IRIA

CENTRE DE RENNES

IRISA

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél. (3) 954 90 20

Rapports de Recherche

N° 206

SIGNAL:
UN LANGAGE POUR LE TRAITEMENT DU SIGNAL

Paul LE GUERNIC
Albert BENVENISTE
Thierry GAUTIER

Mai 1983

SIGNAL : UN LANGAGE POUR LE TRAITEMENT DU SIGNAL

Paul LE GUERNIC, Albert BENVENISTE, Thierry GAUTIER

Publication Interne n° 195
49 pages
Mars 1983

RESUME : SIGNAL est un langage en cours de définition à l'IRISA dans le cadre d'un projet d'aide à la conception d'algorithmes en traitement temps réel du signal. Après avoir caractérisé la classe d'algorithmes visée, nous décrivons succintement les tâches d'un système de CAO dans lequel ce langage devrait être intégré. Les caractéristiques générales du langage sont inspirées des principes flots de données : un programme SIGNAL décrit un réseau d'opérateurs (arithmétiques ou temporels) interconnectés à travers des ports orientés, sur lesquels circulent des signaux. Ces réseaux sont construits progressivement à l'aide d'expressions sur filtres permettant d'interconnecter des ports selon un principe s'appuyant sur une identité de leurs noms. Différents opérateurs sont proposés : des opérateurs binaires permettent des interconnexions effectives selon divers modes, des opérateurs unaires agissent sur les noms de ces ports. La présentation est illustrée par la description d'un égaliseur récursif avec auto-orthogonalisation partielle.

SUMMARY :

SIGNAL is a language (which definition is in progress at IRISA), intended to be the algorithms description language of a CAD system for real time signal processing applications. Salient features of target applications are first presented, followed by a (very) brief presentation of the hypothetical CAD system tasks. Main characteristics of SIGNAL are issued from data flow principles : a program (a filter) describes an oriented graph which opérator nodes (namely, arithmetical and temporal primitive functions) are connected by the way of ports (or data nodes). A filter is progressively constructed by means of filters expressions involving two kinds of operators : unary operators allow port names processing, where as binary ones yield oriented graphs in which identically named ports of operands are connected (depending upon the operator being currently used). This presentation is illustrated by the way of a (partially) self orthogonalizing recursive equalizer in ladder form, treated as an example.

PREAMBULE.

Cette note présente une première version d'un langage de description d'algorithmes constituant l'un des éléments d'un projet de développement d'outils d'aide à la conception de machines pour le traitement du signal. Ce projet, mené conjointement à l'IRISA et à l'INRIA-Rocquencourt*, a pour motivation d'une part les besoins croissants en algorithmique traitement du signal et d'autre part l'apparition de processeurs rapides permettant de mettre en oeuvre des fonctions complexes :

- Le traitement du signal à des débits moyens, élevés, ou très élevés, est appelé à se développer fortement dans les domaines suivants des Télécommunications : traitement de la parole (codage, analyse, synthèse), annulation d'échos (électriques et acoustiques), modems, et, ultérieurement codage d'images bas débit/basse qualité ; on peut également prévoir son développement dans les domaines de la régulation et de la commande de système dynamique (avionique, robotique).

- L'apparition de processeurs programmables intégrés orientés traitement du signal permet d'envisager pour l'avenir une implantation dans de bonnes conditions de fonctions "Traitement du Signal", et ce d'autant plus qu'une chaîne de CAO bien adaptée sera disponible. Dans cette optique, les délais actuels d'implantation sur microprocesseurs, ou de définition d'architectures prototypes, qui sont de plusieurs hommes x années, devraient pouvoir se ramener raisonnablement à plusieurs mois, voire moins, pour les applications simples.

Les processus considérés possèdent en général une structure interne et un jeu d'instructions permettant d'optimiser le calcul du produit de convolution. Cependant la taille de leurs mémoires de données et de programmes est faible, les possibilités de couplages forts entre boîtiers du même type restent très limitées ; on rencontre là, les principaux obstacles à la programmation (au mieux en assembleur) d'applications nécessitant plusieurs de ces composants.

Un système de CAO permettrait d'implanter dans de bonnes conditions, des fonctions complexes en traitement du signal pour que les modules qui le composent permettent au pire de guider l'opérateur dans ses choix de répartition

---

* Ce projet fait l'objet d'une convention avec la DAII.

c- Les algorithmes à structure irrégulière.

Cette dénomination volontairement floue est simplement là pour rappeler que les deux catégories précédentes ne recouvrent pas l'ensemble des besoins. Citons l'exemple de la plupart des procédés de codage par transformation [Tribolet 79, Crochière 82] dans le domaine du traitement de la parole : le signal y est consommé à la volée, puis traité par blocs ; mais la structure du traitement effectué sur chaque bloc est difficile à caractériser.

Nous allons maintenant nous attacher à décrire les caractéristiques essentielles de la classe d'algorithmes qui servira de référence pour la définition du langage. De la description de ces caractéristiques, il ressortira que sont visées par ordre de préférence les classes a-, b- et c- décrites précédemment.

Caractéristiques des algorithmes devant pouvoir être décrits par le langage.

Nous les indiquons par ordre d'importance décroissante.

(C-1) Ce sont des algorithmes séquentiels.

Du point de vue algorithmique, le point essentiel est la propriété suivante : il existe une unique boucle* de longueur non bornée, et toutes les autres boucles de longueur non bornée lui sont assujetties (sous-multiples de cette boucle, ou extraction d'événements "aléatoires" dont l'occurrence est rythmée par cette boucle) : nous interpréterons cette boucle comme l'horloge-temps. Toutes les autres boucles possèdent une longueur bornée.

Jouissent de cette propriété, évidemment les algorithmes de la classe a- (en incluant les détections d'événements tels que : changements brusques de caractéristiques du signal, ou pannes en commande), mais aussi les algorithmes de la classe b- puisque les boucles de calcul y ont une longueur caractérisée par l'algorithme, et ne s'arrêtent pas sur tests (encore qu'il soit toujours possible de borner a priori la longueur d'une boucle d'itération en calcul numérique, mais ce n'est guère satisfaisant).

---

* au sens informatique du terme.

(C-2) Les fonctions et processus qui coopèrent pour réaliser l'algorithme sont connus a priori, et peuvent être spécifiés de manière statique.

Ainsi les automaticiens et traiteurs de signal utilisent volontiers les blocs-diagrammes pour décrire le réseau statique dans lequel circulera le signal.

L'intérêt est ici d'arriver à ce que les noeuds de ce réseau statique soient assez simples pour que la description de ce réseau constitue une part importante de la description de l'algorithme.

Ici encore, la classe a- d'algorithmes mentionnée plus haut jouit de cette caractéristique. Mais, pour les algorithmes de la classe b-, on peut remarquer que, à l'intérieur des boucles de calcul numérique, le flot des calculs est en grande partie décrit par un réseau statique ("perfect shuffle" et papillons dans le cas de Fourier, et "cascade en treillis" dans le cas de la méthode de prédiction linéaire).

(C-3) Ces algorithmes offrent une large place au calcul numérique, et ne mettent en cause que rarement de l'analyse numérique matricielle ou vectorielle de grande dimension.

Il s'agit là d'une propriété bien reconnue en traitement du signal : la plupart des processeurs spécialisés sont justement bâtis autour de l'opération Σxy, opération essentielle dans le domaine du filtrage numérique classique. Il faut néanmoins signaler que les algorithmes adaptatifs nécessitent une variété d'opérations qui peut être bien plus grande (divisions, rotations dans le plan complexe, ...).

(C-4) Ces algorithmes présentent fréquemment des rebouclages de signal.

C'est par essence le cas pour les algorithmes de commande, qui travaillent en boucle fermée, mais c'est aussi le cas de tous les algorithmes adaptatifs [Falconer 80]. L'existence de ces rebouclages limite les possibilités de pipeline lors de l'implantation : ainsi, on peut dire qu'un algorithme reste invariant sous l'effet d'un pipe-line dans le cas où l'introduction de ce pipeline laisse globalement invariant chacun des flots de signaux qui circulent dans l'algorithme. Les rebouclages de signaux sont donc l'obstacle essentiel à l'introduction de pipe-line lors de l'implantation.

(C-5) L'effet de l'implantation en précision numérique limitée doit pouvoir être étudié sur ces algorithmes.

Il s'agit en fait, non pas d'une caractéristique des algorithmes, mais d'une conséquence de l'implantation sur "petits processeurs", qui correspond à la situation standard dans la période actuelle ; ce point perdra peu à peu de son importance avec la disponibilité de processeurs plus puissants. Le point à noter ici est que, dans bien des cas parmi lesquels les algorithmes adaptatifs, il est difficile, voire impossible, d'analyser a priori l'effet des erreurs d'arrondi et des mauvais cadrages, ce qui conduit alors à reporter une telle étude au niveau de la simulation.

I-2. Le type d'implantation visé.

Pour les raisons technologiques mentionnées dans le préambule, mais aussi en raison des propriétés des algorithmes qui nous intéressent, notre choix se porte vers le type d'implantation suivant :

- un réseau statique de "petits" processeurs interconnectés,
- avec prise en compte de la contrainte prioritaire de satisfaction du débit de calcul (contrainte "temps-réel").

Chacun de ces processeurs peut, soit être réalisé par un circuit physiquement autonome (cas des réseaux de processeurs orientés traitement du signal, mentionnés dans le préambule), soit être une partie d'un circuit plus important (cas des circuits VLSI à structure planaire régulière).

Bien entendu, un des problèmes clés de l'implantation (le problème de "l'allocation des ressources") sera de plonger le réseau statique associé à l'algorithme dans le réseau statique des processeurs, de façon que chaque noeud de ce second réseau soit un sous-réseau du premier.

Un des outils essentiels pour la satisfaction de la contrainte débit sera l'usage de pipe-lines (plutôt que l'utilisation du parallélisme vectoriel, ceci en raison de la structure des algorithmes), d'où l'importance de la propriété (C-4) et des commentaires qui l'accompagnent : le langage décrivant ces algorithmes devra permettre d'exprimer aussi clairement que possible les possibilités de pipe-line.

I-3. Présentation succincte de l'architecture de la CAO.

Les caractéristiques des classes d'algorithmes présentées ci-dessus suggèrent la définition d'une architecture d'un système de conception adapté d'une part à l'étude des propriétés structurelles des algorithmes et d'autre part à l'étude d'implantation sous contraintes d'environnement pouvant se révéler sévères.

Classiquement, la conception d'un produit informatique est introduite par une phase de spécification formelle dans laquelle la forme des traitements effectués est au maximum oblitérée ; dans l'idéal cette forme (le programme lui-même) devrait être obtenue par une suite de transformations systématiques conduisant d'une description formelle à une description algorithmique liée au langage de programmation utilisé (dont le choix est alors de moindre importance). Cette approche n'est pas directement transposable dans un domaine où la forme de l'algorithme (en raison de la caractéristique (C-5) et des contraintes temps-réel) est un paramètre essentiel de la conception. Dans cet esprit, des méthode formelles de synthèse de filtres ont été proposées telle SIRENA [Gerber 77, Le Baron 79] ; elles permettent de déterminer des coefficients pour une forme prédéfinie de la fonction de transfert associée, spécifiée par $H = \frac{N}{D}$; en effet l'ordre des calculs (N ; D ou D ; N ou divers entrelacements de N et D) a une influence sur les caractéristiques du filtre en raison des bruits de calcul. Il faut de plus noter que de tels outils ne peuvent exister dans le cas des algorithmes adaptatifs et ne sont d'aucune aide pour l'étude d'implantation sur des processeurs, la forme de l'algorithme pouvant ne pas convenir en raison des caractéristiques des processeurs. L'architecture du système de CAO que nous envisageons reflète cette propriété : Un processus de conception de logiciel (1) produit pour un processus de conception d'architecture (2), un graphe flot de données décrivant l'application traitement de signal considérée. Les graphes flots de données présentent des caractéristiques permettant d'étudier l'implantation parallèle des algorithmes.

I-31. Conception de logiciel : Le processus de conception de logiciel comporte lui-même deux processus : un processus de description de filtres et un processus d'étude de caractéristiques logiques.

a- Description de filtres : Un filtre peut être construit par assemblage de filtres existants dans un contexte initial dont le nom est fourni par l'utilisateur (bibliothèque) ou par redéfinition partielle d'un de ces filtres ou

enfin par une définition complète indépendante de tout contexte. Le langage présenté en II est le langage de description de filtres. Il s'agira au niveau de la CAO d'offrir une structure de dialogue permettant de réaliser les opérations ci-dessus ; à long terme un tel "dialogue" pourrait être graphique.

Le processus de description de filtre produit, pour chaque filtre construit, le réseau flot de données associé.

b- Etude de caractéristiques logiques : L'exécution d'un graphe flot de données peut être alors simulée par ce processus afin d'étudier son comportement et en particulier les bruits de calcul résultant d'arithmétiques à précision bornée. Il s'agira alors d'offrir des outils de trace, de modification au vol de coefficients, des interfaces graphiques, ...

I-32. Conception d'architecture (CA) : Ce processus doit permettre au concepteur d'une application de définir un (multi-)processeur de traitement de signal réalisant l'application décrite par le graphe flot de données, respectant des contraintes temps-réel, et utilisant un (ou plusieurs) processeur(s) programmable(s) orienté(s) traitement du signal. Le résultat doit donc être la description d'un ensemble de processeurs (codes et données associés à chacun d'eux) et de leurs interconnexions. Pour atteindre ce résultat ce processus de conception est lui-même composé de trois processus. Un processus de spécification de composant fournissant les descriptions formelles nécessaires aux deux autres processsus de CA : le processus de partition et le processus de compilation. Dans le premier, est réalisée une partition du graphe flot de données en fonction des contraintes temps-réel d'une part, des caractéristiques des processeurs d'autre part (temps de cycle, capacités des mémoires, ...) ; dans le second chacune des parties du graphe est compilée.

En l'état actuel de nos travaux, la présentation succincte ci-dessus constitue un schéma qui n'est ni complet ni (a fortiori) définitif.

II. PRESENTATION GENERALE DU LANGAGE "SIGNAL".

L'objet de ce rapport est de présenter l'état actuel (donc préliminaire) du langage de description des algorithmes. On en présentera d'abord les principes généraux, avec quelques indications sur les types envisagés, et les opérateurs correspondants. L'objet principal du rapport est l'étude détaillée des réseaux statiques associés à chacun des algorithmes : il s'agit d'être capable d'effectuer formellement la construction, la description d'un tel réseau, et d'en explorer les transformations possibles. Ce dernier point en particulier nécessite l'introduction d'un formalisme abstrait de manière à isoler les difficultés clés, sans s'embarrasser d'emblée de l'interprétation de ce formalisme. Ce point fera l'objet d'un rapport distinct dans lequel sera décrite une algèbre possédant toutes les propriétés souhaitées pour les réseaux statiques qui nous intéressent.

Nous attirons encore une fois l'attention du lecteur sur le fait que le langage SIGNAL est le langage de haut niveau qui devra, d'une part servir de point de départ à l'implantation , et d'autre part permettre le transport aisé d'un algorithme. Ce langage n'est pas à confondre avec le langage de commande de la CAO (qui sera lui seul vu de l'utilisateur lors de la mise au point d'un programme ou d'une implantation), dont la structure (langage question-réponse, éventuellement représentations graphiques, par exemple) et la syntaxe pourront être quelque peu différentes.

Le présent chapitre contient tout d'abord une description de caractéristiques générales du langage, résultant des propriétés de la classe d'applications visées (§ 1) ; elle est suivie d'une présentation des formalismes adoptés pour décrire les expressions de filtres ; on trouve enfin les caractéristiques externes des filtres.

II-1. Caractéristiques générales.

Les principes généraux que nous allons énoncer découlent naturellement des caractéristiques (C-1 à 5) relevées pour la classe d'algorithmes que nous avons définie.

II-11. SIGNAL s'inspire des principes flots de données : La recherche de l'efficacité dans une exploitation du parallélisme interne à un programme a conduit au développement, en marge des multiprocesseurs et des processeurs vectoriels, d'architectures flots de données pour lesquelles le séquencement

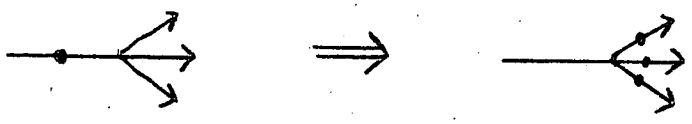
des instructions n'est plus dirigé par un compteur ordinal mais par le flot de données (voir en particulier [Treleaven 82] pour une étude générale). Parallèlement sont apparus les langages applicatifs, pour lesquels un programme n'est pas une suite d'instructions, mais un ensemble de définitions (LISP [Mc Carthy 66], LUCID [Ashcroft 77], FP [Backus 78]). A cette classe de langages appartiennent les "langages flots de données" [Ackerman 79] conçus pour le calcul parallèle, adaptés à des exécutions sur architectures flots de données ([Dennis 74], [Kahn 74], VAL [Mc Graw 82], CAJOLE [Hankin 81]) ; les langages à assignation unique (SAMPLE [Chamberlin 71], LAU [Comte 78]) reposent également sur les principes qui nous intéressent ici : un calcul peut être représenté par un graphe flot de données dans lequel les arcs représentent les chemins des données et les noeuds des opérations. Pour ces langages, différents modèles ont été proposés [Davis 82]. Notre objectif n'étant pas ici une étude de ces formalismes, nous nous contentons d'illustrer les principes flots de données à l'aide du graphisme introduit dans [Dennis 74].

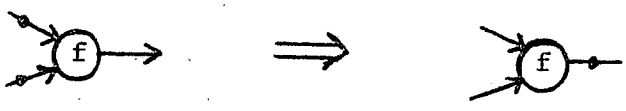
Un calcul est représenté par un graphe bi-parti orienté ; dans ce graphe un noeud peut être :

- une action (dotée de une ou plusieurs entrées et d'une sortie),
- une connexion (dotée d'une entrée et de une ou plusieurs sorties).

Les arcs du graphe décrivent les chemins de données entre les différents noeuds.

Chaque noeud du graphe est activé selon un principe illustré par les schémas suivants :

connexion 

action 

(a) (b)

. Un noeud peut être activé (a) si toutes ses entrées sont disponibles (présence d'un jeton sur chaque entrée), et si ses sorties précédentes ont été consommées (absence de jeton sur chaque sortie).

.. Le noeud est alors automatiquement activé : il consomme les valeurs disponibles à ses entrées et produit de nouvelles valeurs sur ses sorties (b).

Dans un langage flot de données, la notion de variable est proscrite, au profit de la manipulation de valeurs désignées par un nom. La contrainte d'association d'un nom à une valeur unique peut être tempérée par l'usage de la notion de bloc dans le langage et par la possibilité de masquer des noms hors de blocs, comme on le verra plus loin.

De ces caractéristiques découlent les deux propriétés essentielles suivantes, très intéressantes dans notre cas :

- Il est possible de déduire aisément la dépendance entre données de l'écriture des opérations dans le programme.

- Le séquencement dans le programme ne découle que de l'existence de dépendances entre les données correspondantes.

Un programme flot de données permettra donc aisément de décrire la structure d'un algorithme : possibilités de parallélisme et de pipe-line. En raison de l'absence d'effets de bord, conséquence habituelle de l'utilisation de la notion de variable, il est aisé d'effectuer des transformations sur un programme flot de données tout en préservant le caractère fonctionnel de ce programme (à un ensemble de données correspond un ensemble unique de sorties).

Il est clair que la caractéristique (C-1) des algorithmes qui nous intéressent suggère une orientation flot de données pour le langage SIGNAL ; dans ce langage, les fonctions opérant aux noeuds du réseau statique seront appelées des filtres. La description et la construction des réseaux de filtres sera détaillée plus loin.

II-12. "SIGNAL" est adapté à la description de communications de type synchrone entre filtres : L'existence d'une boucle privilégiée, que nous avons appelée "horloge-temps", ou horloge fondamentale, conduit naturellement à faciliter la gestion de cette boucle.

A cette fin, la notion classique de boucle est ici remplacée par la notion d'horloge. Une présentation formelle et complète de cette notion sort du cadre du présent rapport, et sera présentée ultérieurement. Nous nous contenterons d'indiquer ici ce que permet de décrire cette notion d'horloge.

Nous conviendrons de décrire par 0, 1, 2, ..., t, ... l'horloge fondamentale ; la notation générique pour une horloge sera H_0, H_1, ..., H_t, ..., où H_t représente le t-ième "top" de l'horloge, repéré par rapport à l'horloge fondamentale. Toutes les horloges sont, par définition, obtenues de manière récursive à partir de l'horloge fondamentale par l'une des opérations ci-dessous, où l'horloge d'origine sera, pour simplifier la description, l'horloge fondamentale :

- sous-échantillonnage $H_t := tp + q$
- sur-échantillonnage $H_t := \frac{t+q}{p}$
- retard $H_t := t + q$

où p et q sont des entiers ; ce type de construction d'horloge peut être réalisé par simple lecture du programme et ne nécessite pas d'exécution.

- extraction d'événements : $t \to H_t$ sera ici une fonction croissante de $\mathbb{N}$ dans $\mathbb{N}$ dépendant des données qui circulent dans le réseau (signaux de test) ; ce type d'horloge ne peut être engendré qu'à l'exécution.

- $\sup(H_t^1, H_t^2)$: sélectionne et réordonne l'ensemble des "top" des deux horloges.
- $\inf(H_t^1, H_t^2)$: sélectionne les "top" communs aux deux horloges.

L'ensemble de ces constructions permet d'engendrer, à partir de l'horloge fondamentale, tous les types de boucles qui ont été répertoriées à l'exposé des caractéristiques (C-1). En raison de la relative complexité de ces constructions, il y a lieu de fournir des moyens formels permettant de vérifier que toutes les horloges décrites dans un programme pourront effectivement être générées à l'exécution (problème des "deadlock" et des anticipations non causales). Tout cela nécessite l'introduction d'un formalisme qui fera l'objet d'un prochain rapport.

Nous considérons que les données circulent sur le réseau statique de filtres sous la forme de "signaux". Un signal est, par définition, la donnée d'un couple (flot,horloge), où le mot flot désigne une file, éventuellement infinie, de valeurs d'un certain type. Une différence avec le point de vue flot de données réside donc ici dans le caractère synchrone des communications qui sont décrites, par l'introduction de la notion d'horloge associée à un flot.

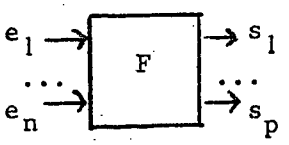
En conclusion : un programme SIGNAL consiste en la description des attributs des signaux, et d'un réseau statiquement défini de filtres interconnectés de manière orientée par ces signaux.

Avant de décrire de manière plus détaillée les réseaux de filtres, signalons que nous nous intéressons essentiellement aux primitives de construction de réseaux, nous omettons, dans cette note, un certain nombre de points ; en particulier les types de signaux et les constructions de filtres élémentaires ne sont que très sommairement introduits dans l'exemple final.

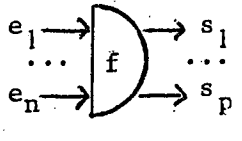
II-2. Description des réseaux de filtres.

Un filtre est représenté par un réseau dont les noeuds sont des "générateurs" associés aux opérations élémentaires du langage (+, x, ...), et dont les arcs orientés représentent le flot des données entre ces opérations élémentaires ; chaque opération élémentaire possède un certain nombre d'opérandes et de résultats représentés par des ports qui sont distingués par des noms [Milner 78] ; on peut également voir les ports comme les noeuds de données dans les graphes flots de données. Il n'est pas dans notre propos de décrire l'ensemble des générateurs du langage ni de détailler les types des données circulant entre les opérations élémentaires ; ces deux sujets ne seront abordés que lorsque ce sera nécessaire à la compréhension de l'exposé.

II-21. Représentation graphique : Sous certaines conditions précisées ultérieurement, certains ports de générateurs constituant un réseau sont visibles (et peuvent être l'objet de connexions) à l'extérieur d'un filtre ; celui-ci est représenté par un rectangle auquel aboutissent (dont dont issues) des flèches représentant les ports d'entrée (de sortie) visibles à l'extérieur du filtre ; un générateur est représenté par un demi-cercle muni également de flèches ; aux flèches sont associés les noms des ports.



filtre



générateur

Lorsque l'illustration des opérations effectuées sur le filtre le rend utile, on s'autorise à décrire partiellement la structure interne d'un filtre en prolongeant intérieurement les arcs visibles.

II-22. Formalisme de description de la syntaxe : Le langage SIGNAL est décrit dans la suite de la note par un ensemble de règles de productions sous une forme proche de BNF*. Pour alléger l'exposé nous avons adopté les conventions suivantes permettant de rendre implicites les descriptions des dérivations associées aux non-terminaux concernés.

- chaînes optionnelles : toute occurrence optionnelle d'un non-terminal <X> est représentée dans une chaîne par le non-terminal <O_X>.

- listes : tout non-terminal NT se dérivant en une liste d'occurrences du non-terminal <X> peut avoir les formes suivantes :
  - <L_X> si toutes les occurrences de <X> sont consécutives
  - <L_S_X> si deux occurrences successives de <X> sont séparées par un mot-clef (qui est alors dérivé de <MC_S_X>)
  - <L_B_X> si NT se dérive en au moins deux occurrences de <X> , deux occurrences successives étant alors séparées par un mot-clef (dérivé de <MC_S_X>).

II-3. Caractéristiques des réseaux.

Un réseau décrit la circulation des données entre les opérations (arithmétiques ou temporelles) sur signaux. Cette circulation doit respecter les règles qui sont présentées en II-31 et pour cela, les ports d'un filtre doivent vérifier les propriétés énoncées en II-32. Nous terminons ce paragraphe par la description de la structure d'un programme.

II-31. Interconnexions de filtres : Un arc entre deux opérations élémentaires représente un flot formé d'une file ordonnée finie ou infinie de valeurs typées ; un signal est la donnée d'un flot, d'une horloge indiquant la liste des instants (repérés par rapport à une horloge fondamentale) où les valeurs du flot sont définies, et de conditions initiales. La construction des arcs représentant les flots est obtenue à l'aide des opérateurs définis en III par interconnexion de ports de filtre selon des règles portant sur les noms de ces ports :

---

* Le lecteur étranger à ce genre de formalisme pourra se reporter en annexe A, avant de lire ce qui suit.

- Un lien entre un port d'entrée et un port de sortie ne peut être établi que si ces deux ports ont un même nom ;

- Une fois établi, un lien entre ports ne peut être rompu ;

- Dans un filtre, deux ports de sortie distincts ne peuvent posséder le même nom.

Cette dernière règle interdit d'alimenter un port d'entrée par des signaux issus de deux ports distincts ; si, pour la conception de systèmes, on peut donner une interprétation implicite raisonnable de cette opération (par exemple entrelacement non déterministe [Hoare 78, Milner 80]), il n'en va pas de même en traitement du signal où toute opération concernant des signaux distincts doit être explicite.

II-32. Interface d'un filtre : Afin de rendre possible la construction des filtres dont la structure interne respecte les règles ci-dessus, leurs ports d'entrée et de sortie doivent vérifier les propriétés suivantes :

- Les ports d'entrée non connectés sont obligatoirement visibles à l'extérieur.

- Les ports d'entrée connectés sont obligatoirement invisibles de l'extérieur (afin d'empêcher une seconde connexion).

- Les ports de sortie (connectés ou non connectés) peuvent rester visibles à l'extérieur ; il est en effet admis que le résultat d'un calcul n'est pas nécessairement utilisé (port non connecté devenu invisible) et, inversement le résultat d'un calcul peut être utilisé en diffusion (port connecté visible) ; l'utilisateur dispose en conséquence de moyens lui permettant de spécifier, parmi les ports de sortie d'un filtre, quels sont ceux qui restent utilisables pour des connexions externes. Cette possibilité est offerte au travers d'opérations de masquage de noms de ports, permettant de considérer un filtre comme une "boîte noire" où seuls sont spécifiés les potentialités de communication avec l'extérieur.

II-33. Structuration d'un programme : Un filtre pourra donc être obtenu à partir des opérations élémentaires du langage par deux classes d'opérateurs :

- des opérateurs permettant de constituer un réseau comme assemblage de sous-réseaux,

- des opérateurs permettant de modifier ou de masquer des noms dans un réseau.

Un programme SIGNAL est une déclaration de filtre constituée d'un identificateur, de paramètres éventuels suivis de la description de ce filtre en termes de ses composants et d'une partie déclarative fournissant la description de ces composants ; nous revenons sur la syntaxe du programme dans la description de l'exemple.

III. CONSTRUCTION DES RESEAUX.

Les réseaux sont construits à l'aide d'opérateurs sur filtres et de structures de constructions de motifs répétitifs.

Il existe deux classes d'opérateurs sur filtres :

- le masquage, le rebouclage et la mutation sont des opérateurs unaires (1) postfixés agissant sur les noms des ports du filtre opérande.

- la composition, la composition collatérale et la séquence sont des opérateurs binaires qui permettent de définir un filtre comme assemblage de deux filtres opérandes ; cet assemblage s'appuie sur l'identité des noms de leurs ports respectifs.

Les opérateurs permettent de définir une expression de comportement de filtre selon la syntaxe ci-dessous :

```
    <EXP_FILTRE> ::= <BINAIRE> ;
    <EXP_FILTRE> ::= <MOTIF> ;
    <MOTIF> ::= <PRIMAIRE> ;
    <PRIMAIRE> ::= <TERME> ;
(1) <PRIMAIRE> ::= <TERME> <L_UNAIRE> ;
    <PRIMAIRE> ::= <EXP_SIGNAL> ;
    <TERME> ::= <OCCURRENCE_FILTRE> ;
    <TERME> ::= <MC_D_PAR> <EXP_FILTRE> <MC_F_PAR> ;
    <MC_D_PAR> ::= ( ;
    <MC_F_PAR> ::= ) ;
```

III-1. Opérateurs sur les noms de filtre.

Un opérateur unaire op est donc un opérateur postfixé dont l'opérande est soit un <TERME>,soit un <TERME> auquel ont été appliqués les opérateurs unaires précédant op dans la liste <L_UNAIRE> associée (cf. ci-dessus). Il existe quatre classes d'opérateurs unaires selon la syntaxe suivante :

Syntaxe

```
(1a)  <UNAIRE> ::= <MC_D_ANTIRESTRICTION>
                   <LS_OCCURRENCE_FLOT>
                   <MC_F_ANTIRESTRICTION> ;
(1b)  <UNAIRE> ::= <MC_D_RESTRICTION> <O_L_S_OCCURRENCE_FLOT>
                   <MC_F_RESTRICTION> ;
(2)   <UNAIRE> ::= <MC_D_FERMETURE> <O_L_S_OCCURRENCE_FLOT>
                   <MC_F_FERMETURE> ;
(3)   <UNAIRE> ::= <MC_D_MUTATION> <L_S_DEFMUTA>
                   <MC_F_MUTATION> ;
```

Ces opérateurs sont maintenant présentés.

III-11. Masquage : Le masquage permet d'inhiber des connexions potentielles en rendant invisibles de l'extérieur certains noms de ports de sortie d'un filtre. Ces opérateurs permettent de faire d'un filtre une "boîte noire", les noms masqués pouvant être réutilisés indépendamment.

Rappelons que les ports d'entrée connectés dans un filtre F sont considérés comme invisibles hors de F.

Le masquage se présente sous les deux formes 1a et 1b ci-dessus dans lesquelles les mots-clés ont respectivement pour valeur :

<MC_S_OCCURRENCE_FLOT> ::= , ;
<MC_D_RESTRICTION> ::= / ;
<MC_F_RESTRICTION> ::= / ;
<MC_D_ANTIRESTRICTION> ::= { ;
<MC_F_ANTIRESTRICTION> ::= } ;

Les éléments définissant chaque opérateur sont les listes d'occurrences de flots que l'on veut masquer (1b) ou au contraire que l'on souhaite laisser seuls visibles. Chaque occurrence de flots peut être soit un flot, soit une famille de flots selon la syntaxe suivante :

Syntaxe

<OCCURRENCE_FLOT> ::= <FLOT> ;
<OCCURRENCE_FLOT> ::= <IDENT> <MC_D_INDICE_FLOT>
<EXP_ENTIER> <MC_F_INDICE_FLOT> ;
<FLOT> ::= <IDENT> ;
<FLOT> ::= <IDENT> <MC_D_INDICE_FLOT> <INTERVALLE>
<MC_F_INDICE_FLOT> ;
<INTERVALLE> ::= <EXP_ENTIER> <MC_S_INDICE> <EXP_ENTIER> ;
<MC_D_INDICE_FLOT> ::= { ;
<MC_F_INDICE_FLOT> ::= } ;
<MC_S_INDICE> ::= .. ;

Par exemple a est un nom, a{i} désigne le flot de nom a indicé par la valeur de i, a{i..j} désigne les flots de nom a indicés par les entiers compris entre les valeurs de i et de j incluses.

Sémantique

Selon sa forme syntaxique le masquage a les trois significations suivantes :

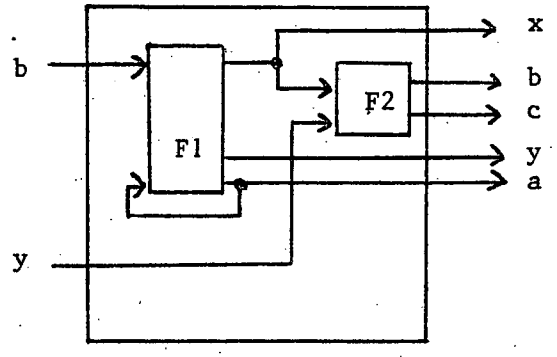
i) Par la production 1a tous les ports de sortie de l'opérande dont les noms n'apparaissent pas dans la <L_S_OCCURRENCE_FLOT> sont masqués.

Par la production 1b on peut avoir les effets suivants :

ii) Si la liste est vide tous les ports déjà connectés dans le filtre sont masqués.

iii) Si la liste est non vide, tous les ports de sortie dont les noms apparaissent dans cette liste sont masqués.

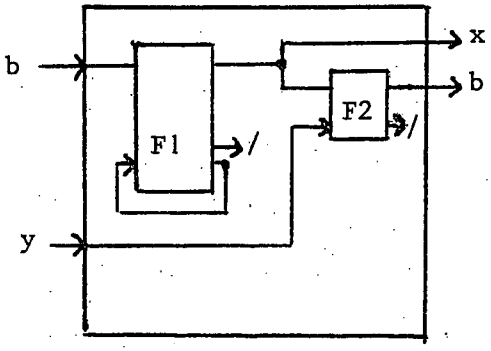
Exemples : Soit F le filtre suivant :



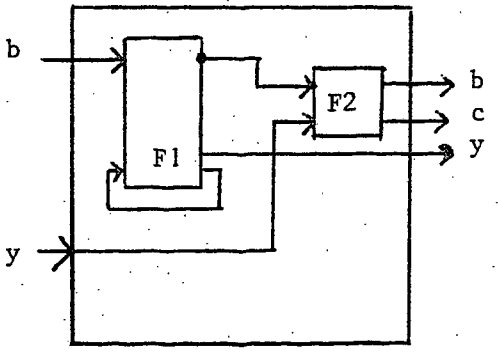
Remarque : L'arc rebouclant sur F1 n'est en principe pas visible hors de F1.

alors

F/a,c,y/ = F{b,x} =



F// =



III-12. Rebouclage : Le rebouclage permet d'effectuer des connexions entre des entrées et des sorties de même nom d'un filtre ; dans la règle syntaxique le décrivant (règle 2 du § III-1), le rebouclage est constitué d'une liste éventuellement vide d'occurrences de flot (dont la syntaxe est donnée au § III-11. ci-dessus) encadrée par deux occurrences du caractère '.

Sémantique

Selon sa forme syntaxique le rebouclage a les deux significations suivantes :

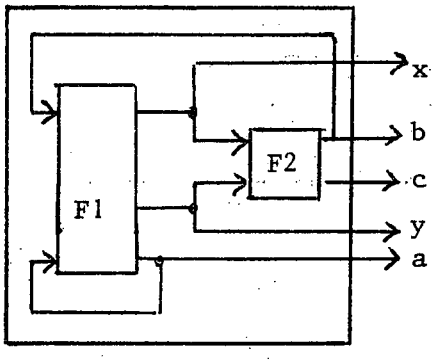
i) Si la liste est vide tous les ports d'entrée et de sortie de F de même nom sont respectivement connectés.

ii) Si la liste est non vide tous les ports d'entrée et de sortie dont le même nom est dans la liste sont respectivement connectés.

Exemple : Soit F le filtre dont le schéma est donné ci-dessus (§ III-11.)

alors

F'b,y' = F'' =



III-13. Mutation : La mutation permet de préparer des connexions futures en modifiant dans un filtre les noms de port (d'entrée ou de sortie) qui ne sont pas masqués. Elle n'a de sens que si elle ne conduit pas à donner un même nom à des ports de sortie distincts (on rappelle que toute action d'entrelacement de signaux doit être explicite). Dans la définition de la mutation (règle 3 du § III-1.) intervient une liste de redéfinitions de noms de ports (<L_S_DEFMUTA>) ; chacune de ces redéfinitions obéit à la syntaxe suivante :

(1) <DEFMUTA> ::= <OCCURRENCE_FLOT> <MC_RENOM>
<OCCURRENCE_FLOT> ;

(2) <DEFMUTA> ::= <MC_OUTPUT> <OCCURRENCE_FLOT>
<MC_RENOM> <OCCURRENCE_FLOT> ;

(3) <DEFMUTA> ::= <MC_INPUT> <ENS_OCCURRENCE_FLOT>
<MC_RENOM> <OCCURRENCE_FLOT> ;

<ENS_OCCURRENCE_FLOT> ::= <OCCURRENCE_FLOT> ;

<ENS_OCCURRENCE_FLOT> ::= <MC_D_PAR>
<L_S_OCCURRENCE_FLOT>
<MC_F_PAR> ;

<MC_RENOM> ::= : ;

<MC_S_DEFMUTA> ::= , ;

<MC_D_MUTATION> ::= < ;

<MC_F_MUTATION> ::= > ;

<MC_INPUT> ::= ? ;

<MC_OUTPUT> ::= ! ;

La règle 1(2) permet d'associer à tout port d'entrée ou de sortie (de sortie) de nom x, un nouveau nom y selon la syntaxe x : y.

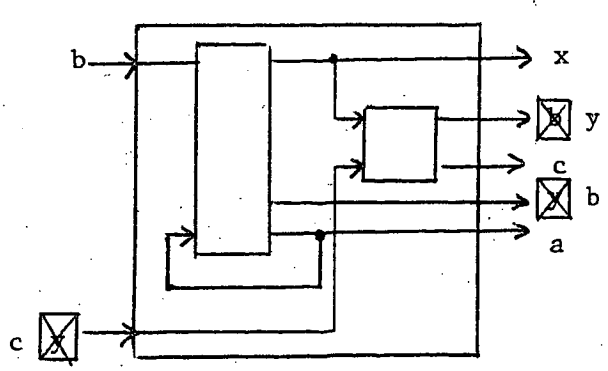
La règle 3 permet d'associer à un ensemble de ports d'entrée de noms respectifs x_1, ..., x_n un nouveau nom y (on aura alors diffusion du même signal sur toutes ces entrées) selon la syntaxe

$$(x_1, \ldots, x_n) : y$$

Il est clair qu'un nom est sujet d'au plus une redéfinition en entrée et une redéfinition en sortie, ou bien une redéfinition en entrée et en sortie par opération de mutation.

Exemple : Considérons le filtre F défini en III-11 :

F<?y:c,!y:b,!b:y> =



III-2. Expressions binaires de filtres.

Les expressions binaires de filtres sont données conformément à la syntaxe suivante :

<BINAIRE> ::= <L_B_COMPOSITION> ;
<BINAIRE> ::= <L_B_SEQUENCE> ;
<BINAIRE> ::= <L_B_COLLATERALE> ;
<SEQUENCE> ::= <PRIMAIRE> ;
<COMPOSITION> ::= <PRIMAIRE> ;
<COLLATERALE> ::= <PRIMAIRE> ;
<MC_S_COLLATERALE> ::= , ;
<MC_S_COMPOSITION> ::= | ;
<MC_S_SEQUENCE> ::= ; ;

Un <binaire> peut donc être une suite d'au moins deux <primaire> (par exemple des noms de filtre) séparés par le mot-clé dénotant l'opérateur considéré (respectivement ",", "|" et ";" pour la composition collatérale, la composition et la séquence). Cette définition interdit de mélanger, dans une expression de filtres non parenthésée, les différentes expressions binaires ; ce choix évite d'avoir à poser arbitrairement une priorité entre opérateurs et assure une meilleure lisibilité des programmes.

Plutôt que F ; G|H on devra écrire

(F;G)|H s'il s'agit d'une composition d'une séquence et d'un filtre

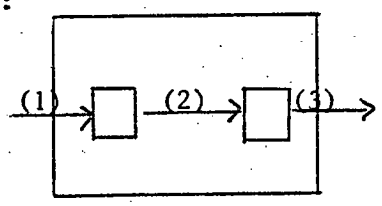
ou F;(G|H) s'il s'agit d'une séquence formée d'un filtre et d'une composition.

Chacun des opérateurs étant associatif, l'ordre d'évaluation d'une expression formée d'une suite d'opérations de même opérateur est indifférent. Le parenthésage est donc inutile dans une telle expression.

Dans la description ci-dessous de chacun des trois opérateurs binaires, nous en illustrons l'effet en nous intéressant à un port de nom a pouvant apparaître dans chacun des opérandes, nous considérons alors les trois filtres prototypes

et nous décrivons l'effet de chaque opérateur à l'aide d'un tableau dans lequel, par commodité, le nom a est omis et un port porte ce nom si et seulement si la flèche qui le représente traverse le cadre le plus externe du filtre.

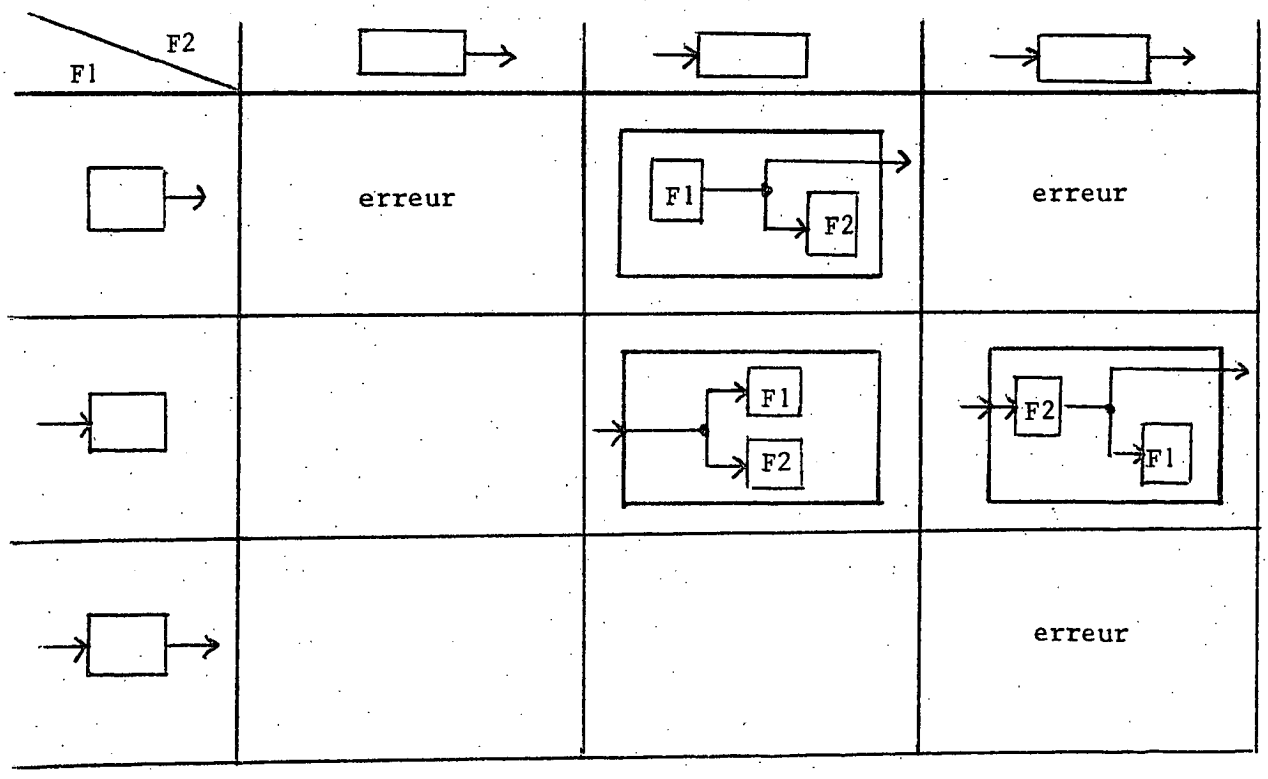
Exemple :



Les ports connectés par (2) ne possèdent pas de nom.
Les ports connectés par (1) et (3) possèdent le nom a.

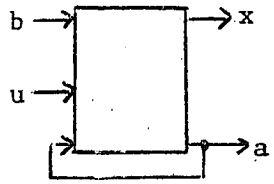
III-21. Composition : La composition est un opérateur binaire permettant de construire un filtre F par assemblage non ordonné de deux filtres opérandes F1 et F2 (elle est commutative) ; les sorties de F1 sont reliées aux entrées de F2 possédant le même nom et inversement. Les deux filtres ne doivent pas posséder une sortie de nom identique.

Tableau F = F1|F2

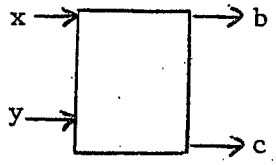


Exemple :

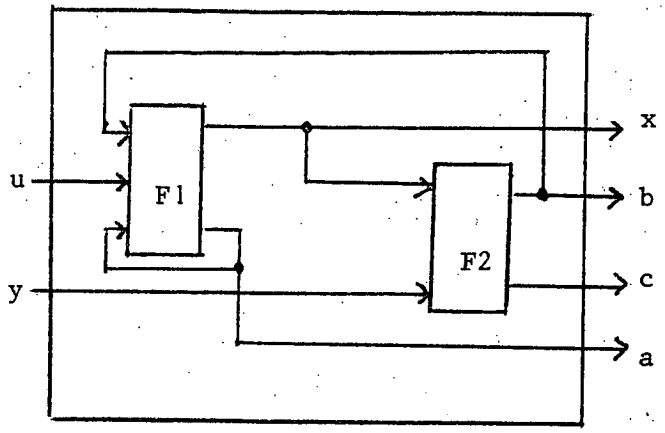
F1 =



F2 =



F = F1|F2 =

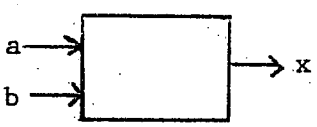


III-22. Composition collatérale : La composition collatérale est un opérateur qui, étant donné deux filtres F1 et F2, confond leurs ports d'entrée respectifs de même nom, en vue d'une diffusion du même signal sur ces ports ; le filtre F résultant ne doit pas posséder de ports de sortie distincts ayant un nom identique.

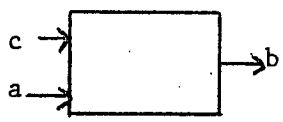
Propriété : La composition collatérale est commutative.

Exemple :

F1 =



F2 =



F = (F1,F2) =

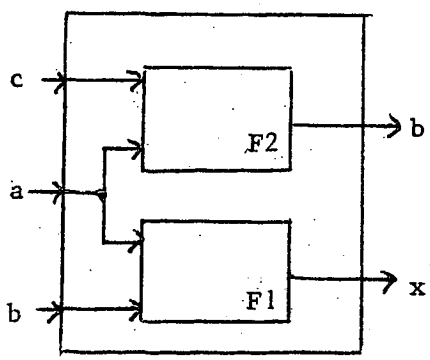
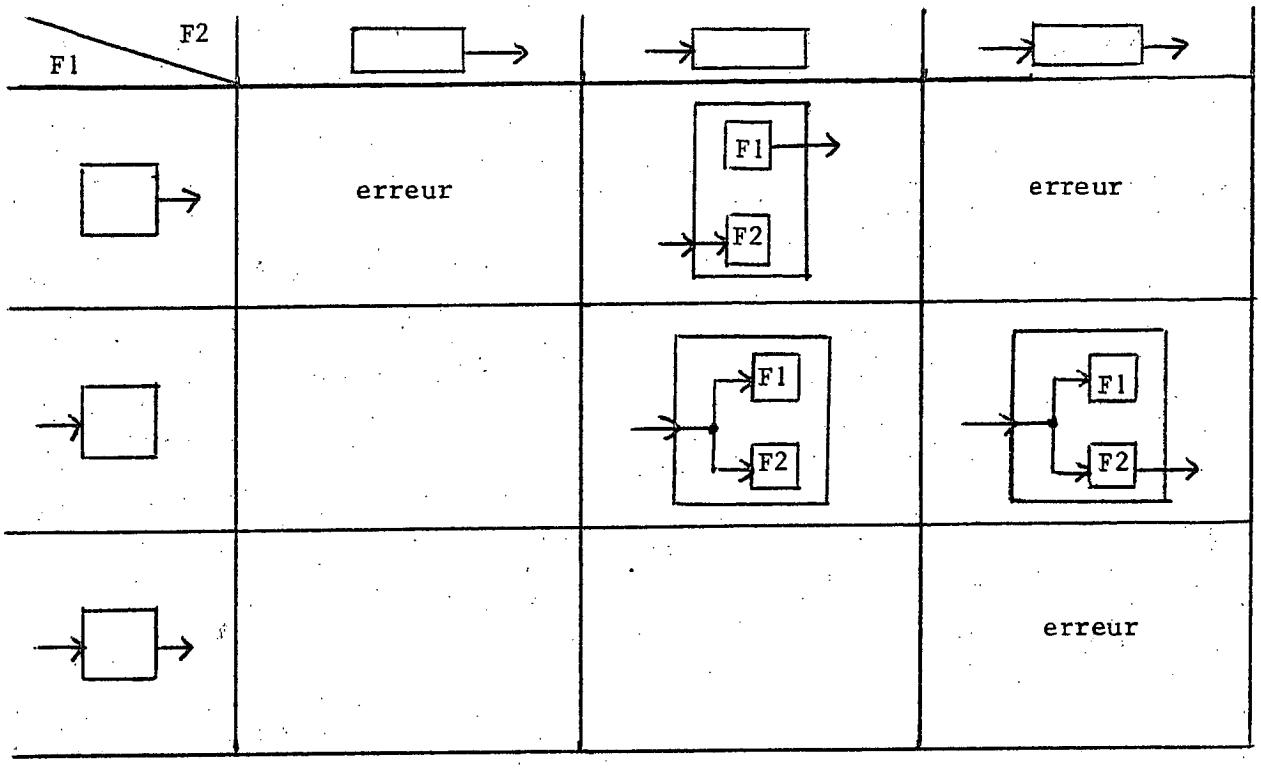


Tableau F = (F1,F2)

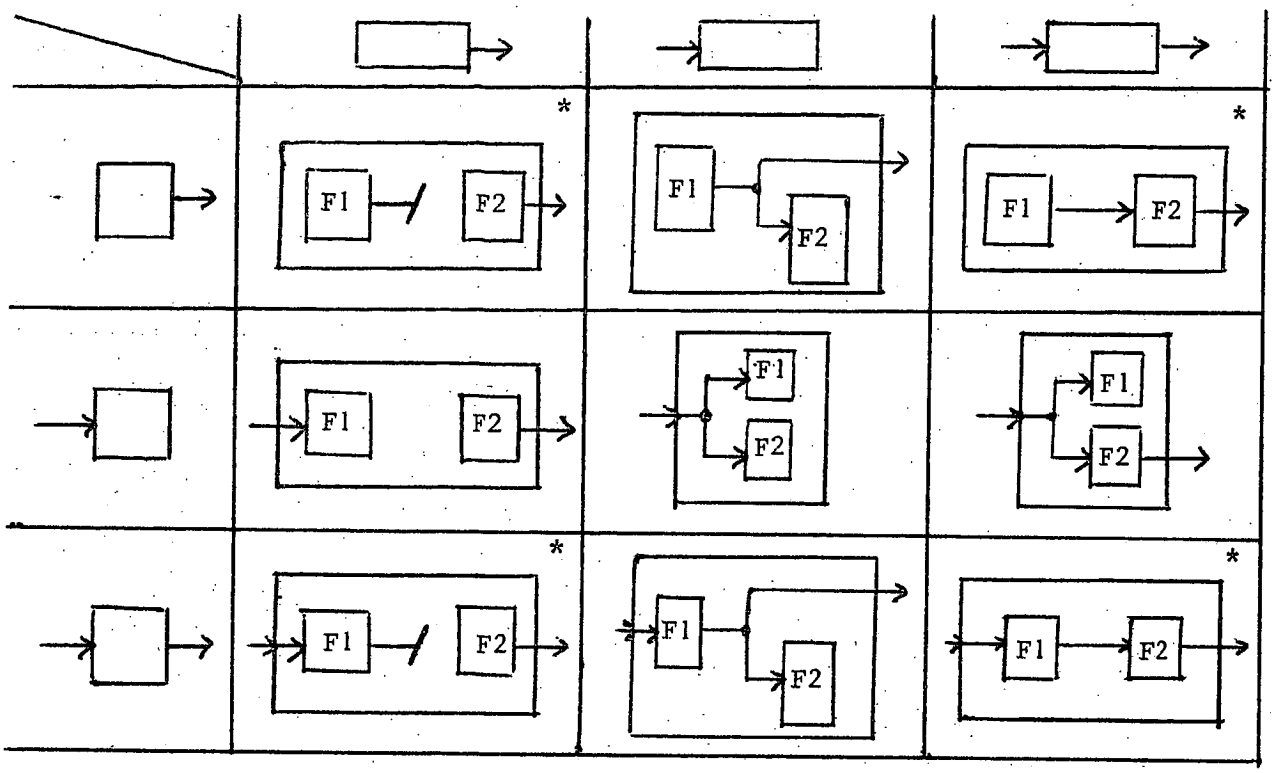


III-23. Séquence : La séquence (composition séquentielle) est un opérateur binaire qui, étant donné deux filtres F1 et F2, connecte les ports de sorties de F1 aux ports d'entrée de F2 de même nom et qui masque les sorties de F1 ayant un nom identique à une sortie de F2 ; cet opérateur n'est donc pas commutatif. Cet opérateur est utile à la fois pour la construction de motifs répétitifs et pour décrire des calculs intermédiaires par la mise en séquence d'expressions "d'affectation" ; à l'intérieur d'une séquence composée de filtres F1, F2, ... dans cet ordre un nom de ports peut être utilisé comme une variable. Il est à noter cependant que cette assimilation ne peut être qu'approximative dans la mesure où, à la différence des variables

- une valeur d'un signal sur un port de nom x ne peut être écrasée par la valeur suivante que si elle a été lue ;

- un port d'entrée non connecté ne peut être masqué (pas de défaut d'initialisation).

Tableau F = F1;F2

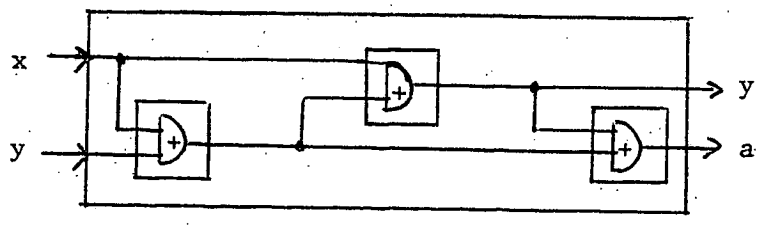


* résolution de conflits potentiels sur des entrées de même nom :

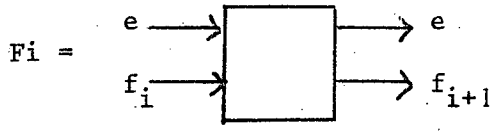
—/ représente une sortie masquée.

Exemples :

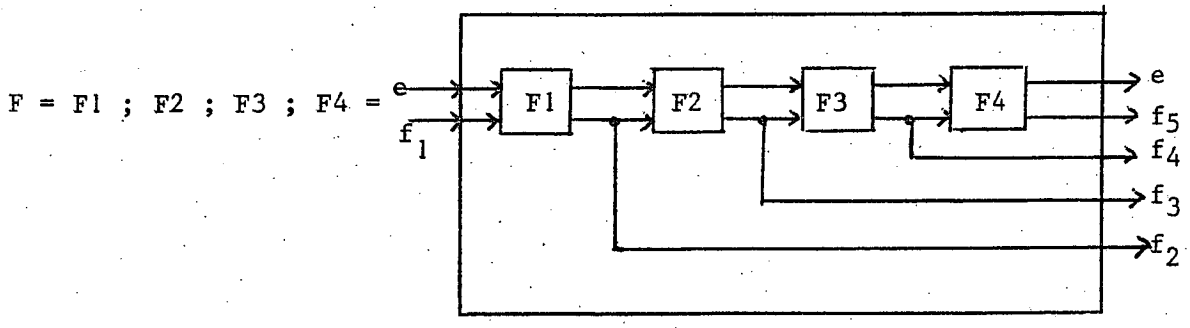
1) La séquence (a := x+y ; y := x+a ; a := a+y) peut être interprétée comme une séquence d'affectations PASCAL où les noms sont regardés comme des variables ; le filtre résultant représenté ci-dessous a pour entrées x et y pour leurs premières occurrences, et pour sorties a et y pour leurs dernières occurrences à gauche du signe d'affectation.



2) Pour i = 1, 2, 3, 4 soit



alors



III-3. Motifs à caractère répétitif.

A chacun des opérateurs binaires définis ci-dessus correspond une structure de définition de motifs à caractère répétitif selon la forme :

mot clé i in m..n of F

où mot clé peut être respectivement seq pour la séquence (;)
par pour la composition collatérale (,)
ser pour la composition (|)

Une telle structure définit un motif construit par la répétition (n-m+1 fois) de l'expression de filtre F au sein de laquelle l'identificateur i est utilisé dans des opérateurs de renommage. Cette construction est rendue possible par l'existence de noms indicés permettant de donner des noms calculés aux ports (cf. III-11.)

Syntaxe

```
<MOTIF> ::= <MC_SEQ> <CORPS_MOTIF> ;
<MOTIF> ::= <MC_COMPOS> <CORPS_MOTIF> ;
<MOTIF> ::= <MC_COLLAT> <CORPS_MOTIF> ;
<CORPS_MOTIF> ::= <FACTEUR> <MC_OF>  <MOTIF> ;
<FACTEUR> ::= <EXP_ENTIER> ;
<FACTEUR> ::= <IDENT> <MC_IN> <INTERVALLE> ;
<MC_OF> ::= of ;
<MC_COLLAT> ::= par ;
<MC_COMPOS> ::= ser ;
<MC_SEQ> ::= seq ;
```

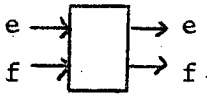
Sémantique

Nous présentons uniquement la structure séquentielle ; les différences respectives avec la structure composée et la structure collatérale étant uniquement liées à l'opérateur binaire associé, sont immédiatement déductibles.

a) L'expression seq i in m..n of F

où F est une expression de filtre et m,n deux valeurs entières vérifiant $0 \leq m \leq n$ est équivalente à la séquence composée de n-m+1 occurrences de F dans lesquelles i prend successivement ses valeurs dans l'intervalle [m,n].

Exemple : Nous reprenons l'exemple présenté en III-23.

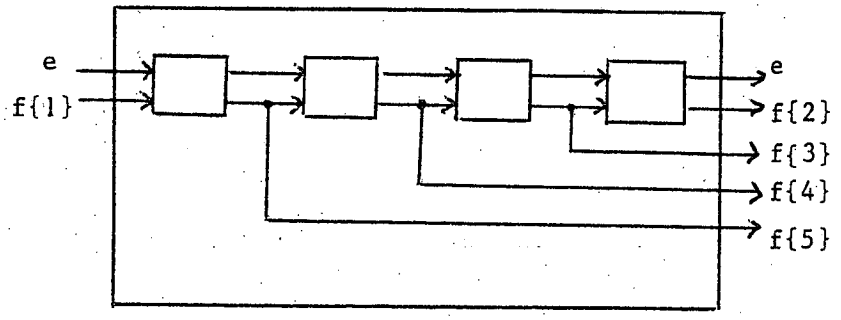
Soit F = 

Soit G = seq i in 1..4 of F <?f:f{i}, !f:f{i+1}>

Avec la définition ci-dessus G est donc le filtre

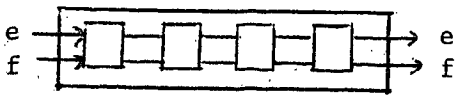
G ≡ F <?f:f{1}, !f:f{2}> ; i = 1
F <?f:f{2}, !f:f{3}> ; i = 2
F <?f:f{3}, !f:f{4}> ; i = 3
F <?f:f{4}, !f:f{5}> i = 4

représenté ci-dessous :



b) Lorsque les résultats intermédiaires d'une telle séquence sont inutiles, on peut omettre la définition de l'identificateur i. On a alors dans l'exemple ci-dessus :

G' = seq 4 of F($\equiv_{\Delta}$ G' = F ; F ; F ; F)

soit G' = 

IV. UN EXEMPLE.

A titre d'illustration, nous décrivons un algorithme-test, choisi pour la variété des problèmes que pose la description du réseau statique qui lui est associé. La suite de ce paragraphe est consacrée à la présentation de cet exemple (qui constitue une application plausible bien que non effective).

IV-1. Un égaliseur récursif avec auto-orthogonalisation partielle.

Le problème de l'égalisation en transmission de données peut être présenté de manière idéalisée comme suit :

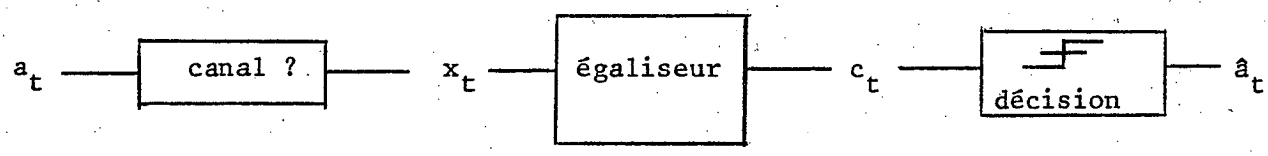


Fig. IV.1

La fig. (IV.1) présente le schéma dit "en bande de base équivalent" d'un système de transmission numérique. Le message (a_t) est émis à travers un canal $S = (s_k)$ inconnu, dont l'effet est de produire en sortie le signal

$$x_t = \sum_k s_k \, a_{t-k} + b_t, \tag{IV.1}$$

où b_t est un bruit. L'objet de l'égalisation est d'inverser cette transformation pour retrouver, avec un faible taux d'erreur, le message émis (a_t). Pour cela, l'idée est de mettre derrière (x_t) un filtre ajustable, appelé égaliseur, ayant pour objet de minimiser en moyenne quadratique l'erreur $c_t - a_t$, le message reconstitué $\hat{a}_t$ étant alors obtenu par quantification de c_t (a_t est à valeurs discrètes, typiquement de la forme ± 1, ± 3, ..., $\pm(2K+1)$ avec K petit).

Une structure d'égaliseur assez fréquemment utilisée est l'égaliseur récursif avec décision dans la boucle, où c_t est calculé par :

$$c_t = \sum_{i=-N}^{+N} h_i \, x_{t-i} + \sum_{j=1}^{P} k_j \, \hat{a}_{t-j} \tag{IV.2}$$

où les coefficients h_i et k_j sont à ajuster en permanence pour assurer la condition "$c_t - a_t$ petit". Une amélioration de (IV.2) consiste à remplacer les signaux x_t intervenant dans le produit de convolution par $2N+1$ signaux

$$x_t(+N), \ldots, x_t(0), \ldots, x_t(-N)$$

combinaisons linéaires des $x_{t+N}, \ldots, x_{t-N}$ (donc contenant la même information) mais décorrélés (ou orthogonaux), de façon à améliorer les propriétés numériques de l'algorithme ; une telle structure est proposée dans [Benveniste 79], le réseau qui lui est associé sera intéressant à décrire.

Décrivons maintenant l'algorithme.

L'orthogonaliseur adaptatif en treillis [Benveniste 81, Benveniste 82]. Il est donné (par exemple) par les formules

$$\text{(IV.3)} \qquad \begin{aligned} e_t(n+1) &= e_t(n) - k_t(n+1)\, f_{t-1}(n) \\ f_t(n+1) &= f_{t-1}(n) - k_t(n+1)\, e_t(n) \end{aligned}$$

$$\text{(IV.4)} \qquad e_t(0) = f_t(0) \triangleq x_{t+N}$$

$$\text{(IV.5)} \qquad k_t(n+1) = \mathrm{Cor}(k_{t-1}(n+1),\ e_t(n),\ f_{t-1}(n))$$

$$\text{(IV.6)} \qquad x_t(+N) = f_t(0),\ \ldots,\ x_t(-N) = f_t(2N),$$

où la fonction Cor n'a pas été spécifiée ici, par souci de simplicité : il s'agit d'un estimateur séquentiel de la corrélation entre les signaux $e_t(n)$ et $f_{t-1}(n)$.

Le filtre récursif ajustable.

$$\text{(IV.7)} \qquad \begin{aligned} c_t &= \sum_{i=-N}^{+N} h_i(t)\, x_t(-i) + \sum_{j=1}^{P} k_j(t)\, \hat{a}_{t-j} \\ &= H_t^T X_t + K_t^T \hat{A}_t, \end{aligned}$$

où

$$\text{(IV.8)} \qquad \begin{aligned} H^T &= (h_{-N}, \ldots, h_{+N}),\ K^T = (k_1, \ldots, k_P) \\ X_t^T &= (x_t(N), \ldots, x_t(-N)),\ \hat{A}_t^T = (\hat{a}_{t-1}, \ldots, \hat{a}_{t-P}) \end{aligned}$$

Les fonctions d'ajustement de H_t et K_t étant données par

$$\text{(IV.9)} \qquad \begin{aligned} H_{t+1} &= H_t + \gamma\, X_t(\hat{a}_t - c_t) \\ K_{t+1} &= K_t + \gamma\, \hat{A}_t(\hat{a}_t - c_t) \end{aligned}$$

La logique de décision.

Il n'est pas intéressant de la préciser ici ; elle est par exemple de la forme :

(IV.10) $\hat{a}_t = Q(c_t)$

où Q est simplement un quantificateur (par exemple $\hat{a}_t = \text{signe}(c_t)$ dans le cas où a_t est à valeurs ±1).

IV-2. Description de l'algorithme en SIGNAL.

Nous donnons tout d'abord quelques compléments de syntaxe (la syntaxe est dans l'annexe A2) avant de décrire l'égaliseur récursif en SIGNAL.

IV-21. Compléments de syntaxe : La syntaxe d'un filtre en SIGNAL comporte une identification, une partie optionnelle de spécification et un corps selon la forme générale suivante (où % est un commentaire %)

```
(1)  F % nom du filtre %
(2)    "(... % paramètres de structure %)                  identification (a)
(3)    '(... % attributs %)
(4)    :(... % paramètres d'initialisations %)
(5)      input ... % spécification des entrées %           spécification (b)
         output ... % spécification des sorties %
(6)    = ... % expression de définition du comportement
               du filtre %
       where
(7)      ... % attribution de types aux flots %
       filter                                              corps (c)
         ... % déclarations locales de filtres %
(8)  end F.
```

a) L'identification du filtre comprend, outre son nom (1) :

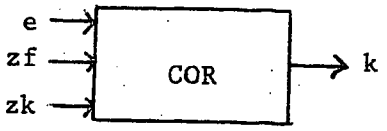
- une liste optionnelle de paramètres formels entiers (2) permettant de construire des modèles de filtres traitant des vecteurs de longueurs différentes (cf. CASCADE dans le traitement de l'exemple)

- un ensemble optionnel d'attributs formels (3) parmi lesquels le type permet de décrire un type implicite pour les flots dont le type n'est pas explicite dans le corps.

Le calcul du coefficient k (IV.5) est exprimé par la composition de deux filtres :

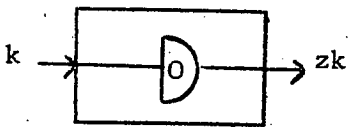
- (E5) `cor` occurrence d'un modèle de filtre "cor" possédant les entrées zf, zk, e et la sortie k dont on ne décrit pas ici la structure.

R5



- (E6) `zk is k delay 1 init 0` est une expression temporelle en SIGNAL permettant de définir le signal zk comme étant le signal k retardé de 1 et initialisé à 0. On représente le réseau associé par la figure suivante :

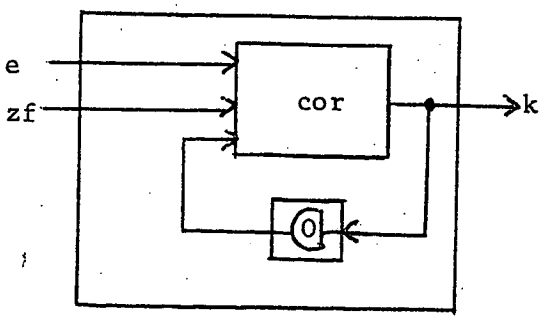
R6



Le calcul de k est alors écrit en SIGNAL sous la forme suivante (dans laquelle on masque le port zk) :

- (E56) `(zk is k delay 1 init 0|cor)/zk/`

(R56)



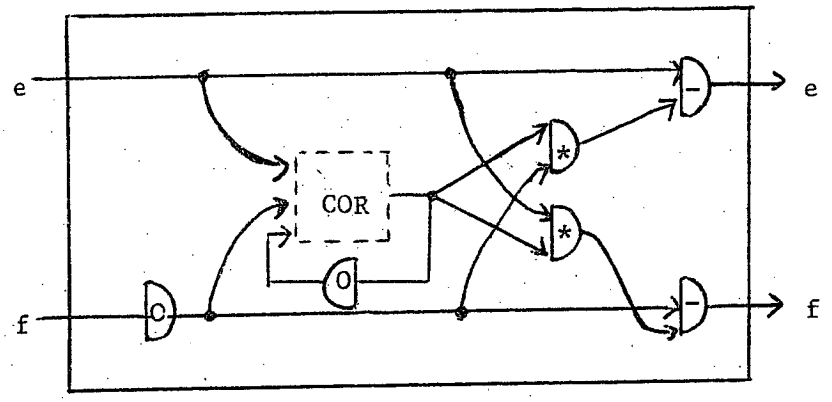
Finalement, le système d'équations (IV.3, IV.5) est décrit par la mise en séquence en SIGNAL de l'expression temporelle définissant zf et des expressions (E56) et (E12) donnant le réseau associé suivant :

- (E12356)

```
(zf is f delay 1 init 0 ;
   (zk is k delay 1 init 0|COR)/zk/ ;
   (e := e - k * zf,f := zf - k * e)) {e,f}
```

l'opérateur {e,f} ne laissant visibles que les ports de nom e et f.

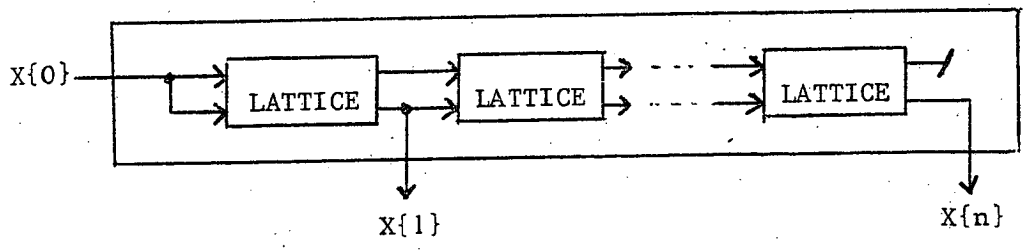
(R12356)



IV-23. Déclarations de filtres : On définit l'orthogonaliseur en SIGNAL par une déclaration de filtre (ORTHOG) utilisant la structure de construction répétitive séquentielle et comportant une partie déclarative décrivant l'élément construit ci-dessus :

```
ORTHOG"(n) =
    (seq i in 1..n of LATTICE<?f:X{i-1}, !f:X{i}>)
            <?e:X{0}>/e/
   where
      filter LATTICE = % expression E12356 %
           where filter COR:{input e, zf, zk  output k} = signal.
           end LATTICE.
   end ORTHOG.
```

A cette déclaration (dans laquelle on peut noter la définition du filtre externe COR) correspond le réseau suivant :

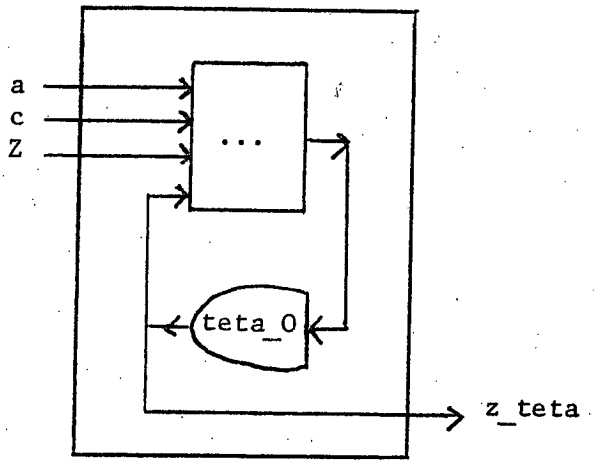


On complète maintenant la définition de l'égaliseur par les définitions suivantes :

Ajustement :

```
AJUST"(k) :  (gamma ;[0..k-1]teta_0) {input a,c;[0..k-1]Z output[0..k-1]z_teta} =
     (z_teta is teta delay 1 init teta_0 | teta := z_teta_gamma*z*(c-a))/teta/
     where
       [0..k-1] teta  % teta est de type [0..k-1]μ, μ étant le type implicite %
end AJUST.
```

Le filtre AJUST (de paramètres gamma et teta_0 et de paramètre de structure k) correspond donc au réseau :



Quantificateur :

C'est un filtre externe :

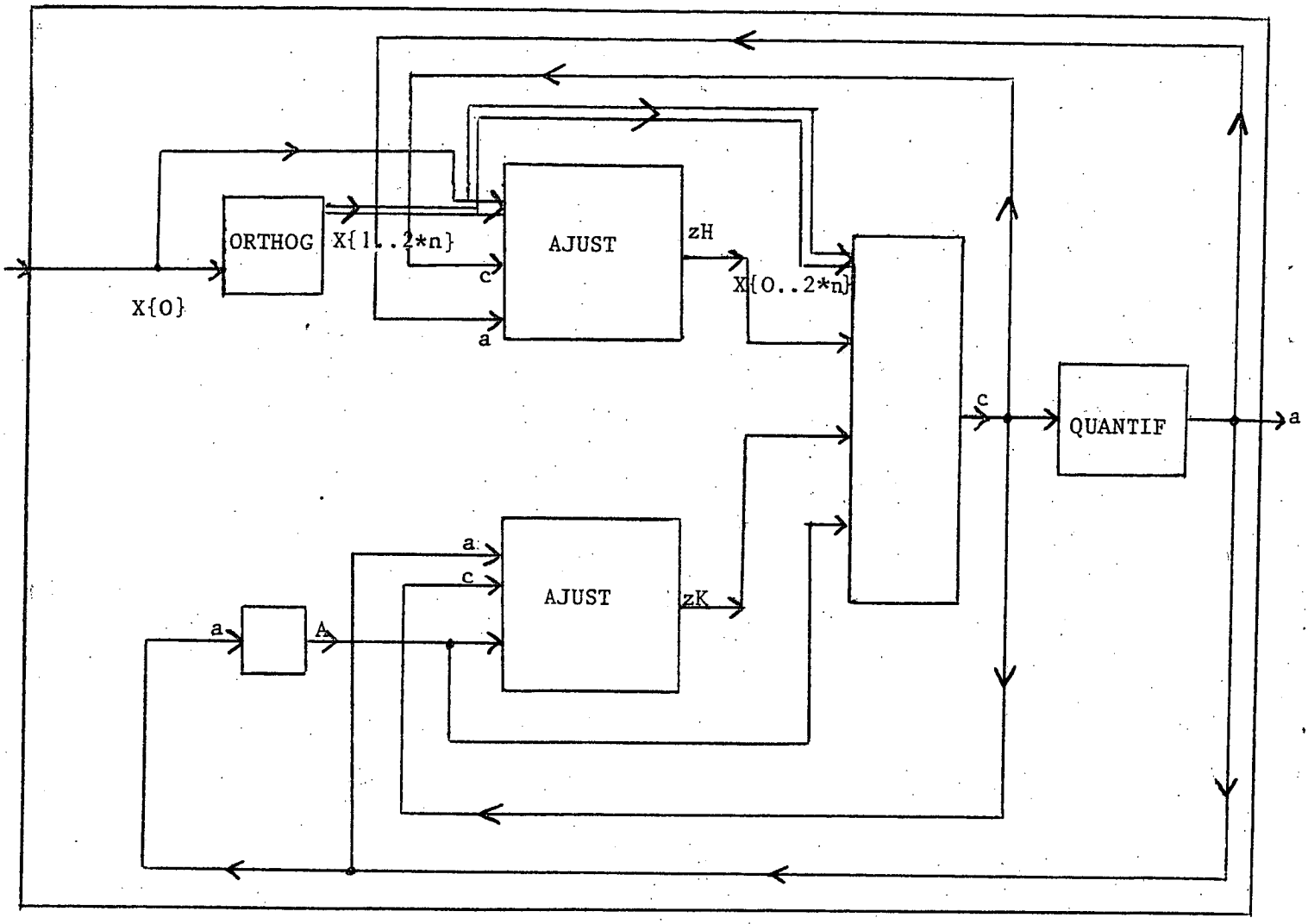
```
QUANTIF : {input c output a} = signal % quantification non définie de c pro-
                                       duisant a %.
```

Complétons maintenant la définition du filtre EGALISEUR :

```
EGALISEUR"(n,p):(gamma1,gamma2;[0..2*n]H0;[0..p-1]K0) {input x output a} =
   (
    ((
      (ORTHOG"(2*n);AJUST"(2 n+1)(gamma1,H0)<? Z:X{0..2*n} ! z_teta:zH>),
      (A is a(p) delay 1 init[1..p=>0];AJUST"(p)(gamma2,K0)<? Z:A
                                                    ! z_teta:zK>)
     ) ; c := zH * X + zK * A) 'c'
     ; QUANTIF
    ) 'a' <? X{0}:x> {a,x}
    where
        [0..2*n]zH ; [0..p-1]A, zK
        filter
            ORTHOG ... = ...
            AJUST ... = ...        % voir ci-dessus les définitions de ces
            QUANTIF ... = ...      trois filtres %
end EGALISEUR.
```

On note la fenêtre de p éléments sur le flot a permettant de constituer le flot A, décalé de 1 par rapport à a (A_t est donc le signal vectoriel $\{a_{t-1}, \ldots, a_{t-p}\}$, A_o étant égal à 0 dans toutes ses composantes par l'initialisation [1..p => 0]).

Le réseau est finalement le suivant :


ORTHOG
X{1..2*n}
AJUST
zH
X{0..2*n}
X{0}
c
a
c
QUANTIF
a
a
c
AJUST
zK
a
A

ANNEXE A : Descriptions de langages.

Un langage L est un ensemble de mots définis sur un vocabulaire V_T ; il est traditionnellement décrit par une grammaire $G = (V_N, V_T, P, S)$, dans laquelle V_T est le vocabulaire des mots de L (ensemble des terminaux ou alphabet terminal), V_N est un ensemble de symboles dont l'intersection avec V_T est vide (alphabet non terminal), S appelé axiome est un symbole distingué de V_N, et P est un ensemble de règles de productions dont chacune associe (pour les grammaires qui nous intéressent ici) un mot formé de symboles de V_N et de V_T à un symbole de V_N. Le langage décrit par une grammaire G est l'ensemble des mots $\alpha_i^{n_i}$ contenant uniquement des terminaux et obtenus, à partir de $\alpha_i^o = S$, par application successive de substitutions vérifiant :

si $\alpha_i^j = \beta X \gamma$ est dérivé de α_i^o
il existe une règle de P qui associe δ à X
où β, γ, δ sont des mots contenant des symboles non terminaux et/ou terminaux
X est un symbole non terminal
alors $\alpha_i^{j+1} = \beta \delta \gamma$ est dérivé de α_i^o

Exemple :

Si on note X —> γ, la règle qui associe γ à X, le langage contenant les mots formés d'une suite alternée de "a" et de "b" commençant par "a" ({a, ab, aba, abab, ...}) peut être décrit par la grammaire suivante : G = ({A,B}, {a,b}, {A—>a,A—>aB,B—>bA,B→b},A). Le mot abab est alors accessible par la suite de substitutions

A, aB, abA, abaB, abab

Un langage de programmation est un langage dont les mots (formés sur un sous-ensemble de caractères traditionnels) sont des programmes.

Pour décrire les règles de production d'une grammaire on utilise un "meta-langage" comme BNF (Backus-Naur Form) employé dans la note.

En BNF les symboles du vocabulaire non-terminal sont de la forme <α> où α est une chaîne de caractères.

Les règles sont décrites par les définitions formées d'un non-terminal suivi du symbole ::=, lui-même suivi du mot associé ; dans notre cas les règles sont terminées par le dernier symbole ; d'une ligne.

On décrirait par exemple le langage ci-dessus au moyen des règles :

```
<A> ::= a ;
<A> ::= a<B> ;
<B> ::= b<A> ;
<B> ::= b ;
```

La présentation succincte ci-dessus est limitée aux constructions utilisées dans la note ; le lecteur intéressé ou insuffisamment éclairé peut consulter [Pagan 81].

ANNEXE B.

```
definition of SIGNAL_BNF is

    <FILTRE_AXIOME>::= <FILTRE> ;
    <FILTRE>::= <IDENTIFICATION> <MC_DENOTE> <CORPS> <MC_F_FILTRE> ;
    <FILTRE>::= <IDENTIFICATION> <SPECIF> <MC_DENOTE> <CORPS>
                <MC_F_FILTRE> ;
    <FILTRE>::= <IDENTIFICATION> <SPECIF> <MC_DENOTE> <LANGAGE>
                <MC_F_FILTRE> ;
    <LANGAGE>::= <MC_FORTRAN> ;
    <LANGAGE>::= <MC_SIGNAL> ;
    <MC_DENOTE>::= = ;
    <MC_FORTRAN>::= fortran ;
    <MC_SIGNAL>::= signal ;
    <MC_F_FILTRE>::= . ;

    chapter IDENTIFICATION
        <IDENTIFICATION>::= <IDENT> <O_FORMAL_STRUCT>
                            <O_FORMAL_ATTRIBUT> ;
        <FORMAL_STRUCT>::= <MC_D_GENERIQUE> <L_S_IDENT> <MC_F_PARAM> ;
        <FORMAL_ATTRIBUT>::= <MC_D_INSTANCE> <MC_F_PARAM> ;
        <MC_S_IDENT>::= , ;

    chapter SPECIFICATION
        <SPECIF>::= <MC_D_SPECIF> <FORMAL_VAL> ;
        <SPECIF>::= <MC_D_SPECIF> <O_FORMAL_VAL> <SPECIF_IO> ;
        <FORMAL_VAL>::= <MC_D_PARAM> <L_S_TYPE_FLOTS> <MC_F_PARAM> ;
        <SPECIF_IO>::= <MC_D_SPECIF_IO> <O_SPECIF_INPUT> <SPECIF_OUTPUT>
                       <MC_F_SPECIF_IO> ;
        <SPECIF_INPUT>::= <MC_INPUT> <L_S_TYPE_FLOTS> ;
        <SPECIF_OUTPUT>::= <MC_OUTPUT> <L_S_TYPE_FLOTS> ;
        <MC_D_SPECIF>::= : ;
        <MC_D_SPECIF_IO>::= { ;
        <MC_F_SPECIF_IO>::= } ;

    chapter CORPS
        <CORPS> ::= <EXP_FILTRE> ;
        <CORPS> ::= <EXP_FILTRE> <O_DECLARATIVE> <MC_F_CORPS> <O_IDENT> ;
        <MC_F_CORPS>::= end ;

        chapter EXPRESSIONS_FILTRES
            <EXP_FILTRE>::= <BINAIRE> ;
            <EXP_FILTRE>::= <MOTIF> ;
            <MOTIF> ::= <PRIMAIRE> ;
            <PRIMAIRE>::= <TERME> ;
            <PRIMAIRE>::= <TERME> <L_UNAIRE> ;
            <PRIMAIRE>::= <EXP_SIGNAL> ;
            <TERME> ::= <OCCURRENCE_FILTRE> ;
            <TERME> ::= <MC_D_PAR> <EXP_FILTRE> <MC_F_PAR> ;
```

```
chapter BINAIRES
      <BINAIRE>::= <L_B_COMPOSITION> ;
      <BINAIRE>::= <L_B_SEQUENCE> ;
      <BINAIRE>::= <L_B_COLLATERALE> ;
      <SEQUENCE>::= <PRIMAIRE> ;
      <COMPOSITION>::= <PRIMAIRE> ;
      <COLLATERALE>::= <PRIMAIRE> ;
      <MC_S_COLLATERALE>::= , ;
      <MC_S_COMPOSITION>::= | ;
      <MC_S_SEQUENCE>::= ; ;

chapter MOTIFS
      <MOTIF> ::= <MC_SEQ> <CORPS_MOTIF> ;
      <MOTIF> ::= <MC_COMPOS> <CORPS_MOTIF> ;
      <MOTIF> ::= <MC_COLLAT> <CORPS_MOTIF> ;
      <CORPS_MOTIF>::= <FACTEUR> <MC_OF> <MOTIF> ;
      <FACTEUR>::= <EXP_ENTIER> ;
      <FACTEUR>::= <IDENT> <MC_IN> <INTERVALLE> ;
      <MC_OF>::= of ;
      <MC_COLLAT>::= par ;
      <MC_COMPOS>::= ser ;
      <MC_SEQ>::= seq ;

chapter UNAIRES
      <UNAIRE>::= <MC_D_MUTATION> <L_S_DEFMUTA>
                  <MC_F_MUTATION> ;
      <UNAIRE>::= <MC_D_RESTRICTION> <O_L_S_OCCURRENCE_FLOT>
                   <MC_F_RESTRICTION> ;
      <UNAIRE>::= <MC_D_ANTIRESTRICTION>
                  <L_S_OCCURRENCE_FLOT>
                  <MC_F_ANTIRESTRICTION> ;
      <UNAIRE>::= <MC_D_FERMETURE> <O_L_S_OCCURRENCE_FLOT>
                  <MC_F_FERMETURE> ;

      chapter MUTATION
            <DEFMUTA>::= <OCCURRENCE_FLOT> <MC_RENOM>
                         <OCCURRENCE_FLOT> ;
            <DEFMUTA>::= <MC_INPUT> <ENS_OCCURRENCE_FLOT>
                         <MC_RENOM> <OCCURRENCE_FLOT> ;
            <DEFMUTA>::= <MC_OUTPUT> <OCCURRENCE_FLOT>
                         <MC_RENOM> <OCCURRENCE_FLOT> ;
            <ENS_OCCURRENCE_FLOT>::= <OCCURRENCE_FLOT> ;
            <ENS_OCCURRENCE_FLOT>::= <MC_D_PAR>
                                     <L_S_OCCURRENCE_FLOT>
                                     <MC_F_PAR> ;
            <MC_RENOM>::= : ;
            <MC_S_DEFMUTA>::= , ;

      chapter MOTS_CLES
            <MC_D_ANTIRESTRICTION>::= { ;
            <MC_D_FERMETURE>::= ' ;
            <MC_D_MUTATION>::= < ;
            <MC_D_RESTRICTION>::= / ;
            <MC_F_ANTIRESTRICTION>::= } ;
            <MC_F_FERMETURE>::= ' ;
            <MC_F_MUTATION>::= > ;
            <MC_F_RESTRICTION>::= / ;
            <MC_S_OCCURRENCE_FLOT>::= , ;
```

```
chapter OCCURRENCE
     <OCCURRENCE_FILTRE>::= <INSTANCE_FILTRE> ;
     <OCCURRENCE_FILTRE>::= <INSTANCE_FILTRE> <MC_D_PARAM>
                            <L_S_EXP_ARITHM> <MC_F_PARAM> ;
     <INSTANCE_FILTRE>::= <GENERIQUE_FILTRE> ;
     <INSTANCE_FILTRE>::= <GENERIQUE_FILTRE>
                          <ATTRIBUT_FILTRE> ;
     <GENERIQUE_FILTRE>::= <IDENT> ;
     <GENERIQUE_FILTRE>::= <IDENT> <MC_D_GENERIQUE>
                           <L_S_EXP_ENTIER> <MC_F_PARAM> ;
     <ATTRIBUT_FILTRE>::= <MC_D_INSTANCE> <TYPE>
                          <MC_F_PARAM> ;
     <MC_S_EXP_ARITHM>::= , ;

chapter EXP_SIGNAL
     <EXP_SIGNAL>::= <CONSTANTE> ;
     <EXP_SIGNAL>::= <AFFECTATION> ;
     <EXP_SIGNAL>::= <EXP_TEMPOR> ;

     chapter CONSTANTE
          <CONSTANTE>::= <IDENT> <MC_DENOTE> <EXP_ARITHM> ;

     chapter AFFECTATION
          <AFFECTATION>::= <VARIABLE> <MC_DEVIENT>
                           <EXP_ARITHM> ;
          <MC_DEVIENT>::= := ;

     chapter TEMPOREL
          <EXP_TEMPOR>::= <IDENT> <MC_DEFRETARD>
                          <FLOT_RETARDE> <O_INIT> ;
          <FLOT_RETARDE>::= <OCCURRENCE_FLOT> <O_FENETRE>
                            <O_RETARD> ;
          <RETARD>::= <MC_DELAY> <EXP_ENTIER> ;
          <FENETRE>::= <MC_D_FENETRE> <EXP_ENTIER>
                       <MC_F_FENETRE> ;
          <INIT>  ::= <MC_INIT> <EXP_ARITHM> ;
          <MC_D_FENETRE>::= ( ;
          <MC_DEFRETARD>::= is ;
          <MC_DELAY>::= delay ;
          <MC_F_FENETRE>::= ) ;
          <MC_INIT>::= init ;

chapter DECLARATIONS
     <DECLARATIVE>::= <MC_D_DECLARATIVE> <O_L_S_TYPE_FLOTS>
                      <O_FORMAL_FILTERS> ;
     <FORMAL_FILTERS>::= <MC_FORMAL_FILTERS> <L_FILTRE> ;
     <MC_D_DECLARATIVE>::= where ;
     <MC_FORMAL_FILTERS>::= filter ;
```

```
chapter COMMUN
     <TYPE_FLOTS>::= <TYPE> <L_S_FLOT> ;

     chapter TYPES
          <TYPE>  ::=  ;
          <TYPE>  ::= <ID_TYPE> ;
          <TYPE>  ::= <L_INDICES> <ID_TYPE> ;
          <TYPE>  ::= <L_INDICES> ;
          <INDICES>::= <MC_D_VECT> <INTERVALLE> <MC_F_VECT> ;
          <ID_TYPE>::= real ;
          <ID_TYPE>::= longreal ;
          <ID_TYPE>::= compl ;
          <ID_TYPE>::= longcompl ;
          <ID_TYPE>::= int ;
          <ID_TYPE>::= bool ;

     chapter FLOTS
          <OCCURRENCE_FLOT>::= <FLOT> ;
          <OCCURRENCE_FLOT>::= <IDENT> <MC_D_INDICE_FLOT>
                               <EXP_ENTIER> <MC_F_INDICE_FLOT> ;
          <FLOT>  ::= <IDENT> ;
          <FLOT>  ::= <IDENT> <MC_D_INDICE_FLOT> <INTERVALLE>
                      <MC_F_INDICE_FLOT> ;
          <INTERVALLE>::= <EXP_ENTIER> <MC_S_INDICE> <EXP_ENTIER> ;
          <MC_D_INDICE_FLOT>::= ( ;
          <MC_F_INDICE_FLOT>::= ) ;
          <MC_S_INDICE>::= .. ;

     chapter EXPRESSIONS
          <SIGNAL>::= <OCCURRENCE_FLOT> <O_L_SELECTION> ;
          <VARIABLE>::= <IDENT> <O_L_SELECTION> ;
          <SELECTION>::= <NON_DEFINI> ;

          chapter EXP_ARITHM
               <EXP_ARITHM>::= <MC_D_VECT> <DEF_VECT> <MC_F_VECT> ;
               <EXP_ARITHM>::= <SIGNAL> ;

               chapter DEF_VECT
                    <DEF_VECT>::= <IDENT> <MC_IN> <INTERVALLE>
                                  <MC_DEF_VECT> <DEF_COMPOSANTS> ;
                    <DEF_VECT>::= <INTERVALLE> <MC_DEF_VECT>
                                  <EXP_ARITHM> ;
                    <DEF_COMPOSANTS>::= <EXP_ARITHM> ;
                    <DEF_COMPOSANTS>::= <MC_D_DEF_COND> <L_S_CONDDEF>
                                        <MC_F_DEF_COND>
                                        <EXP_ARITHM> ;
                    <CONDDEF>::= <EXP_BOOL> <MC_OP_DEF_VECT>
                                 <EXP_ARITHM> ;
                    <MC_S_CONDDEF>::= elif ;
                    <MC_DEF_VECT>::= => ;
                    <MC_OP_DEF_VECT>::= : ;
                    <MC_D_DEF_COND>::= if ;
                    <MC_F_DEF_COND>::= else ;
```

```
        chapter EXP_BOOL
             <EXP_BOOL>::= <NON_DEFINI> ;

        chapter EXP_ENTIER
             <EXP_ENTIER>::= <EXP_ENTIER> <MC_OP_SOMME>
                             <PRODUIT_ENTIER> ;
             <EXP_ENTIER>::= <PRODUIT_ENTIER> ;
             <PRODUIT_ENTIER>::= <PRODUIT_ENTIER> <MC_OP_PRODUIT>
                                 <TERME_ENTIER> ;
             <PRODUIT_ENTIER>::= <TERME_ENTIER> ;
             <TERME_ENTIER>::= <ENTIER> ;
             <TERME_ENTIER>::= <MC_D_PAR> <EXP_ENTIER> <MC_F_PAR> ;
             <MC_S_EXP_ENTIER>::= , ;
             <ENTIER>::= <IDENT> ;
             <ENTIER>::= <NOMBRENTIER> ;

    chapter LITTERAUX
        <NON_DEFINI>::= %NONDEFINI ;
        <IDENT> ::= %IDENT ;
        <NOMBRENTIER>::= %NOMBRENTIER ;

    chapter MOTS_CLES
        <MC_D_GENERIQUE>::= ''( ;
        <MC_D_INSTANCE>::= '( ;
        <MC_D_PAR>::= ( ;
        <MC_D_PARAM>::= ( ;
        <MC_D_VECT>::= [ ;
        <MC_F_PAR>::= ) ;
        <MC_F_PARAM>::= ) ;
        <MC_F_VECT>::= ] ;
        <MC_IN>::= in ;
        <MC_INPUT>::= input ;
        <MC_INPUT>::= ? ;
        <MC_OP_SOMME>::= + ;
        <MC_OUTPUT>::= output ;
        <MC_OUTPUT>::= ! ;
        <MC_S_FLOT>::= , ;
        <MC_S_TYPE_FLOTS>::= : ;
end definition
```

REFERENCES.

[Ackerman 79]

W.B. Ackerman : "Data Flow Languages"
Proc. AFIPS Conference (E. Merwin, Ed.), New York, 1979.

[Achcroft 77]

E.A. Ashcroft, W.W. Wadge : "Lucid, a Nonprocedural Language with iteration"
CACM, Vol. 20, n° 7, 1977.

[Backus 78]

J. Backus : "Can Programming Be Liberated from the von Neuman Style ?
A Functional Style and Its Algebra of Program"
CACM, Vol. 2, n° 8, August 1978.

[Benveniste 79]

A. Benveniste, C. Chauré : "Une Méthode Rapide pour arriver à des algorithmes d'estimation rapides pour les fonctions de transfert AR et ARMA synthétisée en treillis"
Publication Interne IRISA n° 116, 1979.

[Benveniste 81]

A. Benveniste, C. Chauré : "AR and ARMA Identification algorithms of Levinson type : An Innovations approach"
IEEE-AC-26 n° 6, 1981.

[Benveniste 82]

A. Benveniste : Exposés 1, 2, 3 dans
"Algorithmes rapides pour les systèmes dynamiques linéaires"
Vol. 2 de "Outils et Modèles Mathématiques pour l'Automatique et le Traitement du Signal"
Editions du CNRS, Paris, 1982.

[Chamberlin 71]

D.D. Chamberlin : "The single-assignement approach to parallel processing"
Procs. AFIPS Fall Joint Computer Conference, 1971.

[Comte 78]

D. Comte, G. Durrieu, O. Gelly, A. Plas, J.S. Syre : "Parallelism Control and Synchronization Expressions in a Single Assignement Language"
SIGPLAN Notices, Vol. 13, n° 1, January 1978.

[Crochière 82]

R.E. Crochière, R.V. Cox, J.D. Johnston : "Real Time Speech Coding"
IEEE-COM-30 n° 4, 1982.

[Davis 82]

A.L. Davis, R.M. Keller : "Data Flow Program Graphs"
IEEE Computer, Vol. 15, n° 2, February 1982.

[Dennis 74]

J.B. Dennis : "First Version of a Data Flow Procedure Language"
Proc. Colloque sur la Programmation (B. Robinet, Ed.), Paris, Avril 1974,
LNCS Vol. 19, Springer-Verlag.

[Falconer 80]

D.D. Falconer, V.B. Lawrence, S.K. Tewksbury : "Processor Hardware Considerations for Adaptative Digital Filter Algorithms"
Proc. ICC-1980, Seattle.

[Gerber 77]

R. Gerber, Y. Quenec'hdu, Y. Thomas : "Interactive systems as aids to teaching Automatic Control"
Proc. IFAC Symposium on Trends in Automatic Control Education, Barcelona, 1977.

[Hankin 81]

C.L. Hankin, H.W. Glaser : "The Data Flow Programming Language CAJOLE. An Informal Introduction"
SIGPLAN Notices, Vol. 16, n° 16, July 1981.

[Hoare 78]

C.A.R. Hoare : "Communicating Sequential Processes"
Comm. ACM, Vol. 21, n° 8, August 1978.

[Kahn 74]

G. Kahn : "The Semantics of a Simple Language for Parallel Programming"
Proc. IFIP Congress 74 (J.L. Rosenfeld, Ed.), 1974.

[Le Baron 79]

J.P. Le Baron : "Conception Assistée en Automatique"
Thèse Docteur-Ingénieur, Université de Rennes, 1979.

[McCarthy 66]

J. McCarthy et al. : "LISP 1.5 Programmer's Manual"
MIT Press., 1966.

[McGraw82]

J.R. McGraw : "The VAL Language : Description and Analysis"
ACM Trans. on Programming Languages and Systems, Vol. 4, n° 1,
January 1982.

[Milner 78]

R. Milner : "Synthesis of Communicating Behaviour"
Proc. Mathematical Foundations of Computer Science (J. Winkowski, Ed.),
Zakopane, 1978
LNCS Vol. 64, Springer-Verlag.

[Milner 80]

R. Milner : "A Calculus of Communicating Systems"
LNCS Vol. 92, Springer-Verlag, 1980.

[Pagan 81]

F.G. Pagan : "Formal Specification of Programming Languages, a panoramic primer"
Prentice-Hall, 1981.

[Treleaven 82]

P.C. Treleaven, D.R. Brownbridge, R.P. Hopkins : "Data-Driven and Demand-Driven Computer Architecture"
Computing Surveys, Vol. 14, n° 1, March 1982.

[Tribolet 79]

J.M. Tribolet, R.E. Crochière : "Frequency Domain Coding of Speech"
IEEE-ASSP-27, n° 5, 1979.

Liste des Publications Internes IRISA

PI 150 **Construction automatique et évaluation d'un graphe d'«implication» issu de données binaires, dans le cadre de la didactique des mathématiques**
H. Rostam , 112 pages ; Juin 1981

PI 151 **Réalisation d'un outil d'évaluation de mécanismes de détection de pannes]-]Projet Pilote SURF**
B. Decouty, G. Michel, C. Wagner, Y. Crouzet , 59 pages ; Juillet 1981

PI 152 **Règle maximale**
J. Pellaumail , 18 pages ; Septembre 1981

PI 153 **Corrélation partielle dans le cas « qualitatif »**
I.C. Lerman , 125 pages ; Octobre 1981

PI 154 **Stability analysis of adaptively controlled not-necessarily minimum phase systems with disturbances**
Cl. Samson , 40 pages ; Octobre 1981

PI 155 **Analyses d'opinions d'instituteurs à l'égard de l'appropriation des nombres naturels par les élèves de cycle préparatoire**
R. Gras , 37 pages ; Octobre 1981

PI 156 **Récursion induction principle revisited**
G. Boudol, L. Kott , 49 pages ; Décembre 1981

PI 157 **Loi d'une variable aléatoire à valeur R^+ réalisant le minimum des moments d'ordre supérieur à deux lorsque les deux premiers sont fixés**
M.Kowalowka, R. Marie , 8 pages ; Décembre 1981

PI 158 **Réalisations stochastiques de signaux non stationnaires, et identification sur un seul échantillon**
A. Benveniste J.J. Fuchs , 33 pages ; Mars 1982

PI 159 **Méthode d'interprétation d'une classification hiérarchique d'attributs-modalites pour l'«explication» d'une variable ; application à la recherche de seuil critique de la tension artérielle systolique et des indicateurs de risque cardiovasculaire**
B. Tallur , 34 pages ; Janvier 1982

PI 160 **Probabilite stationnaire d'un réseau de files d'attente multiclasse à serveur central et à routages dépendant de l'état**
L.M. Le Ny , 18 pages ; Janvier 1982

PI 161 **Détection séqentielle de changements brusques des caractéristiques spectrales d'un signal numérique**
M. Basseville, A. Benveniste , pages ; Mars 1982

PI 162 **Actes regroupés des journées de Classification de Toulouse (Mai 1980), et de Nancy (Juin 1981)**
I.C. Lerman , 304 pages ;

PI 163 **Modélisation et Identification des caractéristiques d'une structure vibratoire : un problème de réalisation stochastique d'un grand système non stationnaire**
M. Prévosto, A. Benveniste, B. Barnouin , 46 pages ; Mars 182

PI 164 **An enlarged definition and complete axiomatization of observational congruence of finite processes**
Ph. Darondeau , 45 pages ; Avril 1982

PI 165 **Accès vidéotex à une banque de données médicales**
A. Chauffaut, M. Dragone, R. Rivoire, J.M. Roger , 25 pages ; Mai 1982

PI 166 **Comparaison de groupes de variables définies sur le même ensemble d'individus**
B. Escofier, J. Pages , 115 pages ; Mai 1982

PI 167 **Transport en circuits virtuels internes sur réseau local et connexion Transpac**
M. Tournois, R. Trépos , 90 pages ; Mai 1982

PI 168 **Impact de l'intégration sur le traitement automatique de la parole**
P. Quinton , 14 pages ; Mai 1982

PI 169 **A systolic algorithm for connected word recognition**
J.P. Banâtre, P. Frison, P. Quinton , 13 pages ; Mai 1982

PI 170 **A network for the detection of words in continuous speech**
J.P. Banâtre, P. Frison, P. Quinton , 24 pages ; Mai 1982

PI 171 **Le langage ADA : Etude bibliographique**
J. André, Y. Jégou, M. Raynal , 12 pages ; Juin 1982

PI 172 **Comparaison de groupes de variables : 2ème partie : un exemple d'application**
B. Escofier, J. Pajès , 37 pages ; Juillet 1982

PI 173 **Unfold-fold program transformations**
L. Kott , 29pages ; Juillet 1982

PI 174 **Remarques sur les langages de parenthèses**
J.M. Autebert, J. Beauquier, L. Boasson, G. Senizergues , 20 pages ; Juillet 1982

PI 175 **Langages de parenthèses, langages N.T.S. et homomorphismes inverses**
J.M. Autebert, L. Boasson, G. Senizergues , 26 pages ; Juillet 1982

PI 176 **Tris pour machines synchrones ou Baudet Stevenson revisited**
R. Rannou , 26 pages ; Juillet 1982

PI 177 **Un nouvel algorithme de classification hiérarchique des éléments constitutifs de tableau de contingence basé sur la corrélation**
B. Tallur , Juillet 1982 ;

PI 178 **Programmes d'analyse des re'sultats d'une classification automatique**
I.C. Lerman et collaborateurs , 79 pages ; Septembre 1982

PI 179 **Attitude à l'égard des mathématiques des élèves de sixième**
J.Degouys, R. Gras, M. Postic , 29 pages ; Septembre 1982

PI 180 **Traitements de textes et manipulations de documents : bibliographie analytique**
J. André , 20 pages ; Septembre 1982

* épuisées

PI 181 **Algorithme assurant l'insertion dynamique d'un processeur autour d'un réseau à diffusion et garantissant la cohérence d'un système de numérotation des paquets global et réparti**
Annick Le Coz, Hervé Le Goff, Michel Ollivier , 31 pages ; Octobre 1982

PI 182 **Interprétation non linéaire d'un coefficient d'association entre modalités d'une juxtaposition de tables de contingence**
Israël César Lerman , 34 pages ; Novembre 1982

PI 183 **L'IRISA vu à travers les stages effectués par ses étudiants de DEA (1è année de thèse)**
Daniel Herman , 41 pages ; Novembre 1982

PI 184 **Commande non linéaire robuste des robots manipulateurs**
Claude Samson , 52 pages ; Janvier 1983

PI 185 **Dialogue et représentation des informations dans un système de messagerie intelligent**
Philippe Besnard, René Quiniou, Patrice Quinton, Patrick Saint-Dizier, Jacques Siroux, Laurent Trilling , 45 pages ; Janvier 1983

PI 186 **Analyse classificatoire d'une correspondance multiple ; typologie et régression**
I.C. Lerman , 54 pages ; Janvier 1983

PI 187 **Estimation de mouvement dans une sequence d'images de télévision en vue d'un codage avec compensation de mouvement**
Claude Labit , 132 pages ; Janvier 1983

PI 188 **Conception et réalisation d'un logiciel de saisie et restitution de cartes élémentaires**
Eric Sécher , 45 pages ; Janvier 1983

PI 189 → sur le point de paraître

PI 190 **Généralisation de l'analyse des correspondances à la comparaison de tableaux de fréquence**
Brigitte Escofier , 35 pages ; Mars 1983

PI 191 **Association entre variables qualitatives ordinales «nettes» ou «floues»**
Israël-César Lerman , 42 pages ; Mars 1983

PI 192 } sur le point de paraître
PI 193 }

PI 194 **Régime stationnaire pour une file M/H/1 avec impatience**
Raymond Marie et Jean Pellaumail , 8 pages ; Mars 1983

PI 195 **SIGNAL : un langage pour le traitement du signal**
Paul Le Guernic, Albert Benveniste, Thierry Gautier , 49 pages ; Mars 1983

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