERENCE-MODELE**) dans l'expression associée à $P$ et aux expressions associées aux autres sous-modèles de $P$. Pour ces expressions, elle masque un éventuel modèle de même nom qui, sans elle, se trouverait visible. L'expression associée à un modèle $P$ ne peut contenir transitivement de référence à $P$.

## IV-5.2 Interface d'un modèle

L'interface d'un modèle comporte une description optionnelle de ses paramètres formels suivie d'une description de sa partie visible ; celle-ci est formée des listes éventuellement vides de ses signaux d'entrée et de sortie dans cet ordre.

1. **Syntaxe Hors Contexte.**

♠ **INTERFACE** ::=
- [ **PARAMETRES** ] [{] **ENTREES SORTIES** [}] •

♠ **PARAMETRES** ::=
- • [(] [ { **S-DECLARATION** [;] ... } ] [)] •

♠ **ENTREES** ::=
- • [?] [ { **S-DECLARATION** [;] ... } ] •

♠ **SORTIES** ::=
- • [!] [ { **S-DECLARATION** [;] ... } ] •

2. **Horloges.**

(a) Si un sous-modèle est une fonction externe, ses entrées et ses sorties sont synchrones.

Les noms des paramètres, des signaux d'entrée et des signaux de sortie doivent être distincts deux à deux. Un modèle doit posséder au moins une entrée ou une sortie ou une communication avec un processus externe d'horloge non nulle.

## IV-5.3 Signaux locaux

Les noms des signaux déclarés localement dans un modèle doivent être distincts deux à deux; ils doivent être distincts des noms des signaux d'interface et des paramètres.

# Annexe A

# Utilisation du compilateur

## A-1 Construction d'un programme

Un programme EXEMPLE du **langage** SIGNAL est contenu dans un fichier de nom quelconque, soit SOURCE.
Si ce programme possède des paramètres formels, leurs valeurs doivent être contenues dans un fichier également de nom quelconque, soit PARAMETRE.

**L'appel du compilateur** est réalisé par :

- si le programme EXEMPLE n'a pas de paramètre
  `sig {liste d'options} SOURCE`
- si le programme EXEMPLE a des paramètres
  `sig {liste d'options} SOURCE PARAMETRE`

Chaque option du compilateur est préfixée par le caractère "-" pour la version UNIX du compilateur et par le caractère "/" pour la version VMS.
Par défaut (c'est-à-dire sans option), le compilateur :

- produit un fichier EXEMPLE_LIS.SIG où EXEMPLE est le nom du programme Signal. Ce fichier contient :
  - le programme Signal initial annoté éventuellement par des messages d'erreurs ou des remarques.
  - les éventuelles erreurs liées à un cycle dans les dépendances de données (ou d'horloges).
- ne génère pas de code séquentiel (voir les options c et fortran).
- ne fournit pas de représentation externe du programme obtenu après résolution du système d'équations (voir les options tra et z3z).

Les différentes options sont les suivantes :

- nolist : pas de création du fichier EXEMPLE_LIS.SIG
- nowar : le compilateur ne donne pas de remarques (ou warnings)

- tra : le compilateur construit un fichier EXEMPLE_TRA.SIG qui contient un programme Signal équivalent au programme compilé après calcul d'horloges (cf. le paragraphe A-1.3)
- z3z : le compilateur construit un fichier EXEMPLE.Z3Z qui contient une représentation externe sous forme polynomiale, dans la syntaxe MACSYMA, des expressions de synchronisation après le calcul d'horloge.
- fortran : génération de code FORTRAN ; le compilateur construit trois fichiers EXEMPLE_M.f, EXEMPLE_S.f, EXEMPLE_E.f (cf. le paragraphe A-1.5)
- fortran=x : génération des fichiers spécifiés par la suite x constituée d'une combinaison des caractères "s", "m" ou "e" :
  - "m" production du fichier EXEMPLE_M.f
  - "s" production du fichier EXEMPLE_S.f
  - "e" production du fichier EXEMPLE_E.f
- c : génération de code C ; le compilateur construit trois fichiers EXEMPLE_M.c, EXEMPLE_S.c, EXEMPLE_E.c (cf. le paragraphe A-1.6)
- c=x : génération des fichiers spécifiés par la suite x constituée d'une combinaison des caractères "s", "m" ou "e" :
  - "m" production du fichier EXEMPLE_M.c
  - "s" production du fichier EXEMPLE_S.c
  - "e" production du fichier EXEMPLE_E.c

**Le résultat de la compilation** La compilation peut échouer durant l'analyse syntaxique, durant l'analyse des propriétés contextuelles (erreurs de type ou de profil), durant la génération de code.

### A-1.1 Echec à l'analyse syntaxique

En cas d'erreur à l'analyse syntaxique, le compilateur sort le numéro de la ligne où se trouve l'erreur ainsi qu'un code de récupération ; ce code désigne la plus petite structure syntaxique dans laquelle un essai de récupération est fait. Selon la correction de cette tentative de récupération, les erreurs détectées par la suite peuvent ne pas être réelles.

**Exemple de résultat**

```
   SIGNAL VERSION H_2.1  - 10 Oct 1989 / 8 Nov 1990 -
===> Analyse du programme
Erreur de Syntaxe 10 ligne 19
```

### A-1.2 Echec à l'analyse des propriétés contextuelles

En cas de messages provenant du compilateur et si l'utilisateur n'a pas mis l'option nolist, le fichier EXEMPLE_LIS_SIG peut être visualisé. Ce fichier contient le programme SOURCE muni des messages placés près de l'instruction en cause.

### A-1.2 A Remarques

Le compilateur peut découvrir des instructions non conformes à la définition du langage SIGNAL. Si la récupération est possible et jugée non dangereuse, le problème est simplement signalé à l'utilisateur à l'aide d'un message de la forme suivante (extraite de l'exemple ci-dessous)

```
%** REM:sortie ZERO calculee et non specifiee dans ...
```

et la compilation continue.

### A-1.2 B Erreurs

En cas d'erreur un message est affiché près de l'utilisation erronée du signal ; un second message est affiché à la déclaration du signal en cause.

- forme du premier message

```
%** ERR (ref 1): Dim. de WE incoherente avec ... %
```

- forme du second message

```
 %** ERR: cf message No 1
** ERR: cf message No 2
%[ N ]real WE init[ { to N }: 0.0 e 0 ]
```

### A-1.2 C Exemple complet

Après lancement du compilateur, il apparait à l'écran :

```
SIGNAL VERSION H_2.1  - 10 Oct 1989 / 8 Nov 1990 -
===> Analyse du programme
===> Analyse des parametres
===> Reduction dans le langage noyau
===> Generation du graphe des dependances.
```

Le fichier ERREUR_COMPILATION_SIG contient :

```
process ERREUR_COMPILATION_LIS=
    { ?
      ! }
    (| ERREUR_COMPILATION( 4, 3, 3.0 e 0,[ [ 1 ]: 1.0 e 0,[ 2 ]: 2.0 e 0,[ 3 ]:
                                          3.0 e 0 ],[ [ 1 ]: 3.0 e 0,[ 2 ]: 2.
                                           0 e 0,[ 3 ]: 1.0 e 0 ])
     |)
    where
    process ERREUR_COMPILATION=
         ( integer N, M;
           real B0;
           [ N-1 ]real B;
```

```
        [ M ]real A )
      { ? real E
        ! real S }
      (S:= ( E*BO )+PRODSCAL( N-1, B ){ E }+PRODSCAL( M, A ){ S })@ S
      where
      process PRODSCAL=
          %** REM: Sortie ZERO calculee et non specifiee dans l'interface
          %( integer N;
             [ N ]real COEFF )
          { ? real E
            ! real PRSC }
          %** REM: Instance de ZERO creee
          %(| %** ERR (ref 1): Dimensions des operandes incompatibles
              ** ERR (ref 2): Dimensions des operandes incompatibles
              %WE:= E $1 window (N+1)
            | %** REM: Signal ou parametre non declare
              %ZERO:= 0.0 e 0
            |
              array I to N
              of PS:= PS+( WE[ ( N+1 )-I ]*COEFF[ I ])
              with PS[0]:ZERO
              end
            | PRSC:= PS[ N ]
            |)
          where
              %** ERR: cf message No 1
              ** ERR: cf message No 2
              %[ N ]real WE init[ { to N }: 0.0 e 0 ], PS
      end
  end
end
```

### A-1.3 Résultat de la compilation

En cas de réussite des étapes précédentes et si l'option tra a été demandée, un programme équivalent au programme **EXEMPLE** est produit dans le fichier EXEMPLE_TRA.SIG. Le programme obtenu possède la même interface que le programme **EXEMPLE**. Il possède un corps qui a la structure suivante :

- il contient une composition de processus appelés ici **processus-horloge** ; chaque **processus-horloge** est constitué
  - d'une définition d'horloge `H_i_H` := $E_i$
  - d'une instruction de synchronisation
    `synchro{H_i_H, S_1,..., X_3}`
    donnant la liste `S_1,..., X_3` des signaux externes ou locaux synchrones à `H_i_H`
  - d'un appel à un processus de même nom `H_i_H` ;

- il contient les déclarations de signaux locaux communiqués entre ses **processus-horloge** ;
- il contient les déclarations des modèles de processus locaux H_i_H appelés dans son expression de processus ;
- **Anomalie.**
  *La présence de paramètres tableaux rend syntaxiquement incorrect le programme produit par le compilateur*

**Chaque modèle** H_i_H contient, selon la même structure :

- l'ensemble des signaux (associés à leur traitement) qui possèdent une horloge **calculable dans** H_i_H ;
- l'ensemble de **processus-horloge calculables dans** H_i_H qui ne sont pas **calculables dans** une horloge elle-même **calculable dans** H.

**Une horloge est calculable dans** H si elle s'exprime comme une fonction de signaux booléens, synchrones de H_i_H, où dont l'horloge est (récursivement) **calculable dans** H

**L'échec du calcul d'horloge** résulte en un ensemble comprenant plus d'une horloge non **calculables dans** d'autres horloges. Si des contraintes sont néanmoins exprimées sur ces horloges, elles apparaissent sous la forme d'instructions de synchronisation entre horloges ; dans ce cas, la génération de code FORTRAN n'a pas lieu.

### A-1.4 Echec à la génération de code

En dehors du cas évoqué ci-dessus, la production de code FORTRAN ou C (si elle est demandée) peut échouer en raison d'un cycle dans les dépendances de données ou d'horloges. Cet échec se traduit par l'un des deux messages suivant à l'écran :

```
** ERR:Cycle de dependance dans le graphe

** ERR:Contrainte(s) sur horloge(s)
```

Le fichier EXEMPLE_LIS.SIG contient la liste des signaux (ou des horloges), ainsi que leurs expressions de définition, à partir desquels la génération est devenue impossible ; ce sont donc les signaux (ou les horloges) qui appartiennent au cycle.
Exemple d'un cycle sur les signaux :

```
process CYCLE_SIG=
     { ? integer a
       ! integer s }
        (s := a + s)@s
end
```

Le fichier CYCLE_SIG_LIS.SIG contient :

```
process CYCLE_SIG_LIS=
     { ?
       ! }
```

```
     (| CYCLE_SIG( )
      | (| S_2:= ( A_1+S_2 )when H_6_H
         | ^output{ }
         |)
      |)
     where
     process CYCLE_SIG=
          { ? integer A
            ! integer S }
            ! integer S }
          (S:= A+S)@ S
     end
end
```

Exemple d'un cycle sur les horloges :

```
process CYCLE_HORL=
     { ? event OK
       ! integer N }
     (| ZN:= N $1
      | N:= ( ZN+1 )when OK
      |)
     where
          integer ZN init 0
end
```

Le fichier CYCLE_HORL_LIS.SIG contient :

```
process CYCLE_HORL_LIS=
     { ?
       ! }
     (| CYCLE_HORL( )
      | (| (H_7_H:= H_6_H when H_7_H)@ H_7_H
         |)
      |)
     where
     process CYCLE_HORL=
          { ? event OK
            ! integer N }
          (| ZN:= N $1
           | N:= ( ZN+1 )when OK
           |)
          where
               integer ZN init 0
     end
end
```

### A-1.5 Résultat de la génération de code FORTRAN

La génération de code produit les fichiers suivants :

- EXEMPLE_S.f contient un sous-programme SEXEM formé des constituants suivants :
  - les déclarations locales FORTRAN correspondant aux signaux internes
  - un point d'entrée IEXEM appelé à l'initialisation de l'exécution
  - un point d'entrée CEXEM appelé à chaque instant logique avec un paramètre horloge qui provoque l'arrêt de l'exécution lorsqu'il prend la **faux**
  - les appels aux fonctions externes réalisés au moyen de l'instruction FORTRAN CALL
- EXEMPLE_E.f contient
  - une procédure BEGIO d'ouverture des fichiers d'entrée-sortie
  - une procédure RSIG de lecture des données du signal SIG dans le fichier RSIG ; en fin de lecture, l'horloge d'arrêt d'exécution est rendue avec la valeur **faux**
  - une procédure WSIG d'écriture des valeurs calculées du signal SIG, dans le fichier WSIG
  - éventuellement une procédure WH par horloge fantôme H synthétisée par la compilation
  - une procédure ENDIO d'ouverture des fichiers d'entrée-sortie
- EXEMPLE_M.f contient
  - l'appel à l'initialisation BEGIO des fichiers d'entrée sortie
  - l'appel à l'initialisation IEXEM
  - une itération sur l'appel à CEXEM jusqu'à obtenir la valeur **faux** au paramètre horloge
  - l'appel à la terminaison ENDIO des fichiers d'entrée sortie

### A-1.6 Résultat de la génération de code C

La génération de code produit les fichiers suivants :

- EXEMPLE_S.c contient :
  - les déclarations correspondant aux signaux
  - une fonction iexem appelée à l'initialisation de l'exécution
  - une fonction cexem appelée à chaque instant logique tant qu'elle rend la valeur **vrai** et qui provoque l'arrêt de l'exécution lorsqu'elle rend la valeur **faux**
- EXEMPLE_E.c contient
  - une procédure begio d'ouverture des fichiers d'entrée-sortie
  - une procédure rsig de lecture des données du signal sig dans le fichier RSIG ; en fin de lecture, l'horloge d'arrêt d'exécution est rendue avec la valeur **faux**
  - une procédure wsig d'écriture des valeurs calculées du signal sig, dans le fichier WSIG
  - éventuellement une procédure wh par horloge fantôme h synthétisée par la compilation
  - une procédure endio d'ouverture des fichiers d'entrée-sortie
- EXEMPLE_M.c contient
  - l'appel à l'initialisation begio des fichiers d'entrée sortie
  - l'appel à l'initialisation iexem
  - une itération sur l'appel à cexem jusqu'à ce que le résultat de la fonction soit **faux**
  - l'appel à la terminaison endio des fichiers d'entrée sortie

## A-2 Un exemple complet : Filtrage récursif

Nous présentons ici un programme de filtrage récursif écrit en SIGNAL et les différents produits de la compilation

### A-2.1 Le programme SIGNAL

```
process FILTRE_REC_2 =
   ( integer  N, M ;
     real  B0 ;
      [N − 1] real  B ;
      [M] real  A )
   { ? real  E
     ! real  S
   }
   (| S :=  (E * B0) +
         PRODSCAL (N − 1, B) {E}
         +
         PRODSCAL (M, A) { S}
    |) @ S
   where
   process PRODSCAL =
      ( integer  N ;
        [N] real  COEFF )
      { ? real  E
        ! real  PRSC
      }
      (| WE := E $ 1 window N
       | ZERO := 0 . 0e 0
       | array I to N
           of  PS := PS + (WE [ (N + 1) − I] * COEFF [I] )
           with PS [0] : ZERO
         end
       | PRSC := PS [N]
       |)
      where
        [N] real  WE init [ { to N} : 0 . 0e0] , PS ;
       real  ZERO
   end
end
```

### A-2.2 Les paramètres

```
[OL_E_SIGN]
4,3,3.0,[[1]:1.0,[2]:2.0,[3]:3.0],[[1]:3.0,[2]:2.0,[3]:1.0]
```

### A-2.3 Le programme SIGNAL produit par le compilateur

```
process FILTRE_REC_2_TRA =
   { ? real  E_6
     ! real  S_7
   }
   (| (| H_9 _H := event E_6
      | synchro {H_9_H, S_ 7}
      | H_9_H ()
      |)
    |)
   where event  H_9_H
   process H _9_H =
      { ? event  H_9_H ;
          real  E_6
        ! real  S_7
      }
      (| synchro {H_9_H, PRSC_8, WE_14, PS_15, ZERO_16 , PRSC_18, WE_23, PS_24,
          ZERO_25}
       | (| WE_23 :=  S_7 $ 1 window 3
         | WE_14 := E_6 $ 1 window (4 - 1)
         | ZERO_25 := 0 . 0 e0 when H_9_H
         | array I_26 to 3
            of PS_24 :=  (PS_24 + (WE_23 [ (3 + 1 ) - I_26] *
                  [ [ 1 ]: 3.0 e 0,[ 2 ]: 2.0 e 0,[ 3 ]: 1.0 e 0 ] [I_26] ) ) when H _9_H
            with PS_24 [0] : ZERO _25
           end
         | PRSC_18 := PS_24 [ 3] when H_9_H
         | ZERO_16 := 0 . 0 e0 when H_9_H
         | array I_17 to 4 - 1
            of PS_15 :=  (PS_15 + (WE_14 [ ( (4 - 1) + 1) - I_ 17] *
                  [ [ 1 ]: 1.0 e 0,[ 2 ]: 2.0 e 0,[ 3 ]: 3.0 e 0 ] [I_17] ) ) when H_9_H
            with PS_15 [0] : ZERO_16
           end
         | PRSC_8 := PS_15 [ 4 - 1] when H_9_H
         | S_7 :=  ( (E_ 6 * 3 . 0e0) + PRSC_8 + PRSC_18) when H_9_H
         |)
       |)
      where
       real  PRSC_8, ZERO_16 , PRSC_18, ZERO_25 ;
       [3] real  WE_14 init [ {XZX_11 to 4 - 1} : 0 . 0e0 ] ;
       [3] real  PS_15 ;
       [3] real  WE_23 init [ {XZX_20 to 3 } : 0 . 0e0] ;
       [3] real  PS_24
   end
end
```

## A-2.4 Les programmes FORTRAN produits par le compilateur

### A-2.4 A Le programme principal

```
C
      LOGICAL H
      H = .TRUE.
      CALL BEGIO
      CALL IFILT
1      CALL CFILT(H)
      IF(H) GOTO 1
      CALL ENDIO
      END
```

### A-2.4 B Le sous-programme d'entrée-sortie

```
C
      SUBROUTINE EFILT
      LOGICAL H4H,H9H
      INTEGER IE6
      REAL TE6(0:3)
      INTEGER IS7
      REAL TS7(0:3),PRSC8
      INTEGER XZX11,IWE14
      REAL PS15(0:4),ZER16
      INTEGER I17
      REAL PRS18
      INTEGER XZX20,IWE23
      REAL PS24(0:4),ZER25
      INTEGER I26
C Initialisations des Entrees-Sorties
      ENTRY BEGIO
      OPEN (10,FILE = 'RE ',STATUS = 'OLD ')
      REWIND (10)
      OPEN (11,FILE = 'WS',STATUS = 'NEW ')
      REWIND (11)
      RETURN
C Fermeture des fichiers d'entrees-sorties
      ENTRY ENDIO
      CLOSE (10)
      CLOSE (11)
      RETURN
C Lecture de l'entree   : E
      ENTRY RE(TE6,IE6,H4H)
      READ (10,*,END = 1) TE6(IE6)
      RETURN
C Ecriture de la sortie : S
      ENTRY WS(TS7,IS7)
      WRITE (11,*) TS7(IS7)
```

```
      RETURN
1     H4H= .FALSE.
      END
```

### A-2.4 C Le sous-programme de traitement

```
C
      SUBROUTINE SFILT
C  Declaration des signaux
      LOGICAL H4H
      REAL A5(1:3),B4(1:3)
      INTEGER IE6
      REAL TE6(0:3)
      INTEGER IS7
      REAL TS7(0:3),PRSC8
      INTEGER XZX11,IWE14
      REAL PS15(0:4),ZER16
      INTEGER I17
      REAL PRS18
      INTEGER XZX20,IWE23
      REAL PS24(0:4),ZER25
      INTEGER I26
C  Declaration des horloges
      LOGICAL H9H
C  Corps de la procedure d'initialisations
      ENTRY IFILT
      A5(1)=3.0E0
      A5(2)=2.0E0
      A5(3)=1.0E0
      B4(1)=1.0E0
      B4(2)=2.0E0
      B4(3)=3.0E0
      IE6=2
      IS7=2
      IWE14=0
      DO  1 XZX11=1,3
1     TE6(MOD(IWE14+ XZX11- 1,4))=0.0E0
      IWE14=3
      IWE23=0
      DO  2 XZX20=1,3
2     TS7(MOD(IWE23+ XZX20- 1,4))=0.0E0
      IWE23=3
      ZER16=0.0E0
      ZER25=0.0E0
      RETURN
C  Corps du programme
      ENTRY CFILT(H4H)
      H9H= .TRUE.
```

```
      IF (IE6 .EQ. 3)THEN
      IE6=0
      ELSE
      IE6=IE6+ 1
      ENDIF
      CALL RE(TE6,IE6,H4H)
      IF ( .NOT. (H4H))RETURN
      IF (IWE14 .EQ. 3)THEN
      IWE14=0
      ELSE
      IWE14=IWE14+ 1
      ENDIF
      PS15(0)=0.0E0
      DO  3 I17=1,3
      PS15(I17)=(PS15(I17- 1))+ ((TE6(MOD(IWE14+ ((3)+ 1)- (I17)- 1,4)))
     &* (B4(I17)))
3     CONTINUE
      PRSC8=PS15(3)
      IF (IWE23 .EQ. 3)THEN
      IWE23=0
      ELSE
      IWE23=IWE23+ 1
      ENDIF
      PS24(0)=0.0E0
      DO  4 I26=1,3
      PS24(I26)=(PS24(I26- 1))+ ((TS7(MOD(IWE23+ ((3)+ 1)- (I26)- 1,4)))
     &* (A5(I26)))
4     CONTINUE
      PRS18=PS24(3)
      IF (IS7 .EQ. 3)THEN
      IS7=0
      ELSE
      IS7=IS7+ 1
      ENDIF
      TS7(IS7)=((TE6(IE6))* (3.0E0))+ (PRSC8)+ (PRS18)
      CALL WS(TS7,IS7)
      RETURN
      END
```

## A-2.5 Les programmes C produits par le compilateur

### A-2.5 A Le programme principal

```
typedef int event;
typedef int logical;

#define TRUE 1
```

```
#define FALSE 0

extern logical cfiltre_rec_2();
extern void ifiltre_rec_2();
extern void begio();
extern void endio();

extern int main()
{
    begio();
    ifiltre_rec_2();

    while(cfiltre_rec_2());
    endio();
}
```

### A-2.5 B Le sous-programme d'entrée-sortie

```
#include <stdio.h>

typedef int event;
typedef int logical;

#define TRUE 1
#define FALSE 0

FILE
     *fre_6,
     *fws_7;

int i01, i02, i03, i04, i05, i06, i07, i08, i09, i010;

extern void begio()
{
    fre_6 = fopen("RE.dat","r");
    fws_7 = fopen("WS.dat","w");
}

extern void endio()
{
    fclose(fre_6);
    fclose(fws_7);
}

extern void re_6(te_6,ie_6,h_4_h)
float te_6[4];
int ie_6;
```

```
event *h_4_h;
{
    *h_4_h = (fscanf(fre_6,"%f",&te_6[ie_6])!=EOF);
}

extern void ws_7(ts_7,is_7)
float ts_7[4];
int is_7;
{
    fprintf(fws_7,"%f\n",ts_7[is_7]);
}
```

### A-2.5 C Le sous-programme de traitement

```
typedef int event;
typedef int logical;

#define TRUE 1
#define FALSE 0


extern void re_6();

extern void ws_7();

/*C  Declaration des signaux */

float a_5[4], b_4[4];

int ie_6;
float te_6[4];

int is_7;
float ts_7[4];

float prsc_8;
int xzx_11, iwe_14;
float ps_15[5], zero_16;
int i_17;
float prsc_18;
int xzx_20, iwe_23;
float ps_24[5], zero_25;
int i_26;

int i01, i02, i03, i04, i05, i06, i07, i08, i09, i010;

/*C  Declaration des horloges */
```

```
event h_9_h;

event h_4_h;


/*C  Corps de la procedure d'initialisations */

extern void ifiltre_rec_2()
{
    a_5[1] = 3.0e0;
    a_5[2] = 2.0e0;
    a_5[3] = 1.0e0;
    b_4[1] = 1.0e0;
    b_4[2] = 2.0e0;
    b_4[3] = 3.0e0;
    ie_6 = 2;
    is_7 = 2;
    iwe_14 = 0;
    for (xzx_11 = 1;xzx_11<=3;xzx_11++)
        te_6[(iwe_14+xzx_11-1)%4] = 0.0e0;
    iwe_14 = -1;
    iwe_23 = 0;
    for (xzx_20 = 1;xzx_20<=3;xzx_20++)
        ts_7[(iwe_23+xzx_20-1)%4] = 0.0e0;
    iwe_23 = -1;
    zero_16 = 0.0e0;
    zero_25 = 0.0e0;
}

/*C  Corps du programme */

extern logical cfiltre_rec_2()
{
    h_9_h = TRUE;
    ie_6 = (ie_6+1)%4;
    re_6(te_6,ie_6,&h_4_h);
    if (!h_4_h) return FALSE;

    iwe_14 = (iwe_14+1)%4;
    ps_15[0] = 0.0e0;
    for (i_17 = 1;i_17<=3;i_17++)
        {
        ps_15[i_17] = ps_15[i_17-1] + te_6[(iwe_14+((3 + 1) - i_17)-1)%4] * b_4[
            i_17];
        }
    prsc_8 = ps_15[3];
    iwe_23 = (iwe_23+1)%4;
```

```
    ps_24[0] = 0.0e0;
    for (i_26 = 1;i_26<=3;i_26++)
        {
        ps_24[i_26] = ps_24[i_26-1] + ts_7[(iwe_23+((3 + 1) - i_26)-1)%4] * a_5[
            i_26];
        }
    prsc_18 = ps_24[3];
    is_7 = (is_7+1)%4;
    ts_7[is_7] = (te_6[ie_6] * 3.0e0 + prsc_8) + prsc_18;
    ws_7(ts_7,is_7);


    return TRUE;
}
```

## A-3 Un exemple de hiérarchie

Nous présentons ici un programme de gestion de souris écrit en SIGNAL et le programme SIGNAL résultant de la compilation

### A-3.1 Le programme SIGNAL

```
process MOUSE =
   { ? event  CLICK, TOP
     ! integer  M
   }
   (| (| N :=  (0 when RESET) default (ZN + 1)
      | synchro {N, TOP}
      | ZN := N $ 1
      | RESET :=  when (ZN = 4)
      |)
    | (| X :=  ( 0 when RESET) default NEW_X
      | synchro {X, CLICK default RESET}
      | NEW_X :=
            MIN { 2, ZX + 1}
      | ZX := X $ 1
      |)
    | (| M :=  ( NEW_X when CLICK when RESET) default (X when RESET)
      |)
    |)
   where
     integer  X, NEW_X, ZX init 0, N, ZN init 0 ;
     event  RESET
   function MIN =
      { ? integer  U, V
        ! integer  V
      }
   end
end
```

### A-3.2 Le programme SIGNAL produit par le compilateur

```
process MOUSE_TRA =
   { ? event  CLICK_1, TOP_2
     ! integer  M_3
   }
   (| (| H_9 _H := event TOP_2
      | H_9_H ()
      |)
    | H_14_H := H_15 _H when ( ( not H_11 _H) default H_15_H)
    | (| H_15 _H := H_11_H default H_15 _H
              @ H_15_H
      | synchro {H_15_H, X _10, NEW_X_11}
      | H_15_H ()
      |)
    | H_16_H := event CLICK_1
    | H_22_H := H_11 _H when H_16_H
    | H_27_H := H_11 _H when ( ( not H_22 _H) default H_11_H)
    | (| (| H_29_H := H_11_H default H_16_H
         | synchro {H_15_H, H _29_H}
         |)
      |)
   |)
   where
     event  H_27_H, H _22_H, H_16_H, H_15_H, H_14_H, H_11_H, H_9_H ;
     integer  X_10, NEW_X_11
   function MIN =
      { ? integer  U, V
        ! integer  V
      }
   end;
   process H_15_H  =
      { ? event  H_15_H, H _14_H, H_11_H
        ! integer  X_10, NEW_X _11
      }
      (| synchro {H_15 _H, ZX_12, V_18, U_19}
       | (| ZX_12 :=  X_10 $ 1
          | X_10 :=  (0 when H_11_H) default (NEW _X_11 when H_14_H)
          | V_18 :=  (ZX_12 + 1) when H_15_H
          | U_19 := 2 when H_15_H
          | NEW_X_11 :=
                MIN {U_19, V_18}
          |)
       |)
      where integer  ZX_12 init 0, V_18, U_19
   end;
```

```
  process H_9_H =
    { ? event  H_27_H, H _22_H, H_9_H ;
        event  TOP_2 ;
        integer  X_ 10, NEW_X_11
      ! event  H_11_H ;
        integer  M_3
    }
    (| synchro {H_ 9_H, N_13, ZN_14}
     | H_8_H := H_9 _H when ( ( not H_11 _H) default H_9_H)
     | (| H_11 _H :=  when (ZN_14 = 4 )
       | synchro {H_11_H, M _3}
       | M_3 :=  (NEW_X _11 when H_22_H) default (X_10 when H_27_H)
       |)
     | (| ZN_14 :=  N_13 $ 1
       | N_13 :=  (0 when H_11_H) default ( ( ZN_14 + 1) when H_8 _H)
       |)
     |)
    where
      event  H_8_H ;
      integer  N_13, ZN_14 init 0
  end
end
```

# Annexe B

# Les messages du compilateur

## B-1 Principe

Les messages distillés par le compilateur sont soit des erreurs soit des remarques, préfixées respectivement par "** ERR:" et "** REM:". Ils se présentent sous l'une des formes suivantes :

- message simple, invariant tel que :
  "** ERR: constante interdite".

- message précisé par le nom de l'entité en cause (signal, paramètre, processus,...) tel que :
  "** ERR: Deux sorties de meme nom : TOTO"

- ensemble de trois messages concernant les erreurs dans le cadre d'une instanciation ou d'un appel de processus comme dans l'exemple suivant :

```
process P=
        { ? %** ERR: cf message No 1
            %logical B
          ! integer C}
        %** ERR (ref 1): Type de B incompatible avec son utilisation
        %C := PP {B}
        where
        process PP=
                { ? integer I
                  ! integer Z}
                %** ERR: Types des operandesincompatibles
                %Z := I**2
        end
end
```

  - le premier message est *accroché* à la déclaration du signal mis en cause.
  - le second est *accroché* à l'appel du processus.
  - le troisième est *accroché* à l'instruction dans le processus appelé où le signal est incorrectement utilisé.

Cette annexe fournit la liste des messages accompagnés de commentaires lorsque cela a été jugé nécessaire.

## B-2 Erreurs

### B-2.1 Analyse syntaxique

Les erreurs trouvées durant l'analyse syntaxique apparaissent sous la forme suivante :
`Erreur de Syntaxe` *code* `ligne` *n*
dans laquelle *n* est le numéro de la ligne erronée et *code* désigne la structure syntaxique erronée selon le codage ci-dessous :

- 10 : erreur dans la syntaxe d'un **MODELE**
- 13 : erreur dans la syntaxe d'une **INTERFACE** de modèle
- 14 : erreur dans la syntaxe d'une **P-EXPR** (expression sur processus)
- 16 : erreur dans la syntaxe d'une **S-DECLARATION** (déclaration de signaux)
- 54 : erreur dans la syntaxe d'une **S-EXPR** (expression sur signaux)

### B-2.2 Appel du compilateur

- **Nombre de parametres incorrect** : appel du compilateur avec un fichier de paramètres incorrect.

### B-2.3 Déclaration de MODELE

- **Processus XXX non declare** : processus ou fonction externe non declare.
- **Processus sans entree ni sortie**
- **Processus externe ayant des declarations de parametres** : un processus externe est une fonction sur signaux qui ne doit posséder aucun paramètre.
- **Parametre non declare** : un identificateur a été déterminé comme étant un paramètre et n'est pas déclaré.
- **Nb. d'entrees incompatible avec la declaration**
- **Entree XXX utilisee non specifiee dans l'interface**
- **Nb. de sorties de l'appel different du nb. de signaux definis**
- **Sortie XXX specifiee dans l'interface et non calculee**

### B-2.4 S-DECLARATION

- **Indice et Signal de meme nom, XXX.**
- **Double declaration de XXX**

### B-2.5 P-EXPR

♠ **Deux sorties de meme nom : XXX**

♠ **Mutation incorrecte de boucle** : indiçage de sens opposé de signaux dans un *motif* (*X* et *Y* dans l'exemple suivant).

**exemple :**

```
array I to N of
  ....
with X[.]:B, Y[0]:A end
```

♠ **Fermeture sur entree sans sortie** : Opération de fermeture sur une entrée sans sortie de même nom, comme dans l'exemple suivant :

```
(C := B*2)@ B
```

### B-2.6 S-EXPR

♠ **Conflit sur identificateurs** : utilisation d'un signal, (resp. d'un paramètre ou d'un indice), dans un contexte où un signal (resp. un paramètre ou un indice) est interdit.

♠ **Expression ordonnee necessaire** : dans le contexte d'une définition multiple, le membre de droite doit être un appel.

### B-2.7 Dimensions

♠ **Dimension de XXX incompatible avec son utilisation**

♠ **Dimensions des operandes incompatibles** : par exemple, la somme de 2 tableaux n'ayant pas la même dimension.

♠ **Chaque operande de la concatenation doit etre un vecteur** : seul un **Nom-***vecteur* est autorisé comme argument d'une concaténation.

♠ **Operandes scalaires attendus pour l'exp. de relat.** : les tableaux ne sont pas acceptés dans les expressions de relation.

### B-2.8 Types

♠ **Type de XXX incompatible avec son utilisation**

♠ **Types des operandes incompatibles**

♠ **XXX, utilise dans le motif, n'est pas instancie**

### B-2.9 Constantes entières

♠ **Valeur de la dim. courante de XXX <> borne sup. du motif**

♠ **Expression entiere strictement positive attendue**

♠ **Bornes du parcours sur XXX incoherentes avec le pas**

♠ **Pas du parcours sur XXX egal a zero**

♠ **Type entier attendu**

### B-2.10 Horloges

♠ **Constante interdite** : Le contexte courant n'autorise pas l'utilisation d'une constante.

♠ **L'horloge de l'expression est indefinissable** : Utilisation d'une constante dans un contexte où aucune horloge ne peut lui être associée, par exemple : X:= 3 cell C.

### B-2.11 Divers

♠ **Erreur** : Erreur causée par une division par zéro
ou
par un cycle dans les instanciations des processus.

## B-3 Remarques

### B-3.1 Déclaration de MODELE

◊ **Fonction externe sans entree**

◊ **Entree XXX specifiee dans l'interface et non utilisee**

◊ **Sortie XXX calculee et non specifiee dans l'interface**

### B-3.2 S-DECLARATION

◊ **Nom identique pour signal de sortie et signal local (local ignore)**

◊ **Signal ou parametre non declare (parametre ignore)**

### B-3.3 P-EXPR

◊ **Sortie inexistante pour une restriction** : masquage d'une sortie inexistante.

◊ **Double modification d'une entree** : renommage dans un *motif* du style : X[.]:X.

### B-3.4 S-EXPR

◊ **Operateur sans effet** : par exemple, P!A:B où A n'est pas une sortie de P.

### B-3.5 Dimensions

◊ **Un indice de XXX n'est pas controlable** : un indice de tableau est un signal ou une expression arithmétique dont on ne peut déterminer les bornes.

◊ **Parcours sur XXX incontrolable** : expression de parcours dont on ne peut déterminer les bornes.

◊ **Element vectoriel non controlable** : un indice d'un élément vectoriel est soit un signal soit une expression arithmétique dont on ne peut déterminer les bornes.

### B-3.6 Divers

◊ **Les 2 operandes sont non types (controle impossible)**

◊ **Initialisation de XXX non prise en compte** : initialisation de XXX inutile.

◊ **XXX devrait etre initialise** : XXX est le résultat d'une expression de retard ($ ou window) ou d'un cell.

◊ **Instance de XXX creee** : un signal est utilisé et non déclaré.

# Annexe C

# Grammaire SIGNAL

## C-1 Les équations

### C-1.1 Les expressions sur équations

Comme premier argument des opérateurs de profil (?,!,/,!!,@), une définition de signal (:=) doit être encadrée de parenthèses.

- **Syntaxe Hors Contexte. P-EXPR**

  ♠ **P-EXPR** ::=

  - Nom-*signal* [:=] **S-EXPR** •
  - [{] { Nom-*signal* [,] ... } [}] [:=] **S-EXPR** •
  - Nom-*modèle* [(] [ { **S-EXPR** [,] ... } ] [)] •
  - Nom-*modèle* [(] [ { **S-EXPR** [,] ... } ] [)] [{] [ { **S-EXPR** [,] ... } ] [}] •
  - Nom-*modèle* [{] [ { **S-EXPR** [,] ... } ] [}] •
  - [synchro] [{] { **S-EXPR** [,] ... } [}] •
  - [(] **P-EXPR** [)] •
  - [(|] { **P-EXPR** [|] ... } [|)] •
  - **P-EXPR** [?] { Nom-*signal* [:] Nom-*signal* [,] ... } •
  - **P-EXPR** [!] { Nom-*signal* [:] Nom-*signal* [,] ... } •
  - **P-EXPR** [/] { Nom-*signal* [,] ... } •
  - **P-EXPR** [!!] { Nom-*signal* [,] ... } •
  - **P-EXPR** [@] { Nom-*signal* [,] ... } •
  - [array] **I-MOTIF** [of] **S-EXPR** [ [with] { **CONNEXION** [,] ... } ] [end] •

  ♠ **I-MOTIF** ::=

  - Nom-*indice* [to] **S-EXPR** •

  ♠ **CONNEXION** ::=

  - Nom-*signal* •
  - Nom-*signal* [[0]:] Nom-*signal* •
  - Nom-*signal* [[.]:] Nom-*signal* •

### C-1.2 Déclaration de modèles d'équations

• **Syntaxe Hors Contexte.** MODELE

♠ **MODELE** ::=

• [function] Nom-*modèle* [=] **INTERFACE** •
• [process] Nom-*modèle* [=] **INTERFACE P-EXPR**
[ [where] [ { **S-DECLARATION** [;] ... } ] [ { **MODELE** [;] ... } ] ] [end] •

### C-1.3 Interface d'un modèle d'un modèle d'équations

• **Syntaxe Hors Contexte.** INTERFACE

♠ **INTERFACE** ::=

• [ [(] [ { **S-DECLARATION** [;] ... } ] [)] ]
[{] [?] [ { **S-DECLARATION** [;] ... } ]
[!] [ { **S-DECLARATION** [;] ... } ] [}] •

## C-2 Les signaux

### C-2.1 Les déclarations de signaux et paramètres

• **Syntaxe Hors Contexte.** S-DECLARATION

♠ **S-DECLARATION** ::=

• { Nom-*signal* [ [init] **S-EXPR** ] [,] ... } •
• **TYPE-SIGNAL** { Nom-*signal* [ [init] **S-EXPR** ] [,] ... } •

♠ **TYPE-SIGNAL** ::=

• [event] •
• **Type-élément** •
• [[] { **S-EXPR** [,] ... } []] **Type-élément** •

♠ **Type-élément** ::=

• [logical] •
• **Type-numérique** •

♠ **Type-numérique** ::=

• [integer] •
• [real] •
• [dpreal] •
• [complex] •

♠ **Type-scalaire** ::=

• **Type-synchronisation** •
• **Type-numérique** •

♠ **Type-synchronisation** ::=

- [event] •
- [logical] •

## C-2.2 Les expressions sur signaux

Une expression élémentaire (S-EXPR-ELEMENTAIRE) peut être argument d'une expression scalaire (S-EXPR-SCALAIRE) ou d'une expression sur tableau (S-EXPR-TABLEAU) sans être encadrée de parenthèses.

• **Syntaxe Hors Contexte. S-EXPR**

♠ **S-EXPR** ::=

- **S-EXPR-ELEMENTAIRE** •
- **S-EXPR-SCALAIRE** •
- **S-EXPR-TABLEAU** •

♠ **S-EXPR-ELEMENTAIRE** ::=

- **Nom-*signal*** [ [ ] { **S-EXPR** [,] ... } [ ] ] •
- **Nom-*modèle*** [ [(] [ { **S-EXPR** [,] ... } ] [)] ]
  [{] [ { **S-EXPR** [,] ... } ] [}] •
- [(] **S-EXPR** [)] •

### C-2.2 A Les expressions de forme scalaire

Elles apparaissent dans l'ordre des priorités décroissantes sauf en ce qui concerne les expressions dynamiques qui ne peuvent être argument direct des opérateurs arithmétiques unaires et qui admettent en second argument entier ces mêmes opérateurs.

• **Syntaxe Hors Contexte. S-EXPR-SCALAIRE**

♠ **S-EXPR-SCALAIRE** ::=

- **S-EXPR-DYNAMIQUE** •
- **S-EXPR-ARITHMETIQUE** •
- **S-EXPR-BOOLEENNE** •
- **S-EXPR-TEMPORELLE** •

♠ **S-EXPR-DYNAMIQUE** ::=

- **Nom-*signal*** [$] **S-EXPR** •
- **Nom-*signal*** [ [$] **S-EXPR** ] [window] **S-EXPR** •

♠ S-EXPR-ARITHMETIQUE ::=

- Cst-entière •
- CST-REELLE •
- CST-COMPLEXE •
- [+] S-EXPR •
- [−] S-EXPR •
- S-EXPR [*] S-EXPR •
- S-EXPR [/] S-EXPR •
- S-EXPR [**] S-EXPR •
- S-EXPR [+] S-EXPR •
- S-EXPR [−] S-EXPR •

♠ S-EXPR-BOOLEENNE ::=

- [true] •
- [false] •
- S-EXPR [=] S-EXPR •
- S-EXPR [/=] S-EXPR •
- S-EXPR [>] S-EXPR •
- S-EXPR [>=] S-EXPR •
- S-EXPR [<] S-EXPR •
- S-EXPR [=] S-EXPR •
- [not] S-EXPR •
- S-EXPR [or] S-EXPR •
- S-EXPR [and] S-EXPR •

♠ S-EXPR-TEMPORELLE ::=

- [#] S-EXPR [ [from] S-EXPR ] •
- [#] S-EXPR [after] S-EXPR •
- S-EXPR [default] S-EXPR •
- S-EXPR [when] S-EXPR •
- S-EXPR [cell] S-EXPR •
- [event] S-EXPR •
- [when] S-EXPR •

### C-2.2 B Les expressions sur tableaux

• **Syntaxe Hors Contexte. S-EXPR-TABLEAU**

♠ S-EXPR-TABLEAU ::=

- { Nom-*signal* [||] ... } •
- [[] { ITERATION [,] ... } []] •

♠ **ITERATION** ::=

- [ **[** ] { **S-EXPR** [ **,** ] ... } [ **]** ] [ **:** ] **S-EXPR** •
- [ **{** ] { [ **Nom-*indice*** ] [ [ **in** ] **S-EXPR** ] [ [ **to** ] **S-EXPR** ] [ [ **step** ] **S-EXPR** ] [ **,** ] ... }
  [ **}** ] [ **:** ] [ [ **[** ] { **S-EXPR** [ **,** ] ... } [ **]** ] ] [ **:** ] **S-EXPR** •

# Index

# Références


[1] A. Benveniste and P. Le Guernic. Hybrid dynamical systems theory and the signal language. *IEEE transactions on Automatic Control*, 35(5):535–546, May 1990.

[2] A. Benveniste, P. Le Guernic, and C. Jacquemot. *The* **langage** SIGNAL *software environment for real-time system specification,design, and implementation.* Internal report 490, IRISA, Rennes, September 1989.

[3] P. Le Guernic and T. Gautier. Data-flow to von NEUMANN: the **langage** SIGNAL approach. In J. L. Gaudiot and L. Bic, editors, *Advanced topics in data-flow computing*, Prentice-Hall, 1990.

# LISTE DES DERNIERES PUBLICATIONS INTERNES IRISA

## 1991

**PI 570** DESIGN DECISION FOR THE FTM : A GENERAL PURPOSE FAUTL TOLERANT MACHINE
Michel BANATRE, Gilles MULLER, Bruno ROCHAT, Patrick SANCHEZ
Janvier 1991, 30 pages

**PI 571** ANIMATION CONTROLEE PAR LA DYNAMIQUE
Georges DUMONT, Marie-Paule GASCUEL, Anne VERROUST
Février 1991, 84 pages

**PI 572** MULTIGRID MOTION ESTIMATION ON PYRAMIDAL REPRESENTATIONS FOR IMAGE SEQUENCE CODING
Nadia BAAZIZ, Claude LABIT
Février 1991, 48 pages

**PI 573** A SURVEY OF TREE-TRANDUCTIONS
Jean-Claude RAOULT
Février 1991, 18 pages

**PI 574** THE OPTIMAL ADAPTATIVE CONTROL USING RECURSIVE IDENTIFICATION
Anatolij B. JUDITSKY
Février 1991 - 26 pages

**PI 575** MANUEL SIGNAL
Patricia BOURNAI, Bruno CHERON, Bernard HOUSSAIS,
Paul LE GUERNIC
Février 1991, 84 pages

Imprimé en France
par
. l'Institut National de Recherche en Informatique et en Automatique