Transformations du graphe des programmes SIGNAL

Olivier Maffeis, Bruno Chéron, Paul Le Guernic

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 de Recherche

1992

25ème anniversaire

N° 1574

Programme 2
Calcul Symbolique, Programmation
et Génie logiciel

TRANSFORMATIONS DU GRAPHE DES PROGRAMMES SIGNAL

Olivier MAFFEÏS
Bruno CHÉRON
Paul LE GUERNIC

Janvier 1992

RR_1574

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX FRANCE
Tél. : 99 84 71 00 - Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Transformations du graphe des programmes SIGNAL

Olivier MAFFEÏS Bruno CHÉRON Paul LE GUERNIC

IRISA/INRIA, Campus de Beaulieu
35042 Rennes Cedex

Publication Interne n°619 - Novembre 1991 - 82 pages - Programme 2

Résumé : Dans ce document, nous présentons les concepts essentiels introduits pour la mise en œuvre des programmes SIGNAL, de la séquentialité vers le parallélisme.
La première partie du document se concentre sur la mise en œuvre séquentielle; elle explicite la méthode de production de code séquentiel hiérarchisé, les optimisations fonctionnelles effectuées et les différents modes de communication avec l'environnement.
Ensuite, nous décrivons un modèle général de calcul des transformations de graphes. Ce modèle est utilisé pour définir l'évaluation paresseuse en SIGNAL et le calcul de certains concepts de la dernière partie.
Cette dernière partie présente quelques transformations de graphes pour la mise en œuvre parallèle de SIGNAL. Outre la définition d'une structure permettant l'étude d'ordonnancements au niveau macro-parallèle, elle définit deux concepts importants. Le premier, appelé *enrichissement*, spécifie un critère d'ordonnancement qualitatif, donc indépendant de l'architecture cible. De plus, ce critère apporte une aide certaine à l'obtention d'un mécanisme de compilation séparée pour SIGNAL.
Le second concept définit un partitionnement de graphe afin de prendre en compte l'asynchronisme de l'environnement dans la mise en œuvre synchrone de SIGNAL. Ce dernier concept doit fournir une mise en œuvre maximalement séquentielle des processus dédiés à un processeur.

Transformations of Graphs from SIGNAL Programs

Abstract : In this report, we present the essential concepts introduced for SIGNAL implementation, from sequentiality towards parallelism.
The first part of this report deals with sequential implementation; it explains the generation method of hierarchized sequential code, the functional optimizations and the different ways of communication with the environment.
Then, we describe a general model to specify graph transformations. This model is used to define the lazy execution scheme for SIGNAL and to compute some concepts supplied in the last part of that document.
This last part presents some graph transformations for SIGNAL parallel implementation. As well as the definition of a structure for coarse-grain scheduling studies, this document defines two important concepts. The first one, called *enhancement*, specifies a qualitative scheduling criterion, thus an independent one for target architectures. Besides, that criterion is useful to provide SIGNAL with a separated compilation mechanism.
The second concept defines a graph partitioning to take in account the asynchronism of the environment in SIGNAL synchronous implementation. This last concept provides a maximally sequential implementation for processes which are attributed to a processor.

Table des Matières

Chapitre 1

SIGNAL : un langage temps-réel synchrone

Lors de la conception et la réalisation d'applications temps-réel, deux approches se dégagent relativement à la notion de temps :

(1) l'approche traditionnelle *asynchrone* qui utilise une notion chronométrique du temps. L'indéterminisme inhérent à l'asynchronisme introduit des difficultés de test et de preuve des programmes dans lesquels la gestion du temps est à la charge du programmeur. Ces insuffisances ont conduit à l'élaboration de la seconde approche.

(2) l'approche *synchrone* qui possède une vision chronologique du temps donne un sens à la simultanéité. Dans le cadre du temps-réel, cette approche est enrichie en considérant les temps de calcul comme nuls, ce qui est acceptable si les temps effectifs d'exécution sont bornés : la vérification des contraintes sur la durée d'exécution s'effectue alors à la mise en œuvre.

    Dans l'ensemble des langages temps-réel synchrones, deux classes se détachent. La classe des langages fondés sur les automates, celle-ci comprend ESTEREL[3] et, sous une forme graphique, les STATECHARTS[12]. La seconde classe contient les langages de style déclaratif et orientés flot de données que sont, LUSTRE[5] et SIGNAL[8, 1]. Ce document se propose d'expliciter certains éléments du compilateur du langage SIGNAL.

Le compilateur SIGNAL comporte deux phases. La première synthétise les propriétés temporelles des programmes, en vérifie la correction et finalement, construit leur représentation interne sous la forme d'un graphe dynamique. La seconde phase du compilateur SIGNAL réalise le passage des graphes dynamiques vers leur mise en œuvre.

Une présentation informelle des éléments de base du langage SIGNAL est réalisée dans la section I.1 (p. 2). La section suivante contient la description succincte des mécanismes mis en œuvre dans la première phase de compilateur.
La section I.3 (p. 5) définit précisément la représentation interne des programmes SIGNAL sur laquelle sont exprimés tous les concepts décrits dans ce document. Ce chapitre se termine par la présentation de la structure du document.

I.1 Définition

Le langage SIGNAL est un langage parallèle conçu pour la programmation d'applications temps-réel. C'est un langage synchrone, flot de données, de style déclaratif, dans lequel les objets manipulés sont des signaux : un signal **x** dénote une séquence infinie de valeurs $\{x_t\}$ où t est index de temps.

Un programme SIGNAL est un système d'équations décrivant des relations temporelles et fonctionnelles entre les signaux ; chaque signal est défini par une expression formant un processus élémentaire. Le programme complet est obtenu par composition de ces expressions.
Les processus élémentaires se répartissent en deux classes : les processus monochrones et ceux polychrones. Après la présentation des processus élémentaires selon cette répartition, nous décrirons les deux opérateurs structurels que sont, la *composition* et la *restriction.*

Les processus monochrones
Les processus monochrones sont constitués des opérateurs qui nécessitent la manipulation d'une seule référence temporelle i.e tous les signaux intervenant dans ces opérateurs ont la même horloge (ils sont synchrones).

- *Les fonctions instantanées.*

  Ces fonctions sont les extensions aux séquences (signaux) des fonctions classiques sur les éléments des séquences ; elles constituent l'ensemble des processus élémentaires *statiques monochrones.*

  L'expression SIGNAL

  $$\mathbf{y} := f\ (\mathbf{x1}, \ldots, \mathbf{xn})$$

  définit un processus élémentaire tel que

  $$\forall t \quad y_t = [\![f]\!](x1_t, \ldots, xn_t)$$

- *Le décalage temporel.*

  Il permet l'expression du passé et constitue le processus élémentaire *dynamique monochrone*. L'expression SIGNAL

  $$\mathbf{y := x\ \$1}$$

  définit la relation ci-dessous dans laquelle la valeur initiale $v0$ est associée à la déclaration de **y**

  $$\forall t > 1 \quad y_t = x_{t-1},\ \ y_0 = v0$$

Les processus polychrones

Les opérateurs polychrones manipulent plusieurs références temporelles ; ils permettent de définir des signaux non synchrones avec les signaux incidents à l'opérateur.

- *Le sous-échantillonnage.*

  L'opérateur *when* spécifie le sous-échantillonnage de signaux par une condition booléenne. L'expression SIGNAL (**c** étant un signal booléen, **x** et **y** deux signaux de même type)

  $$\mathbf{y := x\ when\ c}$$

  signifie : **y** est défini et égal à **x** quand **x** est défini et que **c** porte la valeur vraie ; sinon **y** est indéfini. L'horloge du signal **y** est le sous-échantillonnage de celle de **x** par celle définie sur la condition "**c** porte la valeur vraie"

- *Le mélange prioritaire.*

  Cet opérateur permet la fusion de signaux ; l'horloge du signal produit est l'union des horloges des signaux incidents. L'expression SIGNAL

  $$\mathbf{x := u\ default\ v}$$

  signifie que **x** est défini et égal à **u** si celui-ci est défini ; **x** est égal à **v** sinon.

La composition

L'opérateur de composition réalise la création de nouveaux processus par composition des systèmes d'équations définissant les processus cités ; il est associatif et commutatif.
La composition de deux processus SIGNAL $P1$ et $P2$, notée

$$P1 \mid P2$$

désigne un nouveau processus où les signaux de même nom dans $P1$ et $P2$ correspondent aux mêmes signaux (liens de communication entre $P1$ et $P2$).

La restriction
Cet opérateur compose de nouveaux processus par confinement de signaux définis dans un processus (notion de portée d'un signal).
L'expression SIGNAL

$$P1/\mathbf{a}_1,\ldots,\mathbf{a}_n$$

définit un nouveau processus dont les communications sont celles de $P1$ exceptées $\mathbf{a}_1,\ldots,\mathbf{a}_n$.

Un exemple de programme SIGNAL définissant une horloge avec remise à zéro (sous-ensemble d'un chronomètre) est présenté dans l'annexe B (p. 63).

I.2 Compilation

La compilation des programmes SIGNAL, du texte source à leur représentation interne, comporte trois principaux mécanismes.

- **La modélisation des synchronisations.**

  La modélisation des synchronisations est l'élément de la compilation qui formalise les contraintes d'un programme SIGNAL sous la forme d'un système d'équations. Elle est constituée de trois parties : le codage des propriétés temporelles et logiques du processus SIGNAL en équations sur le corps commutatif $\mathbf{Z}/3\mathbf{Z}$ (entiers modulo 3) ; la résolution du système d'équations et la hiérarchisation du contrôle [4]. Les tables A.1 (p. 59) et A.2 (p. 60) définissent l'encodage en équations sur $\mathbf{Z}/3\mathbf{Z}$.

- **La construction du graphe aux dépendances conditionnées**

  Le caractère flot de données du langage SIGNAL induit la représentation d'un processus sous la forme d'un réseau statique orienté. L'existence de signaux ayant des horloges différentes impose la notion de *dépendance conditionnée.*

  Ce graphe représente les dépendances instantanées entre les différents objets du programme : les signaux [14]. Les dépendances entre ces nœuds sont étiquetées par des fonctions de $\mathbf{Z}/3\mathbf{Z}$ dans le treillis booléen qui modélisent les conditions auxquelles ces dépendances sont effectives. La table A.3 (p. 60) définit l'encodage en dépendances conditionnées.

- **La synthèse du graphe dynamique**

  Les deux structures obtenues, la hiérarchie des horloges et le graphe aux dépendances conditionnées, synthétisent respectivement les relations, temporelles et fonctionnelles, des programmes SIGNAL ; la représentation interne, appelée *Graphe Dynamique* [2], est définie par leur fusion.

I.3 Représentation interne

La représentation interne doit synthétiser, dans une structure unique, l'ensemble des constituants des programmes SIGNAL. Nous allons procéder à la description de cette représentation, selon la progression décrite dans la section I.2 (p. 4), en précisant successivement : la modélisation du contrôle, la modélisation des dépendances, et les visions possibles de la fusion de ces deux éléments de représentation.

I.3.1 Modélisation du contrôle

L'expression des comportements des processus élémentaires s'exprime en terme de *présence/absence* de valeurs sur les signaux mais aussi en terme de valeurs booléennes (voir la définition du sous-échantillonnage). L'ensemble fini $\mathcal{F}_3=\mathbf{Z}/3\mathbf{Z}$ des entiers modulo 3 permet, dans une algèbre unique, le codage des booléens et des propriétés d'horloge comme suit.

vrai	:	1
faux	:	-1
absent	:	0
présent	:	1

Un signal booléen b est codé par une variable b ; b^2 représente son horloge : l'horloge d'un signal x sera dénotée x^2. Ce codage s'effectue dans l'anneau $< \mathcal{F}_3, +, ., 0_h >$ où 0_h représente l'horloge nulle i.e l'horloge qui ne définit aucune occurrence.

L'ensemble des relations sur booléens ou horloges d'un programme SIGNAL définit un ensemble Σ d'équations ; ces équations de $\mathcal{F}_3$ font intervenir un ensemble de variables X : $\Sigma \subset \mathcal{F}_3[X]$. Le codage des contraintes induites par les processus élémentaires est donné par les tables A.1 (p. 59) et A.2 (p. 60).
Cet ensemble Σ permet de vérifier des propriétés statiques de cohérence [4], ainsi que des propriétés dynamiques (voir [13] qui comporte la structure détaillée de Σ).

En prenant $\mathcal{F}_3$ pour base, les horloges sont des fonctions de $\mathcal{F}_3$ à valeur dans $\{0,1\}$. L'ensemble C des horloges possibles d'un programme se définit par :

$$C = \left\{ f : \mathcal{F}_3[X] \rightarrow \{0,1\} \right\}$$

Sur C, une relation d'ordre peut être définie. $< C, \vee, . >$ est un treillis :

$$h^2 \vee k^2 = (h^2 + k^2)^2 = h^2 + k^2 - h^2.k^2$$

$$\begin{aligned} h^2 \leq k^2 &\Leftrightarrow h^2 \vee k^2 = k^2 \\ &\Leftrightarrow h^2.k^2 = h^2 \end{aligned}$$

Ce treillis est représenté en arbre sur les sous-échantillonnages par condition booléenne (voir [4] pour la justification de cette structure). Soit H un sous-ensemble de C, la *hiérarchie d'horloges* sur H est un arbre $< H, \Theta >$ dans lequel l'application Θ définit la notion de père.

La racine de l'arbre est notée h_0, elle définit l'horloge la plus rapide de H. Dans le cas où h_0 est une horloge du programme, ce dernier est dit *endochrone*. Dans le cas contraire plusieurs maxima (*horloges maîtresses locales*) existent et dénotent une définition partielle des synchronisations du programme ; le programme associé est déclaré *exochrone*.

Pour exemple, la hiérarchie d'horloges de l'horloge avec remise à zéro (sous-ensemble d'un chronomètre) est présentée dans l'annexe B (p. 69).

I.3.2 Graphe aux dépendances conditionnées

Le graphe de dépendances associé à un processus P est quadruplet $G =< N, \Gamma, I, O >$ où :

- $< N, \Gamma >$ est un graphe orienté dont tout sommet est identifié par le nom du signal qu'il définit.
- $I \subset N$ l'ensemble des entrées.
- $O \subset N$ l'ensemble des sorties.

Un graphe aux dépendances conditionnées $< G, h_N, h_\Gamma >$est constitué d'un graphe de dépendance $G =< N, \Gamma, I, O >$ et de deux applications h_N et h_Γ telles que :

$$\begin{array}{l} h_N : N \rightarrow C \\ h_\Gamma : \Gamma \rightarrow C \end{array} \quad \text{avec} \quad \forall i, j \in N \quad h_\Gamma(i,j) \leq h_N(i).h_N(j)$$

Elles représentent respectivement les horloges des signaux et celles des dépendance ; elles définissent les conditions d'existence des éléments du graphe. La hiérarchie d'horloges se fonde sur un ensemble $H = \mathcal{I}m(h_N) \cup \mathcal{I}m(h_\Gamma)$ où $\mathcal{I}m(\varphi)$ dénote l'ensemble image par l'application φ. La table A.3 (p. 60) définit les applications h_N et h_Γ.

Le graphe aux dépendances conditionnées de l'horloge à remise à zéro (sous-ensemble d'un chronomètre) pris en exemple est donné dans l'annexe B (p. 68).

I.3.3 Graphes dynamiques

La structure interne des programmes SIGNAL est la concaténation des structures précédemment définies.

Définition 1.1 (Graphe dynamique) $< G, X, \Sigma, < H, \Theta >, h_N, h_\Gamma >$ *est un graphe dynamique si et seulement si :*

- $< G, h_N, h_\Gamma >$ *est un graphe aux dépendances conditionnées.*
- X *un ensemble de variables.*
- $\Sigma \subset \mathcal{F}_3[X]$ *un ensemble de contraintes.*
- $< H, \Theta >$ *un arbre d'horloges ;* $H \subset \mathcal{C} = \{f : \mathcal{F}_3[X] \rightarrow \{0,1\}\}$ *;* H *contient* $\mathcal{I}m(h_N) \cup \mathcal{I}m(h_\Gamma)$

De cette structure hétérogène mêlant contrôle et dépendances de données, deux visions coexistent ; celles-ci permettent, selon l'objectif, soient des manipulations structurées, soient des transformations de graphes.

- **Projection du graphe sur la hiérarchie**

  La projection du graphe sur la hiérarchie s'obtient par partitionnement du graphe sur l'application h_N puis, par attachement des sous-graphes de nœuds synchrones à leur horloge. La figure I.1 contient une représentation hiérarchique du graphe dynamique obtenue par projection du graphe sur la hiérarchie ; G_i dénote le sous-graphe tel que pour tout nœud n de G_i, $h_N(n) = h_i$. Le graphe dynamique d'un sous-ensemble du chronomètre est donné, sous sa forme hiérarchique, dans l'annexe B (p. 70).

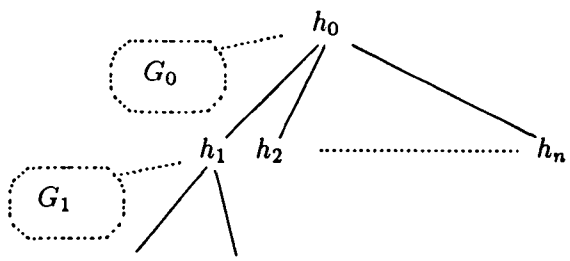


Figure I.1 : Vision hiérarchique d'un graphe dynamique

Cette représentation du graphe dynamique d'un programme SIGNAL reste clairement double et hiérarchique ; elle est largement utilisée dans la traduction vers les langages de programmation qui nécessitent une structuration ; ni la décompilation

en SIGNAL des graphes dynamiques, ni la production de code séquentiel (présentée dans le chapitre 2 (p. 11)) n'enfreignent cette constatation.

La décompilation SIGNAL, qui utilise cette vision hiérarchique des graphes dynamiques, se structure par imbrication de processus : chaque processus `H_i_H` représente le sous-graphe dynamique ayant h_i pour racine d'arborescence, donc il contient comme sous-processus les processus `H_j_H` des horloges h_j filles de h_i.

Le corps du processus `H_i_H` comporte :

- la décompilation du sous-graphe G_i ;
- la décompilation, en SIGNAL, des polynômes qui définissent les horloges filles de h_i ;
- la synchronisation entre le signal `h_i_h` associé à h_i et les signaux identifiant les nœuds de G_i ;
- la référence aux sous-processus `H_j_H` des horloges h_j telles que $\Theta(h_j) = h_i$.

La décompilation en SIGNAL de la représentation interne de horloge à remise à zéro (sous-ensemble d'un chronomètre) est présenté en l'annexe B (p. 71).

- **Enchevêtrement de la hiérarchie et du graphe.**

L'enchevêtrement de la hiérarchie dans le graphe est la représentation inverse de la précédente. H contient les relations d'horloges et, comme le montre la décompilation des graphes dynamiques en SIGNAL, H peut être représenté par des expressions SIGNAL, donc par un graphe de dépendances.

L'enchevêtrement de la hiérarchie et du graphe s'opère en représentant H par graphe de dépendances définit selon les sous-expressions polynomiales communes ; la hiérarchie définie sur H est conservée.

La représentation interne ainsi obtenue est plus homogène : les deux types d'objets du programme (signaux et horloges) appartiennent à un unique graphe. L'homogénéité de cette structure autorise une expression simplifiée, des manipulations formelles et des propriétés, sur l'ensemble du graphe dynamique des programmes SIGNAL ; les transformations des graphes dynamiques, exprimées dans les chapitres 3 (p. 31) et 4 (p. 41), utilisent cette seconde vision des graphes dynamiques.

I.4 A partir des graphes dynamiques

La présentation des concepts de ce document est structurée en trois chapitres.

- La méthode de production de code séquentiel est présentée dans le chapitre 2 (p. 11).

  Cette présentation est répartie suivant les aspects temporels et fonctionnels du langage. La mise en œuvre des aspects temporels de SIGNAL est exprimée par le parcours du graphe (voir section II.2 (p. 12)). Pour les aspects fonctionnels, seules sont présentées les optimisations réalisées. Ces optimisations concernent la gestion des processus dynamiques dans la section II.3 (p. 16) et la méthode de suppression des instructions stériles dans la section II.4 (p. 22).

  En SIGNAL, le problème synchronisme/asynchronisme est de deux ordres : la prise en compte de l'asynchronisme de l'environnement dans l'acceptation des entrées au monde synchrone de SIGNAL, ce sujet est traité en section II.5 (p. 26) ; la mise en œuvre synchrone des programmes SIGNAL exochrones, des traitements sont esquissés dans la section II.6 (p. 28).

- Un modèle général de propriétés est défini dans le chapitre 3 (p. 31).

  Les transformations du graphe s'expriment en algèbres de chemins ayant une structure de dioïdes. Ce chapitre présente la notion de dioïdes et le parcours d'évaluation associé.

  Cette présentation du modèle est suivie d'un exemple d'utilisation dans la section III.3 (p. 37) : l'évaluation paresseuse dans le langage SIGNAL.

- Trois procédés qui préfigurent la mise en œuvre parallèle du langage sont décrits dans le chapitre 4 (p. 41).

  - La réduction du graphe aux entrées/sorties permet d'obtenir une vision synthétique des programmes. La structure obtenue s'utilise lors de l'implantation parallèle et lors de la compilation séparée, deux problèmes très liés en SIGNAL. La modélisation de ce concept est réalisée par l'intermédiaire des dioïdes.
  - Le renforcement du graphe permet de définir des séquentialisations qui préservent l'absence de circuits quel que soit le contexte d'utilisation. Ce renforcement autorise, dans certain cas, une composition "aveugle" de processus séquentialisés
  - Le partitionnement de processus en lignées selon les entrées/sorties ce qui permet de prendre en compte différentes stratégies (dynamique) d'ordonnancement. De même que les réductions de graphes aux entrées/sorties, les lignées sont modélisées par les dioïdes.

Ce document conclut sur un ensemble de perspectives de développement des concepts présentés dans le chapitre 4 (p. 41). Les tables d'encodage de SIGNAL pour définir la structure interne des programmes sont données en annexe A (p. 59). En annexe B (p. 63), un exemple d'horloge minutes-secondes avec remise à zéro (sous-ensemble d'un chronomètre) est donné. Cette annexe présente le processus complet de sa compilation, du programme source, à son code séquentiel FORTRAN ; elle contient non seulement les fichiers produits par le compilateur SIGNAL, mais également des schémas qui illustrent la modélisation interne de cette horloge.

Chapitre 2

Production de code séquentiel

L'objectif de ce chapitre est d'expliciter les mécanismes utilisés dans la traduction des graphes dynamiques vers un langage impératif. Le graphe dynamique source peut être celui d'un programme SIGNAL, mais également celui d'un processus résultant d'un partitionnement.
Le code obtenu suppose un contexte d'exécution mono-processeur ; il constitue, dans un contexte multi-processeur, une première approche de l'exécution d'un processus dédié à un processeur, en particulier dans le cas de processus procéduraux[1] (voir section IV.4 (p. 53)). De plus, les performances du code séquentiel fourni une base de mesure lors d'une implantation parallèle.

La première section de ce chapitre caractérise les programmes SIGNAL qui peuvent être soumis à la production de code séquentiel. La méthodologie utilisée dans la mise en œuvre des aspects temporels est définie dans la seconde section. Les troisième et quatrième sections présentent les principales optimisations de mise en œuvre des aspects fonctionnels de SIGNAL. Les mécanismes d'interfaçage des programmes SIGNAL avec les environnements synchrones ou asynchrones sont décrits la section II.5 (p. 26). La sixième et ultime section de ce chapitre essaye de dégager une méthode pour obtenir l'exécution des programmes SIGNAL *exochrones*, ceux dont les synchronisations ne sont pas totalement spécifiées.

II.1 Hypothèses

La production de code séquentiel à partir d'un graphe de dépendances passe par un *tri topologique* de ce dernier, l'ordre total obtenu définit l'ordre séquentiel d'exécution des sommets. Or, un tri topologique existe si et seulement si le graphe est sans circuit.
Toutefois, la présence de circuits dans les graphes dynamiques, n'est pas synonyme d'interblocage : les dépendances étant conditionnées, un circuit dans le graphe peut ne jamais être effectif. De tels circuits, appelés *faux-circuits*, surviennent en présence de conditions

[1] Ces processus commencent leur exécution lorsque toutes leurs entrées sont disponibles

exclusives sur les arcs d'un circuit ; leur mise en œuvre impose l'ordonnancement dynamique de tâches.

Hypothèse de graphe
les graphes dynamiques soumis à la production de code séquentiel sont supposés sans circuits ni faux-circuits[2]

Deux types de programmes, endochrone et exochrone, peuvent être dégagés selon leur hiérarchie d'horloges. L'endochronisme se définit par la présence d'une racine h_0 qui appartient au programme ; la hiérarchie étant structurée par inclusion d'instants, h_0 est l'horloge la plus rapide : elle définit le cadencement du programme.
Dans le cas exochrone, l'horloge h_0 n'appartient pas au programme ; elle est indéterminée et un complément d'information contextuelle est nécessaire afin d'obtenir l'exécution de tels programmes. L'hypothèse énoncée ci-dessous sera levée (dans certains cas) grâce aux traitements décrits en section II.6 (p. 28).

Hypothèse d'horloge
les graphes dynamiques soumis à la production de code séquentiel sont supposés endochrones

Alors que le monde séquentiel d'exécution peut être considéré comme synchrone, l'environnement est clairement asynchrone. Une mise en œuvre asynchrone doit pouvoir être à l'écoute de l'environnement pour ordonner l'exécution selon les arrivées d'entrées ou en fonction de la demande de sorties.
En l'absence d'ordonnancement dynamique, nous émettons l'hypothèse d'environnement.

Hypothèse d'environnement
les entrées sont supposées disponibles dès qu'elles sont nécessaires

Cette condition d'environnement sera levée grâce à au mécanisme de partitionnement de graphes dynamiques en lignées (cf la section IV.4 (p. 53)). Ce partitionnement s'effectue au niveau des points d'articulation nécessitant un ordonnancement dynamique ; la partition obtenue se compose de processus qui vérifient l'hypothèse d'environnement.

II.2 Méthode de décompilation

La mise en œuvre séquentielle utilise la vision hiérarchique des graphes dynamiques constituée d'une hiérarchie et d'un graphe. L'implémentation séquentielle suppose une efficacité de mise en œuvre de ces deux éléments ; l'étude sur la hiérarchie d'horloges constitue le

[2]Il est toujours possible de ramener un faux-circuit à un graphe sans faux-circuits par duplication de nœuds.

sujet de cette section.

Les horloges conditionnent les éléments des graphes ; elles constituent le contrôle de l'exécution. Pour exemple, prenons le graphe dynamique présenté en figure II.1-b ; ce graphe est induit par le programme de type PASCAL en figure II.1-a.

Figure II.1 : Une conditionnelle et son graphe dynamique

Réciproquement, le contrôle des programmes SIGNAL (modélisé par l'étiquetage des dépendances par des horloges) se met en œuvre, dans un langage impératif classique, avec des conditionnelles.
De plus, chaque sommet étant conditionné, la traduction est encapsulée dans une conditionnelle sur son horloge. En exemple, la traduction de l'opérateur **default** est donnée en figure II.2. Comme $h_N(x) = h_\Gamma(u,x) \vee h_\Gamma(v,x)$, la traduction de **x := u default v** ne fait pas intervenir de condition sur $h_\Gamma(v,x)$: elle est induite avec le **else**.

Figure II.2 : Le graphe de dépendance et la traduction de **x := u default v**

Une telle traduction de chacun des sommets du graphe de dépendances conditionnées en définit une mise en œuvre séquentielle simple. Considérons la mise en œuvre des évaluations des conditions : la mise en œuvre des horloges.

la hiérarchie est structurée sur l'inclusion d'instants ; soient h^2 et k^2 deux horloges telles

que $h^2 \geq k^2$. Les instants où h^2 est évaluée à 0, k^2 peut être déduite comme valant 0. Ainsi, afin de réduire les évaluations d'horloges, le contrôle est mis en œuvre par imbrication de conditionnelles : k^2 n'est évaluée que si h^2 vaut 1.

à chaque instant, une horloge est évaluée si et seulement si ses ancêtres dans l'arborescence sont validés (évalués à 1).

En contrepartie, comme toute évaluation d'un signal est précédée d'une conditionnelle sur son horloge, elle s'en trouve également précédée de conditionnelles sur les ancêtres de son horloge : le code obtenu est structuré par emboîtement de conditionnelles.

II.2.1 Critère d'efficacité

Le contrôle étant traduit en cascades de conditionnelles liées au parcours de la hiérarchie, sa mise en œuvre performante s'exprime par la minimisation du cheminement sur l'arborescence dans le tri topologique recherché.

Intuitivement, le coût de cheminement sur l'arborescence entre deux nœuds de la hiérarchie s'exprime par la distance de chacun d'eux à leur ancêtre commun. Formellement, soient h_i,h_j deux horloges et h_k leur ancêtre commun minimum, le cheminement associé est défini par :

$$\begin{aligned} C_{h_i,h_j} &= d(h_j,h_k) + d(h_i,h_k) && (1)\\ &= 2.p(h_k) - p(h_i) - p(h_j) && (2) \end{aligned}$$

Dans l'égalité (1), d est une fonction de distance : $\Theta^n(h_j) = h_k \Rightarrow d(h_j,h_k) = n$.
L'égalité (2) est une réécriture de (1) où p est la fonction de profondeur définie par :

$$\Theta^n(h_j) = h_0 \iff p(h_j) = n$$

Soient deux éléments e_i et e_j du graphe dynamique, l'extension de la formule (2) à l'ensemble du graphe s'exprime par la formule (3).

$$C_{e_i,e_j} = 2.p'(h_k) - p'(e_i) - p'(e_j) \qquad (3)$$

Dans cette formule, h_k est l'ancêtre de plus grande profondeur qui est commun à e_i, e_j ; p' est la fonction de profondeur étendue à tous les éléments du graphe :

$$\begin{aligned} e_i \in H &\Rightarrow p'(e_i) = p(e_i)\\ e_i \in N &\Rightarrow p'(e_i) = p(h_N^{-1}(e_i)) \end{aligned}$$

Pour la production de code séquentiel, le parcours de graphes dynamiques doit fournir un tri topologique des éléments des graphes tel que $C = \sum C_{e_i,e_{i+1}}$ est minimal (e_i, e_{i+1} sont deux éléments successifs du tri).

II.2.2 Parcours Hiérarchique

Le coût C_{e_i,e_j} avec e_i et e_j synchrone est nul et par cela minimal. Déterminer le tri topologique optimal se réduit, au vue de cette remarque, à déterminer la suite d'horloges en privilégiant le choix local.

Deux types d'algorithmes permettent d'obtenir ce résultat : un algorithme d'évaluation et de séparation successive à solution optimale ou un algorithme glouton fournissant une "bonne" solution. La second type d'algorithme fut choisi pour des raisons de performances de compilation et de simplicité.

Le parcours va consister à extraire itérativement l'ensemble des nœuds sans prédécesseurs. A un stade du parcours, plusieurs horloges possèdent des nœuds sans prédécesseur alors qu'aucun nœud de ce type est présent à l'horloge courante ; la qualité de la solution fournie repose entièrement sur le choix probabiliste de l'horloge suivante dans l'ensemble des horloges ayant des nœuds "libres".

Le critère de choix retenu est de privilégier la descente sur la remontée et, dans la phase descendante, de choisir le sous-arbre de poids maximum en nombre de nœuds libres. Le choix probabiliste de privilégier la descente sur la remontée permet un moindre marquage; ce dernier qui est utilisé dans l'évaluation du critère s'effectue de l'horloge des signaux libérés jusqu'à l'ancêtre commun avec l'horloge courante.
En exemple, la figure II.3 dans laquelle n_2 et n_3 sont des sommets libérés par l'extraction de n_1 illustre ce marquage.

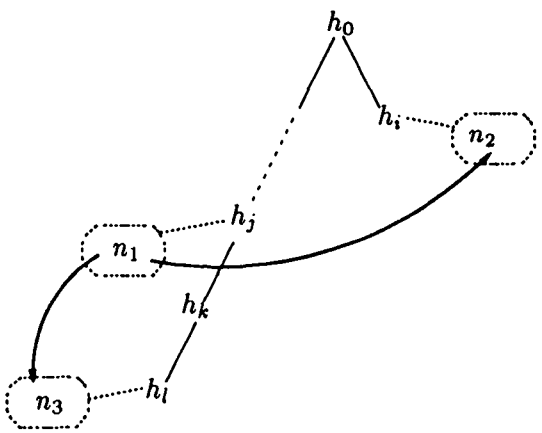


Figure II.3 : Le parcours du graphe dynamique : un exemple de marquage

Lors de la libération de n_2, un marquage jusqu'à h_0 suffit pour connaître l'existence d'un

nœud libre dans le sous-arbre de racine h_i. De même, à la libération du nœud n_3, seules les horloges h_k et h_j sont marquées. Quand tous les nœuds libres de l'horloge h_i sont extraits, les attributs attachés aux filles de h_i sont parcourues à la recherche de l'horloge ayant le plus de nœuds libres dans son sous-arbre. Si aucun nœud n'est libre dans le sous-arbre de racine h_i, la recherche est entreprise en remontant dans l'arbre.

Algorithme de parcours

A chaque horloge h est associé un compteur $cpt(h)$ des nœuds libres dans le sous-arbre de racine h. Le parcours est initialisé par l'extraction de h_0 qui définit l'horloge courante *hcourante*. Ensuite, ce parcours itère sur les cas (1), (2a) et (2b).

(1) $cpt(hcourante) > 0$

- Extraire un nœud libre e de *hcourante* et décrémenter $cpt(hcourante)$.
- Pour chaque nœud e' qui devient sans prédécesseur par l'extraction de e, soit h l'ancêtre commun de plus grande profondeur entre *hcourante* et $h_N^{-1}(e')$. Les compteurs des horloges h_i telles que $h_N^{-1}(e) \leq h_i < h$ sont incrémentés.

(2) $cpt(hcourante) = 0$

(a) Descendant : $\exists h_j \in \Theta^{-1}(hcourante)$ tel que $cpt(h_j) > 0$
Définir *hcourante* par l'horloge h_j dont le compteur possède la valeur maximale.

(b) Ascendant : $\forall h_j \in \Theta^{-1}(hcourante)$ tel que $cpt(h_j) = 0$
Définir l'horloge courante par $\Theta(hcourante)$.

II.3 Processus dynamiques

Dans la section précédente, nous avons donné (en figure II.2) une présentation intuitive de la mise en œuvre de l'opérateur *default*. Cette section présente la méthode utilisée pour la mise en œuvre séquentielle, dans un contexte de mémoire globale, des processus dynamiques `y := x$ 1` ou plutôt de leur généralisation :

$$\mathbf{y} := \mathbf{x}\ \$r\ \mathbf{window}\ f$$

Ce processus signifie que **y** contient les valeurs de **x** avec un retard de r et une fenêtre de vision de taille f ; le processus de base est défini par $r = 1$ et $f = 1$.

La structure interne qui modélise les relations entre les retards est arborescente.

A un ensemble de signaux retardés à partir de **x** :

$$\begin{aligned} \mathbf{zx_{i_1}} &:= \mathbf{x}\ \$r_{i_1}\ \mathbf{window}\ f_{i_1} \\ \mathbf{zx_{i_1,\ldots,i_n}} &:= \mathbf{zx_{i_1,\ldots,i_{n-1}}}\ \$r_{i_1,\ldots,i_n}\ \mathbf{window}\ f_{i_1,\ldots,i_n} \end{aligned}$$

nous associons l'arbre des retards dérivés de **x** que la figure II.4 représente schématiquement.

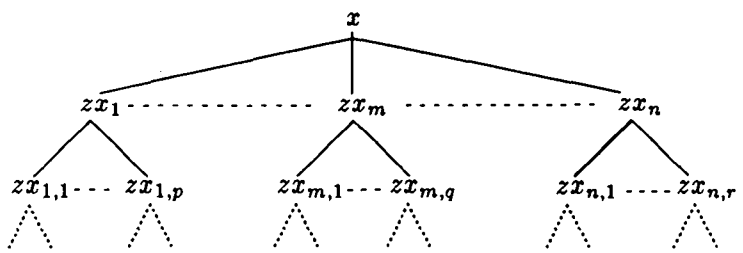


Figure II.4 : L'arbre des retards

Cet arbre constitue la structure permettant le calcul de la taille de la zone cible et des valeurs d'initialisations des indices. La suite de la présentation se compose de trois parties :

- la présentation de la méthode utilisée à l'aide d'exemples significatifs de complexité croissante ;
- la généralisation des formules obtenues dans l'étude des exemples ;
- les différents problèmes de compatibilité d'initialisation rencontrés dans la construction des arbres de retards.

II.3.1 Exemples et lignes directrices

L'étude va porter sur trois cas : le premier est relatif à la gestion de retards de profondeur 1 ; le deuxième concerne la gestion des retards généralisés (avec fenêtre coulissante) au premier niveau ; le dernier cas présente la gestion des retards de profondeur supérieur à 1.

- **zx** := **x** $\$r$

  Par définition du retard :

  $$\forall t \quad zx_t = x_{t-r}$$

  Sa mise en œuvre nécessite donc la mémorisation de $r+1$ valeurs pour représenter **x** et **zx** dans le même vecteur. Ils sont gérés dans un *vecteur* V de taille n supérieure ou égale à $r+1$. Un indice est associé à chaque signal, il est incrémenté modulo n avant

l'utilisation du signal (l'incrémentation de l'indice en début d'instant est justifiée par des possibilités d'agrégations plus importantes, voir section II.4 (p. 22)). **x** est stocké dans $V[i_x]$.

Schématiquement, nous obtenons donc, à un instant t, le vecteur présenté dans la figure II.5

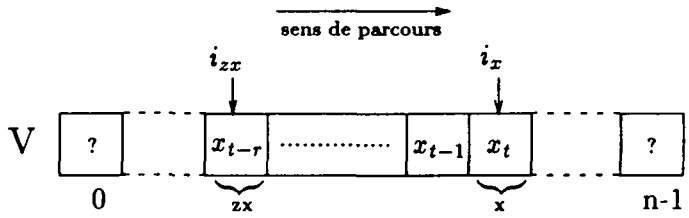


Figure II.5 : Représentation interne d'un vecteur de retards

Dans une telle représentation, pour connaître la valeur de **zx** à l'instant t, il suffit d'accéder à $V[i_{zx}]$ avec $i_{zx} = i_x - r$. Lors de la mise en œuvre, il suffit d'initialiser i_{zx} à $i_x - r$, et d'incrémenter ces indices une fois par instant de présence des signaux (**x** et **zx** sont synchrones) : le décalage entre les indices reste constant.

Nous obtenons donc le tableau récapitulatif II.1.

Expression SIGNAL	—	**zx := x $r**
Définition	—	$zx_t = x_{t-r}$
Relation d'indices	—	$i_{zx} = i_x - r$
Références	—	$zx \rightarrow V[i_{zx}]$
Taille minimale de V	—	$r + 1$

Tableau II.1 : Tableau récapitulatif des relations pour **zx := x $r**

- **zx := x $r window** f avec $f > 1$

Dans cet exemple, f est supposé supérieur à un pour se différencier du premier cas. Par définition de la fenêtre temporelle, nous avons ($zx[i]$ dénote le $i^{ième}$ élément du tableau zx) :

$$\forall i \in [1, f] \quad zx_t[i] = x_{t-(r+f-i)}$$

La définition de **zx** impose de conserver (r+f) valeurs (de l'instant $t - (r + f - 1)$ à t). En suivant la même démarche que précédemment, nous obtenons le tableau de mise en œuvre II.2. La représentation en vecteur est modélisée par la figure II.6

Expression SIGNAL	—	**zx:= x $r window f**
Définition	—	$zx_t[i] = x_{t-(r+f-i)}$
Relation d'indices	—	$i_{zx} = i_x - (r+f)$
Références	—	$zx[i] \rightarrow V[(i_{zx}+i) \bmod n]$
Taille minimale de V	—	$r+f$

Tableau II.2 : Tableau récapitulatif des relations pour **zx:= x $r window f**

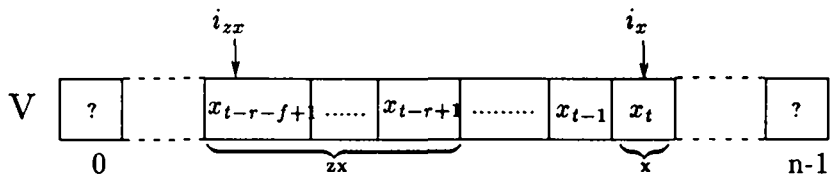


Figure II.6 : La gestion indiciaire d'un retard généralisé

Remarque : L'ajout d'une fenêtre coulissante sur une signal entraîne l'ajout (pour le signal retardé) d'une dimension vis à vis du signal origine. Dans la gestion indiciaire des retards, la fenêtre coulissante est aplanie, et représentée dans une seule dimension.

- **zx** := **x** **$**$r$ **window** f
  zzx := **zx** **$**$rr$ **window** ff

  Dans cet exemple, nous allons étudier les répercussions de l'ajout d'un retard de profondeur deux. De même que précédemment, l'ajout de ce retard se répartie en deux cas distincts : le tableau II.3 contient les relations obtenues pour $ff = 1$; le tableau II.4 récapitule celles pour $ff > 1$

II.3.2 Méthode

Soit, un ensemble de signaux retardés à partir de **x** :

$$\begin{aligned} \mathbf{zx}_{i_1} &:= \mathbf{x}\ \$r_{i_1}\ \mathbf{window}\ f_{i_1} \\ \mathbf{zx}_{i_1,\ldots,i_n} &:= \mathbf{zx}_{i_1,\ldots,i_{n-1}}\ \$r_{i_1,\ldots,i_n}\ \mathbf{window}\ f_{i_1,\ldots,i_n} \end{aligned}$$

Par définition du retard :

$$\begin{aligned} f_{i_1,\ldots,i_p} = 1 &\Rightarrow zx_{(i_1,\ldots,i_p)\ t} = zx_{(i_1,\cdots,i_{p-1})\ t-r_{i_1,\ldots,i_p}} \\ f_{i_1,\ldots,i_p} > 1 &\Rightarrow zx_{(i_1,\ldots,i_p)\ t}[j] = zx_{(i_1,\cdots,i_{p-1})\ t-(r_{i_1,\ldots,i_p}+f_{i_1,\ldots,i_p}-j)} \end{aligned}$$

	— $f = 1$	— $f > 1$
Définition	— $zzx_t = zx_{t-rr}$ $= x_{t-(r+rr)}$	— $zzx_t[j] = zx_{t-rr}[j]$ $= x_{t-(r+f+rr-j)}$
Indices	— $i_{zzx} = i_z x - rr$ $= i_x - (r + rr)$	— $i_{zzx} = i_z x - rr$ $= i_x - (r + f + rr)$
Références	— $zzx \rightarrow V[i_{zzx}]$	— $zzx[j] \rightarrow V[(i_{zzx} + k) \bmod n]$
Taille min. V	— $r + rr + 1$	— $r + f + rr$

Tableau II.3 : Tableau récapitulatif des relations pour $ff = 1$

	— $f = 1$	— $f > 1$
Définition	— $zzx_t[j] = zx_{t-(rr+ff-j)}$ $= x_{t-(r+rr+ff-j)}$	— $zzx_t[k,j] = zx_{t-(rr+ff-k)}[j]$ $= x_{t-(r+f-j+rr+ff-k)}$
Indices	— $i_{zzx} = i_z x - (rr + ff)$ $= i_x - (r + rr + ff)$	— $i_{zzx} = i_z x - (rr + ff$ $= i_x - (r + f + rr + ff)$
Références	— $zzx[j] \rightarrow V[(i_{zzx} + j) \bmod n]$	— $zzx[k,j] \rightarrow V[(i_{zzx} + j + k) \bmod n]$
Taille min. V	— $r + rr + ff$	— $r + f + rr + ff - 1$

Tableau II.4 : Tableau récapitulatif des relations pour $ff > 1$

L'initialisation des indices dénote le décalage constant entre l'indice de la racine et celui des signaux retardés.
Par récurrence, on prouve l'équivalence suivante :

$$i_{zx_{i_1,\ldots,i_p}} = i_x - (r_{i_1} + \cdots + r_{i_1,\ldots,i_p} + f^*_{i_1} + \cdots + f^*_{i_1,\ldots,i_p})$$

avec :

$$f^*_{i_1,\ldots,i_p} = \begin{cases} 0 & \text{si } f_{i_1,\ldots,i_p} = 1 \\ f_{i_1,\ldots,i_p} & \text{sinon} \end{cases}$$

Intuitivement, la taille minimale du vecteur utilisé est définie par le maximum de décalage possible entre tous les signaux pris deux à deux. Par une preuve par récurrence, on démontre l'équivalence suivante définissant la taille minimale du vecteur nécessaire à la représentation d'un arbre de retards :

$$Max(r_{i_1} + \cdots + r_{i_1,\ldots,i_p} + f_{i_1} + \cdots + f_{i_1,\ldots,i_p} - p) + 1$$

La référence à : $zx_{i_1,\ldots,i_p}[k_1,\ldots,k_q]$ (avec $q \leq p$)
s'effectue dans : $V[i_{(zx_{i_1,\ldots,i_p}} + k_1 + \cdots + k_q) \bmod n)]$

Remarque : Dans le cas d'arbre de retard ne représentant qu'un retard du type **zx := x $1**, la gestion indiciaire devrait, dans un tableau à deux éléments, incrémenter deux indices i_x et i_{zx} à chaque instants de présence des signaux. Dans ce cas, la gestion indiciaire introduit un surcoût vis à vis de la recopie de valeurs ; cette dernière sera donc choisie pour la mise en œuvre de tels retards.

II.3.3 Initialisation

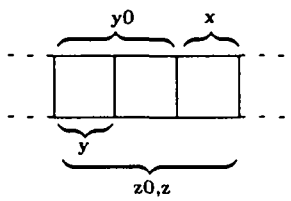
La gestion indiciaire des retards regroupe, dans un seul vecteur, l'ensemble des signaux d'un arbre de retards et, de ce fait, réalise des superpositions de zones.
La gestion indiciaire des signaux d'un arbre impose la compatibilité des initialisations lors des superpositions de zones ; celle-ci est vérifiée lors de la construction des arbres, les signaux aux initialisations incompatibles sont gérés par recopie.

Les différents modes de superposition sont illustrés à l'aide de l'exemple ci-dessous.

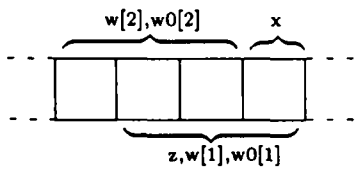
y := x $2	initialisé par	$y0[2]$
z := x window 3	initialisé par	$z0[2]$
w := z window 2	initialisé par	$w0[2,3]$

Les modes de superposition sont au nombre de deux :

- **horizontal.** La zone réservée aux sous-arbres d'un nœud est contigue" à celle du nœud donc, les zones de mémorisation des sous-arbres d'un nœud se chevauchent : il est nécessaire de vérifier la compatibilité d'initialisation entre les sous-arbres frères. Dans l'exemple, ce mode de superposition correspond à celui entre $y0$ et $z0$.



- **zone de projection.** Ce cas survient pour un retard de profondeur supérieur à deux, alors qu'il existe au moins deux fenêtrages sur ses ancêtres dans l'arbre. La réduction en dimensions entre la taille de l'initialisation et la zone de représentation du signal est égale au nombre de fenêtrages sur ses ancêtres. Il suffit d'une initialisation uniforme (à valeur constante) pour obtenir la compatibilité avec la zone de projection. Dans l'exemple, ce mode de superposition correspond à celui par $w0$.



Précédemment, une gestion des processus dynamiques par recopie de valeurs a été mise en œuvre. Dans le cadre du développement d'un poste de traitement de la parole continue basé sur la programmation synchrone [15], l'application d'une gestion indiciaire des retards a permis de réduire par huit, le temps d'exécution du module de segmentation. Ce gain est à mettre en relation avec les manipulations importantes de données effectuées par ce module ; de telles manipulations sont très fréquentes dans les applications de traitement de signal qui constituent le domaine privilégié de SIGNAL.

II.4 Agrégations de signaux

L'agrégation de signaux constitue la seconde optimisation des manipulations de données. La première visait un gestion efficace des processus dynamiques de SIGNAL, cette seconde optimisation porte sur l'ensemble des manipulations de données induites par les processus élémentaires.
Cette optimisation, exprimée par le concept d'agrégation, réalise des optimisations spatiales du code engendré et induit quelques optimisations temporelles.

Sur une architecture du type Von NEUMANN, la mise en œuvre s'effectue en attribuant à chaque signal, un emplacement mémoire dans lequel ses valeurs en cours d'instant sont conservées (pour exemple, voir la figure II.2 (p. 13) qui donne la traduction du *default*). Afin de réduire la taille mémoire nécessaire, nous souhaitons pourvoir attribuer un même emplacement mémoire à deux signaux différents. La vérification de l'absence de conflit de mémorisation doit précéder ces superpositions de zones.
Les deux cas suivants vérifient l'absence de conflits entre deux signaux.

- *Leurs utilisations sont disjointes.*

  Par exemple, deux signaux **x** et **y** peuvent être confondus si, dans un instant logique, l'évaluation du signal **y** succède aux utilisations de **x**. Le graphe de données exprimant les dépendances *instantanées* entre les signaux, cette propriété de disjonction sera vérifiée sur des structures du graphe : l'agrégation ainsi réalisée est appelée *agrégation sur graphe* et décrite dans la sous-section II.4.2.

- *Leurs utilisations simultanées sont compatibles.*

  Si une agrégation sur graphe de **x** et **y** n'a pu être réalisées, c'est qu'au cours d'un même instant logique, des utilisations entrelacées de ces signaux peuvent survenir. Dans ce cas, seule une compatibilité des valeurs des signaux peut induire une absence de conflits ; l'agrégation ainsi obtenue est appelée *sur signaux* et développée dans la sous-section II.4.1.

II.4.1 Agrégations sur signaux

L'agrégation sur signaux doit vérifier une compatibilité des valeurs lors des utilisations entrelacées de signaux. Pour obtenir l'entrelacement, les signaux concernés doivent posséder des occurrences simultanées ; cette remarque induit la définition 2.1 d'agrégation sur signaux.

Définition 2.1 (Agrégation sur signaux) **x** *et* **y** *sont agrégés sur signaux si et seulement si :*

$$x \square_s y \quad \equiv \quad \forall t \quad h_N(x)_t = 1 \wedge h_N(y)_t = 1 \Rightarrow x_t = y_t \tag{4}$$

Cette relation *non transitive* est difficilement vérifiable entre des signaux quelconques. Pour cela, nous ne considérons que les deux cas extrêmes de relations d'horloges : disjonction et inclusion.

- *Disjonction.*

  Les horloges de **x** et **y** sont exclusives : $\forall t \quad h_N(x)_t = 1 \wedge h_N(y)_t = 1$ n'est jamais vérifiée. Dans ce cas, les valeurs signaux **x** et **y** sont représentées dans la même zone ; l'optimisation obtenue est spatiale.

- *Inclusion*

  Les horloges de **x** et **y** sont liées par une relation d'ordre ; supposons $h_N(x) \leq h_N(y)$:

$$h_N(x) \leq h_N(y) \quad \Longrightarrow \quad [\forall t \quad h_N(x)_t = 1 \Rightarrow h_N(y)_t = 1] \tag{5}$$

  La conjonction des équations (4) et de (5) se réécrit en l'équation (6). Cette équation définit une relation d'ordre.

$$x <_\square y \quad \equiv \quad \forall t \quad h_N(x)_t = 1 \Rightarrow h_N(y)_t = 1 \wedge x_t = y_t \tag{6}$$

  Des ordres étant directement induits par les processus élémentaires, vérifions statiquement les possibilités d'agrégation sur ces derniers.

  - `y := f(x).`
    Dans ce cas $h_N(x) = h_N(y)$ donc, **x** s'agrège sur **y** si et seulement si **f** est l'identité.
  - `y := x when c.`
    La sémantique du sous-échantillonnage induit $h_N(y) \leq h_N(x)$ donc, $y <_\square x$ dans le sous-échantillonnage présenté.

- **y** := **u default v**.
  Cette expression induit les ordres $h_N(y) \geq h_N(u)$ et $h_N(y) \geq h_N(u)$ sur les horloges. Le **default** étant une fusion prioritaire, L'agrégation $u <_{\square} y$ est directement déduite.
  Par contre, seulement si $h_N(u)$ et $h_N(v)$ sont exclusives, les relations $v <_{\square} y$, $u <_{\square} v$ et $v <_{\square} u$ sont vérifiées et alors : **u**, **v** et **y** sont associés au même emplacement mémoire.

Ce type d'agrégation permet évidemment une optimisation spatiale de la mise en œuvre des programmes SIGNAL mais également une optimisation temporelle. En effet, la mise en œuvre d'un signal nécessite un calcul induit de son expression de définition puis, sa mémorisation ; lors de la présence simultanée des signaux agrégés, seuls un calcul et une mémorisation sont nécessaires : le gain est alors temporel.
Pour exemple d'optimisation temporelle, considérons le processus **y := u default v**. Celui-ci se met en œuvre par la recopie de la valeur de u dans zone mémoire associée à y ; les zones de représentation de u et de y étant superposées, cette recopie est caduque.

II.4.2 Agrégations sur le graphe

L'absence d'entrelacement dans les références à deux signaux indique que les utilisations de l'un précèdent la définition du second : les références sont séquentielles.
Par exemple, considérons le compteur définit ci-dessous.

```
( |  Y  :=  ZY  +  1
  |  ZY  :=  Y  $1
  | )/  ZY
```

Lors de sa mise en œuvre, les zones de représentation de **y** et **zy** sont confondues : l'agrégation sur graphe de **y** et **zy** est possible ; l'optimisations obtenue est spatiale. Dans notre modèle de graphe, cette séquentialité s'exprime par la définition 2.2.

Définition 2.2 (Agrégation sur graphe) *Soient* **x** *et* **y** *deux signaux.* **x** *et* **y** *sont agrégés sur graphe, dénoté par* $x \square_g y$, *si et seulement si :*

$$\forall t \quad h_N(x)_t = 1 \wedge h_N(y)_t = 1 \quad \Longrightarrow \quad \begin{cases} et \; h^*_\Gamma(x,y) \neq 0_h \\ \forall z \;\; h^*_\Gamma(x,z)_t = 1 \Rightarrow h^*_\Gamma(z,y)_t = 1 \end{cases} \tag{7}$$

Dans cette équation, h^*_Γ dénote la fermeture transitive de h_Γ :

$$h^*_\Gamma(x,y) = \bigvee_{p=(x,z_1,\ldots,z_n,y)} h_\Gamma(x,z_1).\cdots.h_\Gamma(z_n,y)$$

Une définition progressive de cette expression de fermeture transitive est fournie dans la section IV.2 (p. 43) par les équations (16) et (17. Cette section contient également l'algorithmique associée à son évaluation.

D'après l'équivalence (5), l'équation (7) se réécrit en (8).

$$\forall t \quad h_N(x)_t = 1 \wedge h_N(y)_t = 1 \quad \Longrightarrow \quad \left\{ et \begin{array}{l} h^*_\Gamma(x,y) \neq 0_h \\ \forall z \;\; h^*_\Gamma(x,z) \leq h^*_\Gamma(z,y) \end{array} \right. \tag{8}$$

Par une démonstration inductive, nous réécrivons l'équation (7) en l'équation (9).

$$\forall t \quad h_N(x)_t = 1 \wedge h_N(y)_t = 1 \quad \Longrightarrow \quad \left\{ et \begin{array}{l} h^*_\Gamma(x,y) \neq 0_h \\ \forall z \in \Gamma(x) \;\; h_\Gamma(x,z) \leq h^*_\Gamma(z,y) \end{array} \right. \tag{9}$$

II.4.3 Processus CELL : un exemple

A partir du noyau des processus élémentaires SIGNAL, de nombreux autres processus exprimables en SIGNAL de base ont été insérés afin de faciliter la programmation ; le processus **cell** qui réalise un procédé de mémorisation infinie en est un exemple.
Le processus **y := x cell c** est défini en SIGNAL de base par :

```
( | synchro {Y, (event X) default (when C)}
  | Y := X default ZY
  | ZY := Y $1
  | )/ ZY
```

Dans ce processus, l'utilisation de **zy** est unique d'où la possibilité de fusion : $y \; \square_g \; zy$. Concernant l'agrégation de **x** et **y**, celle-ci est obtenue par le *default* : $x <_\square y$. Le processus d'agrégation n'est pas transitif donc, il convient de vérifier l'agrégation de **x** et de **zy**.

Les signaux **zy** et **x** ne sont pas exclusifs quand à leur définition et n'ont aucune raison d'être égaux au niveau des valeurs. Leur agrégation est-elle définitivement impossible ? En réalité **zy** n'est utilisé que pour définir **y** en exclusion avec **x** : l'horloge d'utilisation de **zy** est $h_\Gamma(zy, y)$.
Les horloges d'utilisation définies formellement dans la section III.3 (p. 37) permettent une réduction en fréquence des évaluations des signaux relativement à leurs utilisations. Sur les horloges d'utilisation, les signaux **zy** et **x** sont exclusifs; par l'extension des définitions 4 et 7 aux horloges d'utilisation (les occurrences de h_N sont subtituées par h_U), nous obtenons : **zy**$\square$**x**.
Les signaux du processus **cell** sont comparables deux à deux par le procédé d'agrégation : **x**, **y** et **zy** sont représentables dans une même zone mémoire lors de leur implémentation.

II.5 Communications

SIGNAL modélise des processus réactifs donc, outre les aspects temporels et fonctionnels, une importante caractéristique de sa mise en œuvre réside dans la prise en compte des communications avec l'environnement.

Deux types d'environnements peuvent exister : synchrone ou asynchrone. L'asynchronisme du monde réel (et parallèle) se distingue du monde synchrone où, grâce à la simultanéité, l'ensemble des événements est totalement ordonné. Notons qu'une architecture séquentielle est considérée comme un monde synchrone : les événements sont linéairement ordonnés.

II.5.1 Vers l'asynchrone

Un signal **x** dénote une suite $\{x_t\}_{t>1}$ de valeurs auxquelles un index de temps t est attaché. La racine de l'arborescence en tant qu'horloge la plus rapide définit le cadencement du programme associé : l'ensemble des index de temps est inclus dans ce référentiel temporel. Ainsi, tout index d'un signal **x** est représenté par une variable booléenne h_x qui vaut vrai si l'instant logique courant appartient à son l'index de temps.
Dans le cas général, tout signal d'interface **x** est totalement spécifié par un couple $< h_x, v_x >$ dans lequel v_x est la valeur de **x** à l'instant ou h_x vaut vrai (sinon v_x est indéfini). Un tel couple $< h_x, v_x >$ doit être déterminé à chaque instant logique que définit la racine.

Les entrées

Considérons un programme SIGNAL dont la racine de l'arborescence est une horloge du programme (dans le cas contraire, se reporter à la section II.6) ; cette racine définit le cadencement du programme. Les autres horloges sont déterminées par des sous-échantillonnages de cette racine : les horloges sont définies à l'intérieur du programme.

Donc seules les valeurs des signaux d'entrée proviennent de l'environnement. Les deux caractéristiques suivantes en détermine le mode de communication.

(1) *Non rémanence.*

La fugacité des valeurs impose la réalisation d'une interface ; celle-ci est simplement constituée de files FIFO dont la taille peut être raisonnablement bornée. En effet, le programme de consommation des entrées est temps-réel : son temps d'exécution est borné de manière à éviter un engorgement de ces files d'attente.

(2) *Ordre d'arrivée variable.*

Afin d'obtenir une exécution gouvernée par les entrées, il est nécessaire de disposer de primitives de communication non bloquantes.

Ainsi, la lecture de la valeur d'un signal **x** est réalisée par une procédure d'interface :

$$\text{read } (v_x, p_x)$$

avec : p_x = vrai $\Rightarrow$ la valeur du signal **x** est déterminée
p_x = faux $\Rightarrow$ v_x est indéterminée

Ces procédures sont utilisées dans un automate de scrutation des entrées ; elles sont appelées itérativement sur l'ensemble des signaux présents dont la valeur est indéterminée. La détermination d'une valeur réalise l'activation de la *lignée* d'entrée associée (voir la section IV.4 (p. 53) dans laquelle le concept de lignée est défini)

Sorties

L'émission des sorties consiste à délivrer les valeurs des signaux de sorties, les horloges associées ainsi que la base de cadencement des horloges : l'horloge racine. Des interfaces de sorties dégénérées peuvent être désirées. En particulier, pour certains environnement, seule l'émission des valeurs est nécessaire.

II.5.2 Vers le synchrone

L'utilisation de communications avec un environnement synchrone permet, tout en restant dans l'environnement du langage SIGNAL, d'étendre les possibilités de manipulation de données ainsi qu'apporter un aide au développement dans ce langage.
Les communications synchrones, obtenues par *appels externes*, recouvrent deux types de synchronisme.

- *Fonctions* ou *le synchronisme monochrone*

  Afin d'étendre infiniment ses possibilités de manipulations de données monochrones, SIGNAL a été doté d'un mécanisme permettant l'appel de fonctions instantanées externes telles que : `sin, cos, fft`. Comme pour les fonctions instantanées prédéfinies, l'appel d'une fonction externe réalise une synchronisation des signaux de l'interface.

  Les quelques caractéristiques techniques suivantes ont été remarquées pour la mise en œuvre de ces fonctions.

  - *Absence d'état interne.*
    Les instances d'une telle fonction externe peuvent être ré-ordonnées, dupliquées, regroupées sans modification sémantique ; pour exemple, `sin, cos` appartiennent à cette catégorie contrairement à `random`.

- *Cohérence déclaration/utilisation*

  Outre les informations pragmatiques de mise en œuvre — le type de l'appel (fonction, procédure,...), le mode de passage des paramètres (tableaux [0..n] ou [1..n], passage par nom ou par référence), et les types des paramètres (entier, entier long,...) —, la cohérence d'utilisation de ces fonctions doit être vérifiée. En particulier, les entrées passées par référence, ne doivent pas être modifiées sous peine d'effets de bords proscrits dans un langage flot de données.

- *Processus* ou *le synchronisme polychrone*

  Les processus externes sont des processus évoluant dans un environnement synchrone et manipulant plusieurs références temporelles. En particulier, de tels processus peuvent être des processus SIGNAL prédéfinis (compilation séparée) ou distants (processus SIGNAL placé sur un processeur différent de l'architecture parallèle).

  Les communications des signaux d'interface sont composées des valeurs et des horloges telles l'interface asynchrone d'émission des sorties. Leur description s'effectue par une structure d'interface conditionnée ; cette structure est définie dans la section IV.2 (p. 43).

Pour terminer, notons qu'il existe un chevauchement entre la notion d'E/S du programme et celle d'appel externe. En effet, il est possible de masquer des E/S du programme par l'utilisation d'appels externes afin d'obtenir un mode d'E/S vers des procédés disponibles dès qu'ils sont requis : sortie sur écran, lecture fichier, etc.

II.6 Extension aux programmes exochrones et hétérochrones

La hiérarchie d'horloge est structurée par sous-échantillonnage sur conditions ; une horloge qui ne peut être définie en terme d'un sous-échantillonnage par conditions définit une horloge maîtresse.
Quand une seule horloge maîtresse résulte de la hiérarchisation, celle-ci constitue la racine et représente l'indispensable référentiel temporel pour considérer le programme associé comme endochrone.
Afin d'obtenir un arbre, une racine h_0 est ajoutée ; celle-ci n'étant pas une horloge du programme, le référentiel temporel du programme n'est pas correctement défini et le programme associé est déclaré exochrone.

Soit $\mathcal{M} = \{h_1, \ldots, h_n\}$ l'ensemble des horloges maîtresses d'une hiérarchie de racine h_0 :

$$\forall i \in [1..n] \quad \Theta(h_i) = h_0$$

La hiérarchie ayant le sous-échantillonnage par booléens pour structuration, ces horloges maîtresses sont définies par :

$$\forall i \in [1..n] \quad h_i = h_0(-\varphi_i - \varphi_i^2) \quad \text{avec} \quad h_0 = \varphi_i^2$$

Pour toute horloge maîtresse h_i, un signal booléen φ_i synchrone à h_0 est introduit en entrée du programme. Comme h_0 est l'horloge des signaux booléens φ_i, elle devient une horloge de signaux du programme et peut-être prise comme référentiel.

Ainsi, la transformation d'un programme exochrone en un programme se réalise par l'ajout de signaux booléens d'entrée qui sont synchrones à l'horloge la plus rapide h_0.
Malheureusement, cette simple transformation n'est pas suffisante si les horloges maîtresses sont contraintes. Les processus qui possèdent une telle caractéristique mêlent l'endochronisme et exochronisme; ces processus sont appelés *hétérochrones*. Pour exemple, considérons le compteur qui réalise de manière non synchrone, une incrémentation de sa valeur interne et la réinitialisation par **a**.

```
process ASYNC_COUNT =
        { ? integer A
          ! integer X}
        (| ZX := X $1
         | X := A default ZX+1
         |)/ ZX
        where
              integer ZX init 0
end
```

Ce processus hétérochrone de comptage définit le calcul d'horloge présenté dans la table II.5. Sur les horloges, cette table exprime que le signal **x** est plus fréquent que **a** : $h_2 = h_1 \vee h_2$.

Horloge	Synchro	Définition	Relation d'horloge
h_1	a^2		
h_2	x^2, zx^2	$a^2 + zx^2(1-a^2)$	$h_1 \vee h_2$

Tableau II.5 : Le calcul d'horloge d'un compteur hétérochrone

Ainsi, la caractérisation d'un programmes SIGNAL hétérochrone s'effectue par l'ensemble $\mathcal{M}$ de ses horloges maîtresses et par l'ensemble les contraintes de Σ qui portent sur $\mathcal{M}$; soit $\Sigma_{\mathcal{M}}$ l'ensemble de ces contraintes.

Dans l'exemple du compteur, l'ensemble réduit de contraintes est défini par Σ lui-même. Sur $< \mathcal{M}, \Sigma_{\mathcal{M}} >$, plusieurs configurations peuvent survenir :

- **Exochronisme** : $\Sigma_{\mathcal{M}} = \emptyset$

  Les valuations des φ_i sont indépendantes. L'ensemble des signaux φ_i nécessaire pour déterminer les horloges de $\mathcal{M}$ sont fournies par l'environnement à chaque instant logique défini par h_0.

- **Hétérochronisme** : $\Sigma_{\mathcal{M}} \neq \emptyset$

  L'exemple du compteur se place dans cette seconde configuration de $< \mathcal{M}, \Sigma_{\mathcal{M}} >$.

  Dans cet exemple, le treillis sur $\mathcal{M} = \{h_1, h_2\}$ possède un majorant h_2 : h_2 est l'horloge la plus rapide des horloges maîtresses. En tant qu'horloge la plus rapide, elle peut définir le cadencement du programme donc être égale à $h0$: $\varphi_2 = vrai$.

  L'interface qui permet le passage de l'hétérochronisme vers synchronisme s'effectue en ajoutant un signal d'entrée correspondant à φ_1 ; la mise en œuvre synchrone de ce programme est présentée ci-dessous.

```
process SYNC_COUNT =
        { ? logical PHIA
            integer A
          ! integer X}
        (| synchro {A, when PHIA}
         | synchro {X, PHIA}
         | {X} := ASYNC_COUNT {A}          |)
end
```

Dans le cas général, la mise en œuvre de processus hétérochrones s'avère très délicate. Dans l'exemple présenté, les valeurs de φ_1 peuvent être quelconque mais, il existe des programmes où certaines valuations des φ_i sont interdites ; en présence de valuations non autorisées, le comportement du système est à spécifier (instant ignoré, procédure de correction, etc) ce qui ne permet pas un traitement automatique de ces types de programmes. Par contre, en présence de valuations correctes, le système peut, grâce au système de contraintes, effectuer les anticipations selon les relations d'horloges ; les anticipations possibles sont définies par l'équation (5) ou sa réciproque (10).

$$h^2 \leq k^2 \quad \Longrightarrow \quad [\forall t \quad k_t^2 = 0 \Rightarrow h_t^2 = 0] \tag{10}$$

Les communications des signaux booléens φ_i introduites sont traitées comme celles des signaux de l'interface originelle. En particulier, ils introduisent des lignées supplémentaires.

Chapitre 3

Modèle général de calcul de propriétés

Les manipulations formelles des graphes dynamiques ont des objectifs multiples : la préparation à l'implantation parallèle, l'extraction d'éléments de la représentation interne en vue de la compilation séparée, ou le calcul d'optimisations pour la génération de code séquentiel.

Les trois transformations suivantes sont présentées dans ce document.

(1) La réduction du graphe dynamique aux E/S qui consiste à extraire l'information minimale du graphe dynamique d'un processus afin d'en permettre : la compilation séparée ou le calcul de propriétés pour l'implantation parallèle.

(2) La partition du graphe dynamique d'un processus en lignées. Cette partition permet la prise en compte de l'asynchronisme de l'environnement.

(3) L'évaluation paresseuse du graphe dynamique qui consiste à définir les fréquences de calculs, non plus sur la présence des entrées, mais en fonction de leur nécessité.

Ces transformations de graphes s'expriment sous forme d'algèbres de chemins qui ont une structure de dioïde [11]. Une présentation succincte des dioïdes et de leurs propriétés importantes est réalisée ci-après, dans la section III.1.
Cette homogénéité dans la modélisation des transformations de graphes en simplifie la présentation mais aussi justifie l'unicité du parcours de graphe mis en œuvre pour l'évaluation de ces algèbres de chemins ; ce parcours est présenté dans la section III.2.1. La présence de faux-circuits dans le graphe dynamique des programmes soumis à ces transformations est prise en compte dans la section qui termine la présentation du modèle général de calcul des propriétés.

La section III.3 de ce chapitre modélise, en utilisant les dioïdes, la réécriture (3). Les deux autres utilisations de ce modèle sont présentées dans le chapitre 4 en sections IV.2 (p. 43) et IV.4 (p. 53).

III.1 Dioïdes

Cette structure algébrique, apparue au cours des années 70, fut issue d'un souci de modélisation d'éléments de la théorie des graphes et ainsi, de formalisation des algorithmes appliqués ; elle fut ensuite appliquée à d'autres domaines. Le lecteur intéressé se référera à [10] pour l'ensemble de la théorie et des applications des dioïdes ; la présentation réalisée ci-dessous utilise la terminologie de la théorie des graphes. Notons que les dioïdes présentés dans ce document opèrent sur les graphes dynamiques obtenus par enchevêtrement de la hiérarchie et du graphe (cf p. 8).

Au concept mathématique de dioïde ont été adjoints des algorithmes efficaces de calcul. Ces algorithmes utilisent principalement la représentation des graphes par leur matrice d'incidence. En effet, la matrice d'incidence définit, dans le dioïde considéré, les coefficients d'un système d'équations linéaires ; les algorithmes de Warshall, Gauss-Jordan sont de fameux exemples de ce type d'algorithmes.
Dans le cas particulier du langage SIGNAL, une représentation du graphe par une matrice d'incidence est irréaliste ; la structure interne est à base de pointeurs. Ainsi, un parcours de type MARIMONT à été mis en œuvre (voir section III.2.1) ; ce parcours est enrichi d'un parcours de circuits (voir section III.2.2) afin d'obtenir un algorithme général.

III.1.1 Définition

Cette section comporte la définition des dioïdes et de leurs applications aux algèbres de chemins. L'ensemble de cette présentation est issu de [11], les démonstrations des théorèmes ou des propositions peuvent être retrouvées dans ce livre.

Définition 3.1 (dioïde) *Le dioïde $(S, \oplus, \bullet)$ est une structure algébrique définie sur un ensemble S sur lequel les trois propriétés suivantes sont vérifiées.*

(1) $(S, \oplus, \mathbb{0})$ *est un monoïde commutatif*

(2) $(S, \bullet, \mathbb{1})$ *un monoïde*

(3) $\bullet$ *possède* $\mathbb{0}$ *comme élément absorbant et, est distributif par rapport à* $\oplus$.

La transformation d'une algèbre de chemin en dioïde s'opère en définissant, pour tout graphe $G = (N, \Gamma)$, une *matrice d'incidence* $A = (a_{ij})$ et une *matrice unité* $E = (e_{ij})$.

Généralement, ces matrices sont définies par :

$$\begin{array}{llll} a_{ij} = s_{ij} & si\ (i,j) \in \Gamma & \text{et} & e_{ii} = \mathbf{1} \\ a_{ij} = \mathbb{0} & sinon & & e_{ij} = \mathbb{0} \end{array}$$

L'évaluation des dioïdes revient alors, soit au calcul de A^k (la puissance $k^{ième}$ de A), soit à celui de $A^{(k)} = E \oplus A^1 \oplus \cdots \oplus A^k$. Remarquons que l'idempotence de $\oplus$ permet, en définissant $A' = E \oplus A$, de réduire le calcul de $A^{(k)}$ à celui de A'^k.

Cette première représentation utilise les matrices d'incidence. Une seconde présentation du calcul des dioïdes s'obtient par rapport au poids des chemins.

Définition 3.2 (Poids d'un chemin) *Soit $\mu = (i_1, \ldots, i_n)$ un chemin du graphe $G =< N, \Gamma >$. Le poids de μ dans le dioïde $(S, \oplus, \bullet)$ est défini par :*

$$w(\mu) = s_{i_1 i_2} \bullet \ldots \bullet s_{i_{n-1} i_n}$$

L'ensemble des chemins formés de $k+1$ arcs (non nécessairement distincts) entre les sommets i et j est noté C_{ij}^k ;

$$C_{ij}^{(k)} = \bigcup_{l=1}^{k} C_{ij}^l$$

dénote les chemins d'au plus $k+1$ arcs.
Les deux méthodes, par puissance de matrices d'incidence ou par le poids des chemins, sont équivalentes quand au résultat : soit A_{ij}^k et $A_{ij}^{(k)}$ les éléments i, j des matrices A^k et $A^{(k)}$; ils se définissent en fonction des poids de chemins par :

$$A_{ij}^k = \sum_{\mu \in C_{ij}^k} w(\mu) \tag{11}$$

$$A_{ij}^{(k)} = \sum_{\mu \in C_{ij}^{(k)}} w(\mu) \tag{12}$$

III.1.2 Convergence

L'évaluation d'une algèbre de chemins définie sous la forme d'un dioïde revient au calcul des puissances de A ; la solution est définie par le plus petit point fixe.
Avant d'énoncer le théorème de convergence qui définit les conditions d'existence d'un point fixe dans le calcul de A^k, nous allons définir les concepts de p-régularité et de p-absorption.

Définition 3.3 (p-régularité) *Un élément a est dit p-régulier si $a^{(p)} = a^{(p+1)}$ avec $a^{(p)} = \mathbb{0} \oplus a \oplus \ldots \oplus a^p$.*

Dans cette définition, p désigne le rang pour atteindre le poids fixe dans le calcul des chemins. En fait, une p-régularité est directe pour l'ensemble des chemins sauf pour les circuits. Ainsi, la définition 3.4 permet une définition suffisante pour définir l'existence d'un point fixe dans le calcul des A^k.

Définition 3.4 (circuit pointé, p-absorption) *Soit $\mu = (i_1, \ldots, i_n, i_1)$ un circuit ; soit μ_k le circuit pointé à partir du sommet i_k : $\mu_k = (i_k, \ldots, i_n, i_1, \ldots, i_k)$. Un graphe est dit sans circuit p-absorbant si tous ses circuits pointés sont p-réguliers.*

Remarquons que la commutativité de l'opérateur $\bullet$ induit l'égalité des poids des circuits pointés d'un circuit : la commutativité de l'opérateur $\bullet$ donne un sens au *poids d'un circuit.*

Théorème 1 (Convergence) *Si G n'a pas de circuits p-absorbants et si $\bullet$ est commutatif alors, $A^{(k)}$ possède une limite quand k tend vers l'infini. Cette limite est atteinte pour $k \geq N_p$ avec, N_p qui est le nombre d'arcs du chemin le plus long dans l'ensemble des chemins ne contenant aucun circuit élémentaire plus de p fois.*

Le théorème 1 de convergence est tiré d'une famille de théorèmes du livre [11] ; ces théorèmes se basent sur différentes caractéristiques des opérateurs $\oplus$ et $\bullet$: idempotence, commutativité, etc. Le théorème 1 est celui qui inclut l'ensemble des dioïdes présentés dans ce document et qui en possède le plus fort énoncé de convergence.

III.2 Méthode d'évaluation

Des algorithmes classiques tel que celui de WARSHALL permettent le calcul de ces fermetures transitives modélisée par les dioïdes. Ces algorithmes possèdent une vision des graphes en matrice d'incidence. Avec notre représentation de graphes, une méthode plus itérative de calcul des dioïdes est nécessaire.
L'évaluation des dioïdes de ce document s'effectue sur les graphes dynamiques en représentation homogène ; cette représentation, décrite dans la section I.3 (p. 5), se compose d'un graphe unique avec une hiérarchie sous-jacente.

La présentation de cette méthode d'extraction des chemins va être scindée en deux sous-sections : la première définit un parcours des graphes quand ils sont supposés sans circuit ; la seconde complète le précédent parcours en recherchant les circuits. Cette scission est motivée par la réduction sensible des complexités algorithmiques en cas d'absence de circuits (dont la présence est exceptionnelle).

III.2.1 Parcours de graphes

Tous des dioïdes considérés dans ce document nécessitent le calcul de $A^{(k)}$ et possèdent l'idempotence de $\oplus$. Dénotons par $\pi(i,j)$ le poids de $A_{ij}^{(k)}$ pour $k \geq N_p$ et, supposons le graphe sans circuit.
L'absence de circuits impose l'existence de sommets j sans prédécesseur ($\forall i \quad A_{ij} = \mathbb{0}$) ; les chemins arrivant à ces sommets j sont de longueur maximale 0. Pour de tels sommets j, nous obtenons la formule (13) de calcul des valuations associées

$$\forall k \geq 0 \quad A_{ij}^{(k)} = e_{ij} = \pi(i,j) \tag{13}$$

Cette remarque peut être émise pour des chemins de longueur 1, 2, etc. Ainsi, la reformulation, sous une forme itérative, de l'équation (12) amène à l'équation (14).

$$\pi(i,j) = e_{ij} \oplus \sum_{k \in \Gamma_j^{-1}} \pi(i,k) \bullet s_{kj} \tag{14}$$

En l'absence de circuit, l'évaluation des $\pi(i,j)$ est réalisable sur un tri topologique des nœuds du graphe ; ce tri topologique réalise la triangulation de la matrice d'incidence du graphe associé.
L'algorithme de MARIMONT (de complexité $O(m)$ avec m le nombre d'arcs) réalise, par une méthode itérative, un tri topologique des sous-graphes sans circuits. C'est un algorithme de front qui itére l'étape (1) ci-dessous. Afin de réaliser conjointement le calcul des poids $\pi(i,j)$, l'étape (2) est ajoutée.

Algorithme d'évaluation

Un poids π_{ij} est associé à chaque arc (i,j) du graphe. L'algorithme d'évaluation itére sur ces deux étapes séquentielles. Cet algorithme se termine quand tous des sommets du graphes ont été extraits.

(1) Extraction d'un nœud j sans prédécesseur

un tel nœud existe si le graphe est sans circuit.

(2) Calcul des valuations π_{ij}, dans le dioïde considéré, selon la formule (14).

En présence de circuits, ce parcours s'arrête : l'algorithme d'extraction est exécuté et les valuations des nœuds du circuit ou des circuits sont réalisées. Le circuit ou les circuits qui ont causés l'arrêt du parcours ayant été totalement traités, le parcours peut alors redémarrer : le graphe résiduel possède de nouveau des nœuds sans prédécesseur.

III.2.2 Circuits

L'obtention d'un graphe résiduel non vide à la fin du parcours indique la présence de circuits. Le parcours consiste en un algorithme de front donc, considérons la limite constituée de l'ensemble L des nœuds du graphe résiduel pour lesquels au moins un prédécesseur a été extraits par le parcours.
Cet ensemble L est non vide : les graphes dynamiques considérés possèdent la racine de l'arborescence comme point d'entrée et sont connexes. L'arrêt du parcours précédent impose la présence de circuits ne permettant plus sa progression : il n'existe plus de sommet sans prédécesseur donc, au moins un sommet de l'ensemble L appartient à un circuit[1].

Dans la figure III.1, la limite de progression du parcours est dessinée en pointillés, l'ensemble des nœuds atteints $L = \{a, e, g\}$.

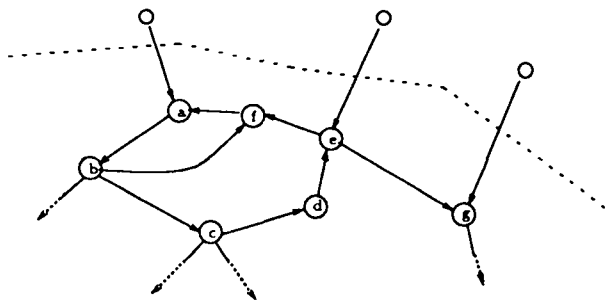


Figure III.1 : Détection des circuits

La recherche des circuits est réalisée par le développement des chemins issus de cet ensemble L. Dans l'exemple, le premier circuit détecté est (a, b, f) ; il est obtenu par le développement des chemins issus de a.
L'étape suivante consiste à rechercher les co-circuits par le parcours de chemins issus des sommets de ce premier circuit : le circuit (a, b, c, d, e, f) est ainsi détecté. De cette manière, tous des circuits du graphe résiduel ayant des sommets atteints dans le graphe résiduel sont extractibles ; le graphe après extraction des circuits est, soit vide, soit il possède des nœuds sans prédécesseur et le parcours précédent peut être redémarré.
L'algorithme de recherche des chemins issus de sommets est réalisé en largeur d'abord et sa complexité est en $O(n^2)$ avec n le nombre de sommets du graphe résiduel.

[1] En raison de l'attribution des horloges aux signaux et de la structuration des horloges, on ne peut avoir de circuits sans entrée.

III.3 Application à l'évaluation paresseuse

L'évaluation paresseuse consiste à effectuer une action lorsque son résultat est nécessaire pour progresser dans le calcul ; elle correspond à une activation par les sorties. Le pendant de la notion d'évaluation paresseuse dans le langage SIGNAL se réalise par le calcul des fréquences d'évaluation des signaux à partir des fréquences d'émission des sorties.

Le concept d'évaluation paresseuse en SIGNAL est illustré par l'exemple en figure III.2 ; sa formalisation sous forme de dioïde est donnée dans le sous-section III.3.2.

III.3.1 Exemple

Grâce à l'évaluation paresseuse en SIGNAL, le processus présenté en figure III.2-a se réécrit en le processus donné en figure III.2-b.

```
process MANIP_VECTEUR =
    (integer N)
    { ? [N] real A, B;
        logical C0, C1
      ! [N] real Y, Z}
    (| synchro {A, B, C0, C1}
     | X := A+B
     | Y := (X when C0) default A
     | Z := (X when C1) default B
     |)/ X
    where
          [N] real X
end
```

(*a*)

```
process MANIP_VECTEUR =
    (integer N)
    { ? [N] real A, B;
        logical C0, C1
      ! [N] real Y, Z}
    (| synchro {A, B, C0, C1}
     | X := (A when (C0 or C1))+
            (B when (C0 or C1))
     | Y := (X when C0) default A
     | Z := (X when C1) default B
     |)/ X
    where
          [N] real X
end
```

(*b*)

Figure III.2 : L'évaluation paresseuse en SIGNAL

Dans le programme III.2-a, le signal **x** est produit dès que **a** et **b** sont présents : $h_N(x) = a^2 = b^2$. En réalité, l'utilisation de **x** n'est effective qu'aux instants où au moins une condition parmi $c0$ et $c1$ porte la valeur vraie. La définition de **x** en fonction de ses utilisations est définie (en SIGNAL) par le sous-échantillonnage sur le *ou* logique des conditions de filtrage de son utilisation.
Le gain résultant est appréciable : l'évaluation coûteuse de l'addition des vecteurs **a** et **b** est supprimée lorsque $c0$ et $c1$ présentent simultanément la valeur $faux$.

III.3.2 Formalisation

Formellement, l'horloge d'utilisation h_U des nœuds du graphe se définit par :

$$h_U(i) = \begin{cases} h_N(i) & \text{si } i \in F \\ \bigvee_{j \in \Gamma(i)} h_\Gamma(i,j).h_U(j) & \text{sinon} \end{cases} \quad (15)$$

Détaillons exhaustivement les signaux de F dont la fréquence de production ne peut être réduite sous peine de modification sémantique.

(1) *Les sorties.*

Clairement, leur fréquence ne peut être réduite sans modifier le comportement du programme.

(2) *Les signaux de retard.*

Ces signaux définissent l'état interne du système. En l'absence d'évaluation des trajectoires, il est impossible d'émettre des hypothèses sur les utilisations futures des valeurs de signaux retardés.

(3) *Les signaux de définition d'horloges.*

Afin de préserver le comportement du contrôle, il suffit de conserver les fréquences des signaux qui en sont les entrées. Les signaux de définition d'horloges apparaissent en partie droite de sous-échantillonnages ; ils définissent les horloges par des dépendances du graphe de dépendances conditionnées vers la hiérarchie (voir les dépendances en pointillé dans la figure B.2.3 (p. 70) qui représente le graphe dynamique de l'exemple de l'horloge à remise à zéro (sous-ensemble d'un chronomètre).

Cette algèbre de chemins se rapporte à l'algèbre booléenne $(F_3, \vee, ., 0_h, h_0)$ qui est donc un dioïde.

Convergence

Tout circuit dans le dioïde $(F_3, \vee, .)$ est 1-absorbant : soit $\mu = (i_1, \ldots, i_n, i_1)$ un circuit et $\mu^2 = (i_1, \ldots, i_n, i_1 \ldots, i_n)$.

$$\begin{aligned} w(\mu^2) &= h_\Gamma(i_n, i_1).\prod_{k=1}^{n-1} h_\Gamma(i_k, i_{k+1}).\prod_{k=1}^{n-1} h_\Gamma(i_k, i_{k+1}) \\ &= h_\Gamma(i_n, i_1).\prod_{k=1}^{n-1} h_\Gamma(i_k, i_{k+1}) \\ &= w(\mu) \end{aligned}$$

Circuit-consistance

Les graphes dynamiques dans leur vision homogène sont constitués d'un graphe de dépendance, d'un graphe d'horloges et de connexion entre ces deux graphes.
L'étude de la circuit-consistance (préservation de l'absence de circuit) se répartit selon les trois types de circuits.

- *Circuit sur signaux.*

  Le calcul de l'évaluation paresseuse enrichit l'ensemble d'horloges d'un programme et donc uniquement le graphe d'horloges : aucun circuit dans le graphe de dépendances conditionnées ne peut être introduit.

- *Circuit sur horloges.*

  L'introduction de tels circuits ne peut survenir car nous travaillons sur des termes clos et le mécanisme (détaillé dans [4]) de calcul d'horloge s'opère hiérarchiquement : les bornes inférieures et supérieures d'horloges placées dans la hiérarchie sont toujours définissables sans ajout de récursivité.

- *Entre signaux et horloges.*

  Considérons x, un nœud du graphe de dépendances conditionnées qui appartient à un tel circuit. L'existence d'un tel circuit impose la présence d'une dépendance du graphe des signaux vers le graphe des horloges et réciproquement. Donc, par transitivité, il faut que le signal x définisse une horloge h et que h participe au calcul de l'horloge d'utilisation de x. Or, si x définit une horloge alors il possède son horloge de définition comme horloge d'utilisation (cf cas numéro 3 dans le détail de F de l'équation (15)) : la création d'un tel circuit est impossible.

Conclusions

La notion d'évaluation paresseuse dans le langage SIGNAL conserve la sémantique initiale. Elle constitue une optimisation : elle permet de réduire les coûts des manipulations de données des programmes. En contrepartie, le coût du contrôle est augmenté : il y a transfert de coût des manipulations de données vers le contrôle. Ce transfert de coût est réducteur : le contrôle est défini dans le treillis booléen donc il possède un coût de mise en œuvre très faible comparé celui des manipulations des données. De plus, cette optimisation en fréquence de calcul des nœuds permet de plus amples agrégations (voir l'exemple dans la section II.4 (p. 22)).
Cette évaluation paresseuse est une réécriture SIGNAL donc l'optimisation obtenue est indépendante de l'architecture cible.

Une réduction plus importante (et plus complexe) des fréquences de calcul est obtenue grâce aux *horloges minimales de calcul* [6]. Ce concept, qui utilise le calcul d'horloge

dynamique, augmente grandement la masse de contrôle : son utilisation ne peut pas être systématique.

Chapitre 4

Vers l'implantation parallèle

La mise en œuvre parallèle de programmes SIGNAL, sous-tendue par le caractère flot de données du langage, nécessite deux étapes : la répartition relativement à l'architecture cible ; l'implantation parallèle du partitionnement retenu.
La répartition des programmes s'opère par une coopération de deux mécanismes : le passage du micro-parallélisme exprimé par le graphe dynamique vers un macro-parallélisme ; la recherche d'une application qui, pour tout nœud d'un graphe dynamique, associe un nœud du graphe de représentation de l'architecture cible. Le premier mécanisme qui ne permet pas de réduire la complexité du second en réduit la taille du problème (résultat d'importance en présence de problème touchant la NP-complétude). L'implantation parallèle efficace, à partir de la répartition retenue, s'obtient par la définition d'un schéma d'exécution optimal sur chaque processeur et par la minimisation du contrôle de coopération.

Ce chapitre précise l'ensemble des concepts introduits dans la recherche d'une solution à ces problèmes, exception faite de la définition d'une partition dont un premier pas fut obtenu par le concept de *granule* présenté dans [14].
Précisément, ce chapitre qui débute par la présentation générale du schéma d'exécution retenu, se poursuit en séquence par la définition des trois concepts suivants.

- La réduction du graphe aux E/S afin d'obtenir la vision synthétique externe des programmes SIGNAL.

- L'enrichissement du graphe qui réalise un renforcement des dépendances du graphe ; appliqué à la réduction au E/S, il permet la détection de propriété de composition et de séquentialisation de processus.

- Les lignées qui s'utilisent dans la mise en œuvre séquentielle de processus dédiés à des processeurs.

IV.1 Méthodologie de répartition

Considérons P un programme SIGNAL, ce programme codé dans $\mathbf{Z}/3\mathbf{Z}$ est réécrit en un programme équivalent constitué de deux sous-processus $C|P'$: C est la décompilation en SIGNAL des relations d'horloges ; P' est équivalent à P modulo les contraintes explicites d'horloges qui sont reportées dans C.
Définir la répartition de P sur une architecture à m processeurs revient, suite à cette réécriture, à définir un partitionnement de C et P' en m processus :

$$\begin{array}{l} C = (|C_1| \ldots |C_m|) \\ Q = (|Q_1| \ldots |Q_m|) \end{array} \equiv \quad P'$$

Grâce aux propriétés d'associativité et de commutativité de l'opérateur $|$, le processus $C|Q$ peut se réécrire en un processus équivalent D qui définit la distribution de P :

$$\begin{array}{rcl} D & = & (|D_1| \ldots |D_m|) \\ D_i & = & (|C_i|Q_i|) \\ D & \equiv & P \end{array}$$

La mise en œuvre efficace de P nécessite :

- la mise en œuvre efficace de chaque Q_i sans introduire d'interblocages.

  Dans cette approche, le schéma d'exécution efficace de chaque Q'_i s'affine grâce à l'ensemble des trois sections de ce chapitre. La section IV.2 définit le point d'articulation entre l'environnement et le processus. Sur la structure définie, le second concept (présenté en section IV.3) permet d'extraire des propriétés de séquentialisation, ce qui contribue à la mise en œuvre efficace de chaque Q_i, et de définir la notion de réutilisablilité de processus en code exécutable, ce qui apporte une aide certaine à la définition de la compilation séparée de programmes SIGNAL.

  La section IV.4 de ce chapitre définit une structure qui complète celle de la première section, dans la recherche d'une implémentation efficace de chaque Q_i ;

- la mise en œuvre des C_i et des mécanismes de synchronisation entre-eux.

  Dans la résolution de ce problème, le concept décrit dans la seconde section permet une optimisation du contrôle de flux ; ce dernier est réalisé par la mise en œuvre des C_i.

Tous de ces travaux visent à concevoir un *atelier logiciel* qui, par la localité dans la vérification de propriétés, est une aide appréciable dans la conception d'importants et complexes systèmes temps-réel.
Nous remarquerons, la similarité des problèmes posés par l'implantation parallèle avec ceux de la compilation séparée.

En effet, en conservant la même représentation,

$$D = (|D_1| \ldots |D_n|),$$

et en supposant chaque $D_i = (|C_i|Q_i|)$ comme un processus compilé séparément, la compilation séparée d'un processus D_i revient à :

- extraire l'information minimale des processus D_i pour vérifier la correction de D.

  Cette information minimale s'obtient par une restriction de D_i à son interface. La vérification de la correction de D s'opère, à l'édition de liens, sur la composition de ces restrictions.

- définir une représentation de D_i la plus proche possible du code cible.

  Cette représentation peut posséder des parties (exhibées par des propriétés du graphe renforcé) directement en code séquentiel. Le programme D s'obtient, à l'édition de lien, par composition de ces représentations.

Ainsi, le point de vue suivant prévaut, tout au moins au niveau des outils :

$$\text{implantation parallèle} \equiv \text{compilation séparée}$$

IV.2 Interface conditionnée : une base d'études

Les graphes dynamiques sont des structures, à grain fins. Le but de ce chapitre est l'étude des ordonnancements permettant de définir la notion de séquentialisation ou celle de réutilisation pour les programmes SIGNAL. L'étude de ces ordonnancements doit porter sur une structure ne contenant que les points d'ordonnancement d'un graphe dynamique.
Globalement, un ordonnancement est déterminé, soit par les entrées dans une mise en œuvre flot de donnée, soit par les sorties en "demand-driven". Ainsi la structure adéquate comporte une vision du programme au niveau de son interface. Afin de conserver le comportement interne du graphe dynamique, il convient d'ajouter les dépendances entre les nœuds de l'interface. Cette structure qui mêle interface et dépendances est appelée *interface conditionnée.*

IV.2.1 Fermeture de graphes

L'obtention des dépendances entre les nœuds de l'interface nécessite, dans les graphes dynamiques, le calcul des horloges $h^*_\Gamma(ij)$ de dépendance entre tout couple (i, j) de sommets de l'interface.
Intuitivement, l'horloge $h^*_\Gamma(ij)$ est définie par vrai si au moins une horloge de chemin entre i et j vaut vrai : $h^*_\Gamma(ij)$ est définie sur le "ou" des horloges de chemins entre i et j. L'horloge d'un chemin de i_1 à i_n vaut vrai si tous les arcs de $i_1, \ldots, i_n$ portent une horloge à vrai : l'horloge d'un chemin de i_1 à i_n se définit sur la borne inférieure des horloges du chemins.

Ainsi, l'horloge $h^*_\Gamma(ij)$ de dépendance entre deux sommets i et j s'évalue selon l'équation (16).

$$h^*_\Gamma(i,j) = \bigvee_{p=(i,k_1,\ldots,k_n,j)} h_\Gamma(i,k_1).\cdots.h_\Gamma(k_n,j) \tag{16}$$

L'équation (16) se réécrit, sur une forme récursive, en l'équation (17) : $h^*_\Gamma(i,j)$ se définit comme la borne supérieure des produits de $h^*_\Gamma(i,k)$ par l'horloge de l'arc de k à j, pour tout k prédécesseur de j

$$h^*_\Gamma(i,j) = \bigvee_{k\in\Gamma^{-1}(j)} h^*_\Gamma(i,k).h_\Gamma(k,j) \tag{17}$$

L'horloge $h_\Gamma(i,i)$ est définie par la borne supérieure du treillis.

Formellement, l'horloge $h_\Gamma(i,j)$ de dépendance de i à j se définit, sous la forme de dioïde, par :

$$\forall i,j \in N, \quad h^*_\Gamma(i,j) = \begin{cases} h_0 & \text{si } i = j \\ \bigvee_{k\in\Gamma^{-1}(j)} h^*_\Gamma(i,k).h_\Gamma(k,j) & \text{sinon} \end{cases}$$

avec $\Gamma^{-1}(i) = \{j \in N | (j,i) \in \Gamma\}$. Ce système d'équation se réécrit simplement en :

$$\forall i,j \in N, \quad h^*_\Gamma(i,j) = \begin{cases} h_0 & \text{si } i = j \\ \bigvee_{k\in N} h^*_\Gamma(i,k).h^+_\Gamma(k,j) & \text{sinon} \end{cases}$$

avec h^+_Γ une extension de h_Γ définie par :

$$\forall (i,j) \in N^2, \quad h^+_\Gamma(i,j) = \begin{cases} h_\Gamma(i,j) & \text{si } (i,j) \in \Gamma \\ 0_h & \text{sinon} \end{cases}$$

Cette algèbre de chemin est définie sur le treillis $(C, \vee, .)$ qui est une algèbre de Boole $< C, \vee, ., 0_h, h_0 >$ de borne supérieure h_0 (la racine de l'arborescence) et de borne inférieure 0_h (l'horloge nulle). $< C.\vee, ., 0_h, h_0 >$ étant une algèbre de Boole, c'est à fortiori un dioïde.

Ce dioïde se compose des monoïdes commutatifs $(C, \vee, 0_h)$ et $(C, \bullet, h_0)$; il s'évalue par le calcul des puissance $A'^{(k)}$ avec $A' = E \vee A$:

$$\forall i,j \in N, \quad A_{ij} = h^+_\Gamma(i,j) \quad \text{et} \quad E_{ij} = \begin{cases} h_0 & \text{si } i = j \\ 0_h & \text{sinon} \end{cases}$$

Convergence

La convergence du dioïde $< C, \vee, . >$ revient à vérifier la p-absorption et donc, déterminer p. Considérons $\mu = (i_1, \ldots, i_n, i_1)$ un circuit élémentaire quelconque ; . étant commutatif, tous les circuits pointés d'un circuit possède le même poids. Le poids de ce circuit noté $w(\mu)$ est défini par :

$$w(\mu) = h_\Gamma(i_n, i_1). \prod_{k=1}^{n-1} h_\Gamma(i_k, i_{k+1})$$

Si μ^n dénote le chemin qui contient n fois le chemin élémentaire μ alors, par définition du poids des chemins $w(\mu^n) = w(\mu^{n-1}).w(\mu)$. Ainsi, vérifier la présence de circuits p-absorbants se réduit à déterminer (s'il existe) n tel que

$$w(\mu^n) = w(\mu^{n+1})$$

Comme $< C, \vee, . >$ est un treillis, $w(\mu^2) = w(\mu).w(\mu) = w(\mu)$ et donc, le dioïde $< C, \vee, ., 0_h, h_0 >$ est 1-absorbant.

Si le poids d'un circuit est supposé égal à 0_h (absence d'interblocage) alors tout circuit est *0-absorbant* et le poids d'un nœud du graphe est défini comme la somme des poids des chemins aboutissant à ce nœud, sans tenir compte du circuit.

Ce dioïde permet d'évaluer la fermeture transitive d'un graphe dynamique.

Définition 4.1 (Fermeture transitive) *Soit* $GD =< G, X, \Sigma, < H, \Theta >, h_N, h_\Gamma >$ *un graphe dynamique avec* $G =< N, \Gamma, I, O >$. *La fermeture transitive de* GD *est un graphe dynamique* $GD^* =< G^*, X, \Sigma, < H', \Theta' > h_N, h_\Gamma^* >$ *tel que :*

- $G^* =< N, \Gamma^*, I, O >$ *ou* $< N, \Gamma^* >$ *est la fermeture transitive habituelle sur les graphes.*
- $< H', \Theta' >$ *est l'arbre d'horloge tel que* H' *est l'union des images de* h_N *et* h_Γ^*.
- h_Γ^* *est la fermeture transitive de* h_Γ *définie sur* $< C, \vee, . >$ *par la formule (16).*

IV.2.2 Projection de graphe

La structure d'interface conditionnée fait abstraction du micro-parallélisme présent dans les graphes dynamiques ; elle ne conserve que le comportement interne faisant intervenir des signaux de l'interface.

Cette réduction des graphes dynamiques aux signaux de l'interface suit la définition 4.2.

Définition 4.2 (Projection) *Soit* $GD^* =< G^*, X, \Sigma, < H, \Theta >, h_N, h^*_\Gamma >$ *une fermeture transitive de graphe dynamique dont* $G^* =< N, \Gamma^*, I, O >$. *Soit un ensemble* A *tel que* $A \subset N$. *La projection de* GD^* *sur* A, *dénotée* $GD^*|_A$, *est un graphe dynamique* $< G^*|_A, X, \Sigma, < H, \Theta >, h|_A, h^*|_{\Gamma_A} >$ *tel que :*

- $G|_A =< A, \Gamma^*|_A, I|_A, O|_A >$ *ou* $\Gamma^*|_A = \Gamma^* \cap (A \times A)$, $I|_A = I \cap A$ *et* $O|_A = O \cap A$.
- $h|_A$ *est la restriction de* h_N *à* A.
- $h^*|_{\Gamma_A}$ *est la restriction de* h^*_Γ *à* $\Gamma^*|_A$

La projection d'un graphe dynamique sur ses signaux d'interface désignés dans I et O définit la structure d'interface conditionnée :

Définition 4.3 (Interface conditionnée) *Soit* $GD =< G, X, \Sigma, < H, \Theta >, h_N, h_\Gamma >$ *un graphe dynamique de graphe* $G =< N, \Gamma, I, O >$. *L'interface conditionnée associée à* GD *se définit par* $GD|_{I \cup O}$

Afin d'obtenir une structure d'interface conditionnée minimale, il faut définir la projection de l'arbre des horloges $< H, \Theta > |_A$. Cette définition est omise, une description plus fine de la hiérarchie étant nécessaire.

Une interface conditionnée générique est fournie dans la figure IV.3. Les notations présentes dans cette figure sont utilisées dans les formules d'enrichissement de graphe ; ce dernier constitue le sujet de la prochaine section.

IV.3 Enrichissement : un critère d'ordonnancement

La recherche d'une mise en œuvre efficace d'un programme P revient à l'implémentation de sa répartition D ; l'efficacité de cette implémentation passe évidemment par la conception d'un schéma maximalement séquentiel de chaque processus D_i dédié au processeur i. Cette section va expliciter un critère d'ordonnancement qui, tout en conservant l'assurance de ne pas introduire d'interblocages, définit un schéma maximalement séquentiel.

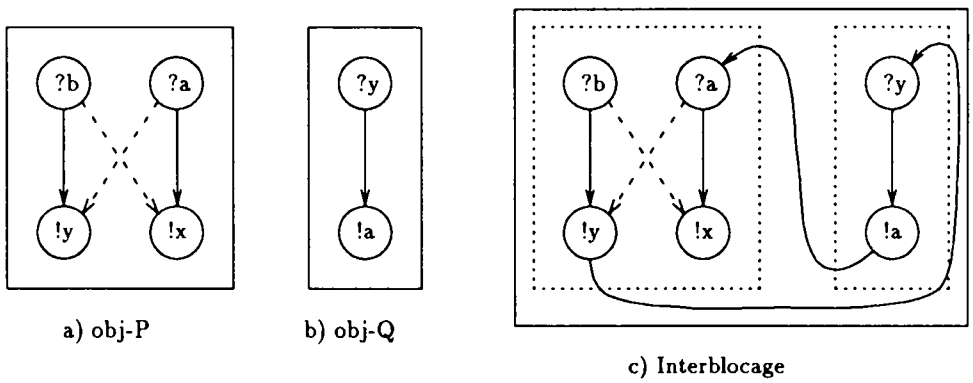
Dans cette section, nous introduisons le concept d'enrichissement du graphe à l'aide d'exemples de complexité croissante ; la dernière partie de cette section est vouée à la définition formelle de ce concept.

IV.3.1 Exemples et lignes directrices

Considérons le programme suivant dans lequel `f` et `g` sont des fonctions quelconques :

```
P = (| y := g(b) | x := f(a) |)
```

La répétition infinie de la séquence : prendre en compte les entrées ; exécuter les calculs et délivrer les résultats, constitue un schéma d'exécution possible. Cette mise en œuvre de `P` dénotée `obj-P` est illustrée par la figure IV.1-a dans laquelle les dépendances en pointillés expriment les précédences définies par le schéma d'exécution choisi : on lit toutes les entrées avant d'effectuer les sorties. Dans cette figure, le programme `P` étant monochrone, les nœuds et des arcs sont étiquetés par la même horloge ; l'étiquette `a` été omise.



a) obj-P b) obj-Q c) Interblocage

Figure IV.1 : Mise en œuvre dépendante du contexte

Malheureusement, le contexte d'utilisation de `P` peut avoir un comportement tel celui imposé par le programme SIGNAL `Q` suivant dans lequel `h` est une fonction quelconque.

```
Q = (| a := h(y) |)
```

Le seul schéma d'exécution possible de `Q` se compose, en séquence, de la répétition infinie : prendre en compte `y` et délivrer `a`, ce qui est illustré par la figure IV.1-b.

Le programme SIGNAL source `P|Q` est correct. Néanmoins, l'exécution concurrente[1] de la mise en œuvre de `P` et de celle de `Q` produit un interblocage (voir figure IV.1-c). En effet, `obj-P` attend `a` et pour produire `y` or `obj-Q` attend `y` pour délivrer `a`. Ainsi, le programme `P` est *non réutilisable* sous sa forme objet `obj-P`.

[1] au sens des systèmes multitâches

Afin d'étudier plus finement l'enrichissement de graphes, nous définissons un autre programme monochrone R qui diffère de P en sa définition de x.

```
R = (| y := g(b) | x := f'(a,b) |)
```

Le graphe de dépendance du nouveau processus R autorise l'ordonnancement total de ses communications grâce à la dépendance en pointillé qui réalise un *renforcement* de son graphe ; voir figure IV.2.

Cette mise en œuvre de R peut introduire un interblocage si : a précède y dans un contexte C. Comme a → x et b → y dans le graphe de R, il faut que dans ce contexte x précède b pour que a précède y.
Dans ce cas, un circuit entre x et b préexiste dans le graphe $G_R|G_C$ obtenu par composition de G_R, le graphe de R, et G_C, le graphe de C : le renforcement du graphe de R, qui définit sa mise en œuvre, n'introduit pas de circuit ; une telle mise en œuvre sera dite *circuit-consistante.*

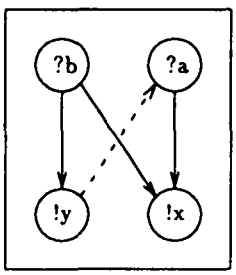


Figure IV.2 : La mise en œuvre obj-R de R

Dans ce dernier exemple, R est *réutilisable* sous sa forme séquentielle quel que soit son contexte d'utilisation

IV.3.2 Définition

La présentation d'exemples a introduit les termes *renforcement* et *circuit-consistance* auxquels nous allons donner une définition précise.

Renforcement.
Ce terme, illustré par les dépendances en pointillé dans les figure IV.1 et IV.2, définit les dépendances induites par la mise en œuvre choisie. En terme de graphe, un renforcement indique l'ajout d'arcs qui modifient la fermeture transitive des graphes et ainsi renforce le préordre induit. Le terme de renforcement étendus aux graphes dynamiques amène à la définition 4.4.

Définition 4.4 (Renforcement) *Soient deux graphes dynamiques*
$GD_1 =< G_1, X, \Sigma, < H_1, \Theta_1 >, h_N, h_{\Gamma_1} >$ *avec* $G_1 =< N, \Gamma_1, I, O >$
$GD_2 =< G_2, X, \Sigma, < H_2, \Theta_2 >, h_N, h_{\Gamma_2} >$ $G_2 =< N, \Gamma_2, I, O >$
GD_2 est un renforcement GD_1 si et seulement si :

- *G_1 est un renforcement de G_2 c'est à dire que* $\Gamma_1^* \subset \Gamma_2^*$
- $\forall x, y \in N, \quad h^*_{\Gamma_2}(x, y) \geq h^*_{\Gamma_1}(x, y)$

Circuit-consistance.
Intuitivement, une mise en œuvre est circuit-consistante si elle préserve l'absence de circuit quel que soit le contexte d'utilisation (sans circuit) du programme. Dans la définition 4.5, l'opérateur . dénote le pendant l'opérateur de composition de graphes dynamiques associés ; un contexte est un graphe qui définit les entrées et consomme les sorties d'un graphe.

Définition 4.5 (Circuit-consistance) *Soient deux graphes dynamiques*
$GD_1 =< G_1, X, \Sigma, < H_1, \Theta_1 >, h_N, h_{\Gamma_1} >$ *avec* $G_1 =< N, \Gamma_1, I, O >$
$GD_2 =< G_2, X, \Sigma, < H_2, \Theta_2 >, h_N, h_{\Gamma_2} >$ $G_2 =< N, \Gamma_2, I, O >$
GD_2 est circuit-consistant pour GD_1 si et seulement si, quel que soit le contexte C d'utilisation de GD_1, $GD_1.C$ sans circuit $\Rightarrow$ $GD_2.C$ sans circuit.

Afin de donner un sens à cette définition, il convient de définir strictement la notion de circuit.

Définition 4.6 (Circuit) *Soit $GD =< G, X, \Sigma, h_N, h_\Gamma >$ un graphe dynamique. GD est sans circuit (interblocage) si et seulement si :*

$$\forall x, y \in N \quad h^*_\Gamma(x, y).h^*_\Gamma(y, x) = 0_h$$

Un faux-circuit, dont la notion fut évoquée en début de chapitre 2 (p. 11), correspond à un circuit de G alors GD est sans circuit.

Nous allons maintenant définir le concept clef de cette section : l'enrichissement.

Définition 4.7 (Enrichissement) *Soient deux graphes dynamiques GD et GD'. GD est un enrichissement de GD si et seulement si GD' est un renforcement circuit-consistant de GD*

L'enrichissement se fonde sur les deux précédentes définitions : c'est un renforcement donc l'enrichissement définit une mise en œuvre des programmes ; il est circuit-consistant donc les programmes mis en œuvre de cette manière n'introduisent pas d'interblocage (ils sont réutilisables).

Avant d'aller plus avant dans la recherche d'une formule permettant de définir l'enrichissement, donnons dans la figure IV.3 l'interface conditionnée générique qui contient les notations utilisées dans l'énoncé de cette formule.

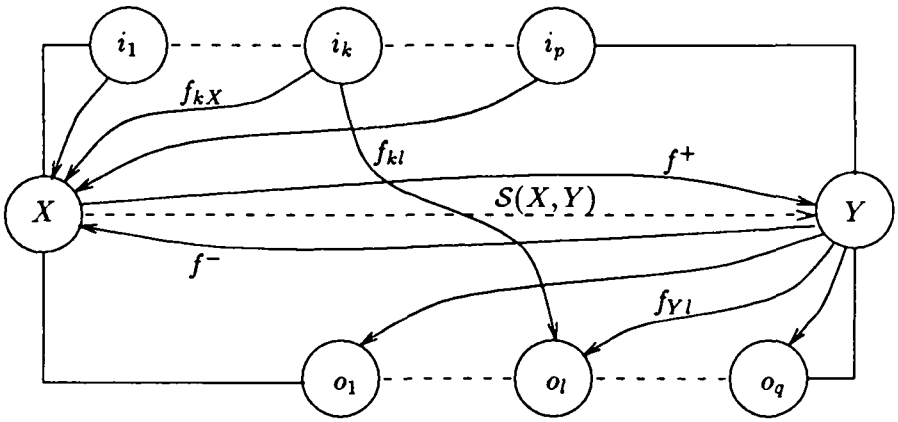


Figure IV.3 : Une interface conditionnée générique

Dans cette figure, les nœuds i_k dénotent des nœuds d'entrée et o_l dénote ceux de sortie. Les X et Y sont indifféremment des nœuds d'entrée ou de sortie. L'enrichissement de l'arc entre X et Y est défini par l'horloge $S(X,Y)$; cette dernière définit l'horloge de mise en Séquence de X avant Y.

Intuitivement, $\mathcal{S}(X,Y)$ n'introduit pas de circuit si :

- $\mathcal{S}(X,Y)$ n'introduit pas de circuit interne : elle vérifie l'équation (18)

$$\mathcal{S}(X,Y).f^- = 0_h \tag{18}$$

- $\mathcal{S}(X,Y)$ n'introduit pas de circuit avec le contexte. Vis à vis de l'extérieur, $\mathcal{S}(X,Y)$ participe à la dépendance en i_k et o_l d'horloge $f_{kX}.\mathcal{S}(X,Y).f_{Yl}$. Soit h l'horloge de la dépendance entre o_l et i_k introduite par le contexte.

  Ce contexte ne crée pas d'interblocage avec le processus originel si h vérifie l'équation (19).

$$f_{kl}.h = 0_h \tag{19}$$

  Au maximum, h peut-être égale à la complémentaire de f_{kl} ; cette complémentaire est représentée par $1 - f_{kl}$. En remplaçant h par sa complémentaire, nous obtenons l'équation (20) qui définit la condition pour ne pas introduire de circuit avec le contexte.

$$\mathcal{S}(X,Y).f_{Yl}.(1 - f_{kl}).f_{kX} = 0_h \tag{20}$$

La conjonction des équations (18) et (20) induit le théorème suivant.

Théorème 2 (Borne d'enrichissement d'arc) *Soient X, Y deux nœuds d'interface et f^+ la dépendance de X vers Y. $\mathcal{S}(X,Y)$ est un enrichissement de f^+ si et seulement si les deux inégalités suivantes sont vérifiées.*

$$\mathcal{S}(\mathtt{X},\mathtt{Y}) \geq f^+ \tag{21}$$

$$\mathcal{S}(\mathtt{X},\mathtt{Y}) \leq f^+ + X^2.Y^2.(1 - f^- - f^+).\prod_{k,l}(1 - f_{kX}.f_{Yl}.(1 - f_{kl})) \tag{22}$$

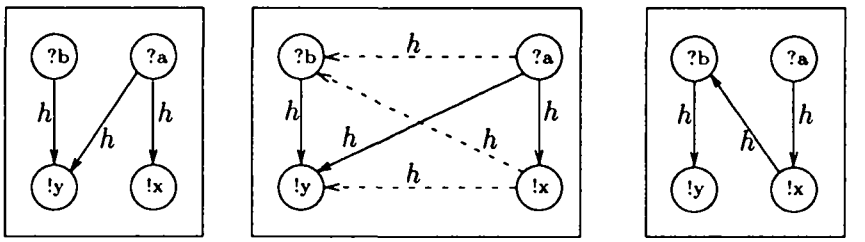
Une preuve formelle de ce théorème sera fournie dans un prochain document. Intuitivement, l'équation (21) provient du renforcement dans la définition d'enrichissement. La seconde équation se déduit, par une algèbre élémentaire, des équations (18) et (20) dont les preuves nécessitent des définitions et propositions annexes. Concentrons nous sur les utilisations de ce théorème.

IV.3.3 Utilisations

Considérons le processus R précédemment défini par :

```
R = (| y := g(b) | x := f'(a,b) |)
```

Le graphe de dépendance de R est rappelé en figure IV.4-a. En appliquant la substitution, de chaque dépendance par son enrichissement maximal défini par l'inégalité (22), nous obtenons le graphe enrichit présenté en figure IV.4-b.



a) Graphe de R b) Enrichissement maximal c) Schéma d'exécution

Figure IV.4 : Vers un schéma d'exécution

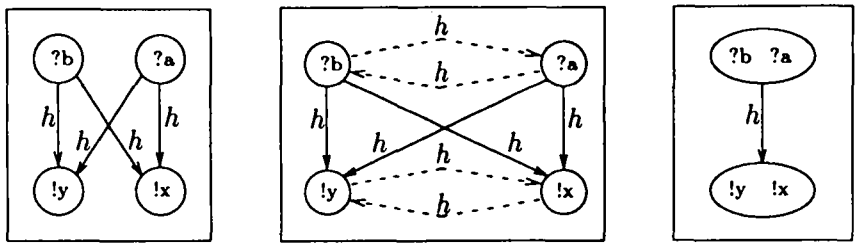
Le graphe enrichit définit un ordre ; il permet de déduire le schéma d'exécution de R présenté en figure IV.4-c.

Bien que l'enrichissement maximal d'un arc soit circuit-consistant pour le graphe d'origine, il n'en est pas de même sur un enrichissement de ce graphe. Pour exemple, prenons le processus S définit ci-dessous.

```
S = (| y := g'(a,b) | x := f'(a,b) |)
```

Le graphe de dépendance de S est dessiné en figure IV.5-a. Comme pour le processus R, calculons l'enrichissement maximal de S obtenu par substitution des arcs par la borne supérieure de leur enrichissement ; cet enrichissement maximal de S est présenté en figure IV.5-b.
Comme l'illustre la figure IV.5-b, l'enrichissement maximal peut contenir des circuits.

Que signifient ces circuits dans l'enrichissement maximal ? L'enrichissement indiquant la possibilité d'ordonnancement circuit-consistant, un circuit d'enrichissement indique une totale indépendance d'ordonnancement. Cette totale indépendance permet de regrouper ces nœuds sans introduire d'interblocage : une communication simultanée des signaux associés aux nœuds peut être entreprise. Sur le processus S, l'enrichissement maximal



a) Graphe de S b) Enrichissement maximal c) Schéma de communication

Figure IV.5 : Vers un schéma de communication

indique un regroupement possible des nœuds d'entrée et de sortie ; la figure IV.5-c illustre le schéma de communication associé à S.
Un schéma de communication comme celui présenté dans la figure IV.5-c indique que le processus associé peut être mis en œuvre comme une fonction : une entité intéressante pour une mise en œuvre pipelinée.

Remarque
En OC [7], les transitions sont atomiques vis à vis de l'environnement : les entrées sont consommées en début de transition; les sorties sont émises en fin de transition. Ce schéma correspond à définir des dépendances maximales entre les entrées i_k et les sorties o_l attachées à une transition :

$$\forall i_k, o_l \quad h_\Gamma(i_k, o_l) = h_N(i_k).h_N(o_l)$$

L'enrichissement, qui s'appuie sur la structure d'interface conditionnée, permet le calcul d'ordonnancements sur les nœuds de l'interface. "Quels sont les effets sur la mise en œuvre de processus sur un processeur?" telle est la question sous-jacente ; la section suivante vise, au travers d'un nouveau concept, à apporter une réponse.

IV.4 Lignées

Les différents ordonnancements étant liés aux signaux de l'interface, la structure d'interface conditionnée a été dégagée. Sur cette structure, des renforcements ont été entrepris et ont permis, sans introduire d'interblocage, de réduire la liberté des nœuds de l'interface. Il convient maintenant de définir les points d'articulations des différents ordonnancements.

Intuitivement, dans une exécution gouvernée par les entrées (flux de données), l'exécution d'un nœud du graphe nécessite la connaissance de ses prédécesseurs. Par transitivité, seul l'ordre d'arrivée des entrées détermine son moment physique d'exécution des nœuds.
Ces constations conjuguées à une réflexion de même nature sur les sorties, nous ont conduit à déterminer une structure interne du graphe obtenue par partitionnement selon les dépendances d'interface (seules les dépendances sur entrées sont considérées dans cette section ; l'étude symétrique à partir des sorties suit la même progression).

La section IV.4.1 présente un exemple de partitionnement en lignées; la section suivante en donne une définition formelle.

IV.4.1 Exemple

La figure IV.6 représente le graphe des lignées obtenu par partitionnement du graphe d'un programme d'entrées a et b. Les nœuds du graphe sont regroupés selon l'état de connaissance sur les entrées nécessaire pour leur exécution.

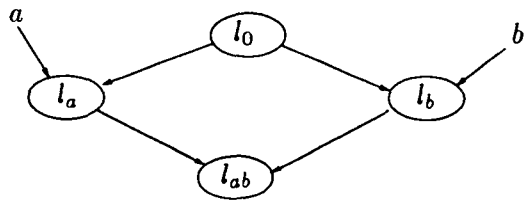


Figure IV.6 : Un graphe de Lignées

Dans cette figure, le sous-graphe des nœuds dépendants des entrées $i_1, \ldots, i_n$ est noté $l_{i_1 \ldots i_n}$. La lignée l_0 contient les nœuds sans entrée comme prédécesseur que sont, par exemple, les retards appartenant au sous-graphe de la racine.

A l'exécution, l'activation des sous-processus définis par les lignées sont activées selon l'ordre de présence des entrées. Dans l'exemple, selon l'ordre d'arrivée de a et b, l'exécution séquentielle contrainte par le graphe des lignées sera l_0 ; l_a ; l_b ; l_{ab} ou l_0 ; l_b ; l_a ; l_{ab}
A la mise en œuvre, les processus correspondant aux lignées sont traduits en code séquentiel comme tout processus SIGNAL. Ces processus sont activés par un automate déduit du graphe des lignées ; cet automate contient des instructions asynchrones de lecture d'entrées comme celles présentées dans la section II.5.1 (p. 26).
L'enrichissement du graphe détermine des ordonnancements circuit-consistant des entrées donc définit (par transitivité) des ordonnancements de lignées. Dans l'exemple, si le renforcement du graphe permet de séquentialiser a et b, toutes les lignées peuvent être séquentialisées et l'automate d'activation disparaît.

IV.4.2 Définition

Par transitivité, l'ensemble des dépendances d'entrées sur le sommet j, noté $l(j)$, se définit sur l'union des ensembles $l(i)$ de ses prédécesseurs et de l'ensemble des prédécesseurs qui sont des entrées.
Formellement, $l(i)$ l'ensemble des dépendances d'entrées sur le sommet i est défini par :

$$l(j) = \bigcup_{i \in \Gamma^{-1}(j)} l(i) \cup \{i | i \in I\}$$

En réalité, les nœuds d'entrée ont un rôle particulier ; il est préférable de les considérer comme point d'entrée de leur lignée.
Ainsi, on obtient la formule de définition (23).

$$\forall j \in N \quad l(j) = \begin{cases} \bigcup_{i \in \Gamma^{-1}(j)} l(i) \cup \{j\} & \text{si } j \in I \\ \bigcup_{i \in \Gamma^{-1}(j)} l(i) & \text{sinon} \end{cases} \tag{23}$$

Soient $\mathcal{P}(I)$ les partitions de I, cette algèbre des lignées s'inscrit dans le dioïde $(\mathcal{P}(I), \cup, \cap)$, Cette algèbre s'évalue selon l'équation (24); les matrices A et E sont définies par (25).

$$\forall j \in N \quad l(j) = \bigcup_i [l(i) \cap a_{ij}] \cup e_j \tag{24}$$

$$a_{ij} = \begin{cases} I & \text{si } (j,i) \in \Gamma \\ \emptyset & \text{sinon} \end{cases} \quad \text{et} \quad e_j = \begin{cases} \{j\} & \text{si } j \in I \\ \emptyset & \text{sinon} \end{cases} \tag{25}$$

Comme la relation d'appartenance à une lignées est une relation d'équivalence, les sous-graphes qui définissent les lignées sont obtenus par partitionnement du graphe initial selon cette relation.

$$\begin{aligned} l_{i_1,\ldots,i_n} &= \{j \mid l(j) = \{i_1, \ldots, i_n\}\} \\ l_0 &= \{j \mid l(j) = \emptyset\} \end{aligned}$$

Quand à la convergence d'un tel dioïde, vérifions que tout circuit est p-absorbant.
Le poids d'un nœud dans la valuation en lignées se définit sur l'union des poids de ses prédécesseurs.

$$\begin{gathered} l(j) = \bigcup_{i \in \Gamma^{-1}(j)} l(i) \cup \{i | i \in I\} \\ \Downarrow \\ \forall\, i \in \Gamma^{-1}(j) \quad l(i) \subseteq l(j) \subseteq I \end{gathered} \tag{26}$$

Soit $\mu = (i_1, \ldots, i_n)$ un circuit ; d'après la relation d'inclusion (26), nous déduisons la relation (27).

$$\begin{gathered} l(i_1) \subseteq l(i_2) \subseteq \cdots \subseteq l(i_n) \subseteq l(i_1) \\ l(i_1) = l(i_2) = \cdots = l(i_n) \end{gathered} \tag{27}$$

La propriété (27) indique que tous les nœuds d'un circuit appartiennent à la même lignée qui est définie sur l'union des poids des prédécesseurs du circuit. Le dioïde définit par est 1-absorbant.

Les sous-graphes représentant les lignées définissent des processus procéduraux : l'activation des processus n'est réalisée que lorsque les données (des entrées) nécessaires à son exécution (totale) sont déterminées.
Chaque lignée se traduit en code séquentiel comme définit dans le chapitre 2. Comme le critère d'optimalité de la mise en œuvre du contrôle (définit en sous-section II.2.1 (p. 14)) n'est pas conservé par concaténation, les ordonnancements obtenus par enrichissement du graphe induisent la fusion des lignées, préalablement à la génération de code.

Le procédé d'activation des processus est réalisé par un automate déduit du graphe des lignées qui a pu être réduit par l'enrichissement du graphe. Cet automate, qui possède une période d'un instant, contient l'état de détermination des entrées dans l'instant. A chaque réception, la lignée associée est activée ainsi que ses successeurs libérés ; l'activation de la lignée sur toutes les entrées termine l'instant logique.
Deux traductions en OCCAM du graphe des lignées sont données dans [16] ; l'une se construit sur les commandes gardées ; l'autre s'appuie sur des processus parallèles en contexte de mémoire partagée.

Chapitre 5

Conclusion

Dans ce document, nous avons présenté un ensemble de mécanismes introduits dans l'objectif d'une mise en œuvre parallèle des programmes SIGNAL. La présentation de ces mécanismes se découpe conformément à la progression dans l'étude de l'implémentation parallèle.

- *La production de code séquentiel.*

  Une implémentation parallèle suppose une exécution maximalement séquentielle sur chaque processeur. L'efficacité du code obtenu réside dans sa structuration hiérarchique du contrôle et dans les optimisations des aspects fonctionnels de SIGNAL qui sont mis en œuvre. L'extension de cette mise en œuvre aux processus asynchrones est motivée par l'implantation des processus dédiés à un processeur.

  Cette production de code séquentiel qui est actuellement opérationnelle réalise une optimisation des temps d'exécution obtenus avec une première version du compilateur, de 10 à 70 % selon les programmes.

- *La définition d'un schéma d'exécution parallèle.*

  Ce schéma d'exécution est esquissé grâce aux concepts d'*enrichissement du graphe* et de *lignées*, tous deux présentés dans ce document. Le premier concept permet d'une part d'augmenter la séquentialité potentielle sur chaque processeur, d'autre part de minimiser le mécanisme de coopération entre les processeurs. Le concept de lignées complète la définition du schéma d'exécution par processeur par une partition en sous-processus totalement séquentialisables.

A partir de cet ensemble d'études, les perspectives de développements sont multiples. Les principaux axes sont précisés ci-dessous.

- L'étude plus fine de l'implantation parallèle des programmes SIGNAL et de leur compilation séparée. Les principaux sujets de recherche s'orientent, soit vers l'optimisation du mécanisme de coopération entre les processeurs (mise en œuvre répartie

du contrôle), soit vers des mécanismes d'implémentation plus spécifiques à certaines architectures ou finalement, vers des critères de répartition.

L'ensemble de ces études pourront tirer avantage des solutions apportées par SYNDEX [9] qui permet une répartition quantitative de programmes synchrones.

- La définition d'un code commun fondé sur les graphes. Ce code commun avec SYNDEX permettrait de profiter de son mécanisme de distribution quasi-automatique et d'implémentation parallèle de programmes synchrones. Ce code commun devrait posséder un passage vers le code commun OC [7] ; ce dernier, fondé sur les automates, est utilisé par ESTEREL-LUSTRE.

- La définition d'une nouvelle syntaxe de SIGNAL en vue d'un développement industriel. Cette syntaxe autorisera la construction modulaire ainsi que l'utilisation aisée de bibliothèques afin de permettre une orientation du langage vers le traitement de la parole, la description d'architectures, etc.

Conjointement au développement du langage, aux recherches entreprises sur sa mise en œuvre parallèle, sa compilation séparée, la preuve de programmes ou de propriétés, le langage SIGNAL est expérimenté sur des applications de traitement du signal ou de contrôle de processus temps-réel. Parmi ces applications, les plus significatives concernent la réalisation d'un poste de traitement de la parole, d'un système radar de suivi de cibles et d'une simulation de passages à niveau.

Annexe A

SIGNAL : l'encodage

A.1 Encodage sur Z/3Z

Avec l'ensemble fini $\mathbf{Z}/3\mathbf{Z}$ des entiers modulo 3, les programmes SIGNAL sont codés, à partir des processus élémentaires SIGNAL, selon les tables A.1 et A.2. Dans l'ensemble de cette section, les signaux `a,b` et `c` sont supposés de type booléens ; les signaux `u, v, x` et `y` sont de types quelconques.

Processus SIGNAL	Codage
y := f (x1,...,xn)	$y^2 = x1^2 = \ldots = xn^2$
y := x $ 1	$y^2 = x^2$
y := x when b	$y = x^2(-b - b^2)$
y := u default v	$y = u^2 + (1 - u^2)v^2$

Tableau A.1 : Équations d'horloges

L'opérateur de composition de SIGNAL réalise l'union termes à termes des systèmes de contraintes $< X, \Sigma >$ des processus invoqués.
Précisons quelques particularités, propriétés et utilisations de ce codage dans $\mathbf{Z}/3\mathbf{Z}$.

- La puissance de $\mathbf{Z}/3\mathbf{Z}$ permet de coder l'ensemble des expressions SIGNAL sur signaux booléens. Ce codage permet des vérifications et des réécritures plus profondes en conservant l'ordre de complexité des algorithmes appliqués sur les booléens.
- Sur $\mathbf{Z}/3\mathbf{Z}$, les termes $(-b - b^2)$, $(b - b^2)$ et b^2 prennent leurs valeurs dans $\{0, 1\}$. Ces termes définissent les trois types de monômes des fonctions $\mathcal{F}_3[X] \rightarrow \{0, 1\}$; ils correspondent respectivement aux fonctions b égal à *true*, *false* et b est *absent*.
- Le codage du retard sur booléen introduit une variable ξ. Cette variable, appelée variable d'état, représente l'état (booléen) courant. Ce type de codage permet l'éva-

luation de propriétés dynamiques telles que *l'observabilité* ou *l'atteignabilité*. L'étude de ces propriétés a fait l'objet d'un premier article [13] qui décrit les possibilités et limites du modèle.

Processus SIGNAL	Codage
c := not a	$c = -a$
c := a and b	$c = a.b.(a.b - (1 + a + b))$
c := a or b	$c = a.b.(1 - (a + b + a.b))$
c := a $ 1	$\xi' = a + (1 - a^2)\xi$ $c = a^2\xi$
c := a when b	$c = a.b^2(-b - 1)$
c := a default b	$c = a + (1 - a^2).b$

Tableau A.2 : Équations sur valeurs

A.2 Encodage en dépendances conditionnées

A.2.1 Sur signaux

Le codage dans **Z/3Z** permet de transcrire l'ensemble des propriétés d'horloges et les relations induites par les expressions booléennes. Ainsi, les expressions booléennes n'introduisent pas de dépendances dans le graphe. Pour cela, les signaux x et y de la table A.3 sont supposés non booléens.

Processus SIGNAL	Dépendances conditionnées
y := f (x1,...,xn)	y^2 : x1 → y ⋮ y^2 : xn → y
y := x when b	y^2 : x → y
y := u default v	u^2 : u → y $v^2(1 - u^2)$: v → y

Tableau A.3 : Dépendances conditionnées

La table A.3 résume, pour chaque processus élémentaire SIGNAL, l'ensemble des dépendances et des étiquetages d'horloges qu'il induit.

Nous notons par $h : \mathtt{x} \rightarrow \mathtt{y}$, l'existence d'un arc entre **x** et **y** conditionné par h :

$$h : \mathtt{x} \rightarrow \mathtt{y} \quad \equiv \quad \begin{array}{l} (x, y) \in \Gamma \\ h_\Gamma(x, y) = h \end{array}$$

Les processus élémentaires dynamiques **y := u $1** ne définissent pas de dépendances instantanées ; la structure interne qui régit ces processus est arborescente et définie en section II.3 (p. 16).

A.2.2 Sur horloges

Cet encodage est effectué après résolution des contraintes Σ. L'ensemble des horloges est structuré sous forme arborescence. Cet ensemble d'horloges est encodé en graphe selon la table A.4.

Définition d'horloge	Dépendances conditionnées
$h_N(\mathtt{x}) = h$	$\mathtt{h} : \mathtt{h} \rightarrow \mathtt{x}$
`h := f(..., [c], ...)`	$c^2 : \mathtt{c} \rightarrow \mathtt{h}$
`h := f(..., h', ...)`	$k : \mathtt{h'} \rightarrow \mathtt{h}$

Tableau A.4 : Dépendances sur horloges

Dans la table A.4, l'horloge k dénote l'ancêtre commun à **h** et **h'** de profondeur maximale. Par cet encodage, le graphe aux dépendances conditionnées contient non seulement les nœuds correspondant à des signaux mais aussi les nœuds associés aux horloges.

Annexe B

Exemple de compilation

Cette annexe contient l'ensemble des structures, internes ou externes, produites par le compilateur actuel. L'exemple choisi est celui d'une horloge avec remise à zéro; cet exemple constitue un sous-ensemble d'un chronomètre.
Le programme SIGNAL originel, présenté sous forme graphique, est donné en section B.1. La section B.2 définit successivement tous les éléments de sa représentation interne : la modélisation du contrôle, la hiérarchie d'horloge, le graphe de dépendances conditionnées et finalement, le graphe dynamique dans sa vision hiérarchique.
La section B.3 donne la décompilation en SIGNAL du graphe dynamique du programme.
La dernière section de cette annexe présente les fichiers résultants de la production de code FORTRAN.

B.1 Programme source

Le programme se décompose fonctionnellement en une mécanique interne définissant les rouages de l'horloge et un interfaçage avec l'environnement.

B.1.1 La mécanique

Ce programme est construit autour des rouages (GEAR) d'un chronomètre. Ces rouages sont plongés dans un environnement et connectés par deux interfaces : en entrée, par l'interface exo-endochrone; en sortie, par le processus externe DISPLAY_TIME.

Considérons le mécanisme de cette horloge simulé le processus GEAR présenté en figure B.1. Ce processus est constitué des deux rouages d'une horloge minute-seconde; chacun de ces derniers est engendré par l'instance d'un processus WHEEL (voir figure B.2).

Le processus WHEEL est construit en composant deux entités. La première, représentée dans la boîte inférieure de la figure B.2, met en œuvre un compteur. Ce compteur qui réalise une remise à zéro à l'occurrence du signal d'entrée RESET possède un cadence

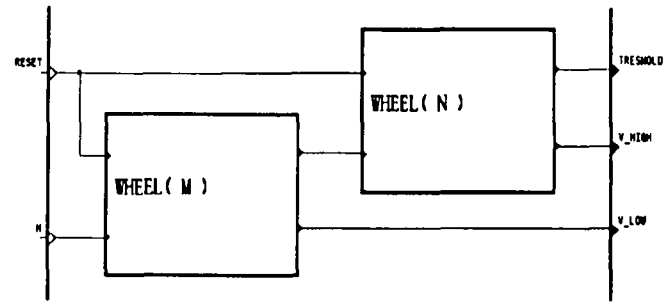


Figure B.1 : Processus GEAR

d'incrémentation laissée libre (non définie). Une occurrence sur le signal de sortie HS est produite lors du dépassement du seuil V_MODULO (paramètre).
L'entité de la boîte supérieure définit le cadencement du compteur comme l'union des occurrence des signaux RESET et H. Cette entité impose également une remise à zéro lors du dépassement de seuil : le signal IRESET (RESET interne) est présent à l'occurrence des signaux RESET (externe) et HS (interne).

Le processus WHEEL n'est autre qu'un compteur modulo de paramètre V_MODULO. En sortie, ce processus WHEEL possède le signal V des valeurs du compteur et le signal HS qui indique le dépassement du seuil donc la remise à zéro du compteur.

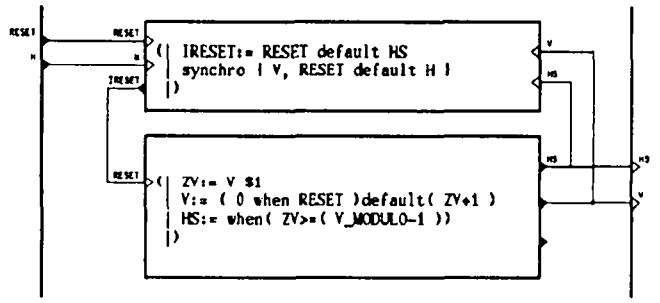


Figure B.2 : Processus WHEEL

La composition de ces deux rouages dans le processus GEAR réalise un compteur à deux positions, l'une modulo M, l'autre modulo N. Notons que WHEEL(N) est cadencé par le signal HS du compteur précédent; ainsi, chaque remise à zéro interne à l'instance WHEEL(M) induit une incrémentation dans le compteur WHEEL(N).

En sortie, le signal V_LOW donne la valeur de la position basse de GEAR, le signal V_HIGH celui de la position haute. Du point de vue des relations temporelles, le signal

V_HIGH n'est présent que lorsque V_LOW est remis à zéro.

B.1.2 L'interfaçage

En sortie

L'affichage des deux positions V_LOW et V_HIGH du processus GEAR est effectuée par le processus DISPLAY (voir figure B.3). L'environnement cible est supposé posséder un affichage non rémanent : les deux positions doivent être rafraîchies d'une manière synchrone.

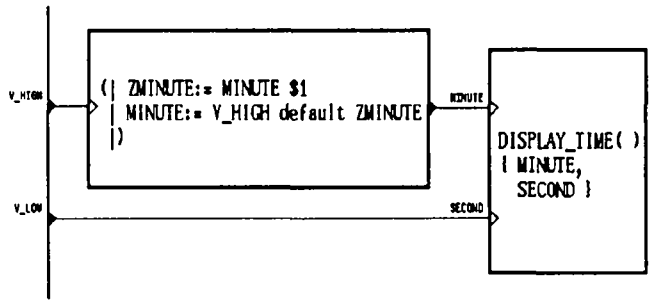


Figure B.3 : Processus DISPLAY

Le signal V_HIGH étant moins fréquent que V_LOW, il doit être recalé sur ce dernier. Ce recalage s'effectue par duplication des valeurs de V_HIGH à la fréquence de V_LOW : la boîte de gauche dans la figure B.3 réalise la recopie de valeurs; la fréquence des recopies est contrainte par le processus externe DISPLAY_TIME qui impose la synchronisation de ses entrées.

La composition des deux processus GEAR et DISPLAY définit le processus TIMER (voir figure B.4) qui possède en entrée les signaux RESET (remise à zéro) et H (cadencement de l'horloge).

En entrée

Le processus TIMER est exochrone : les entrelacements des occurrences signaux H et RESET sont non contraints en interne. La simulation d'un tel processus nécessite la création d'une interface exo-endochrone (cf II.6 pour les considérations générales sur de telles interfaces).
Dans cet exemple, l'interface présentée dans la boîte de gauche de la figure B.5 a été définie de façon à réduire le nombre de signaux d'entrée.
Le signal d'entrée ONRESET est un signal booléen dont la fréquence définit le cadencement H (secondes); ce signal ONRESET indique une remise à zéro de l'horloge (RESET) à l'occurrence de ses valeurs vraies.

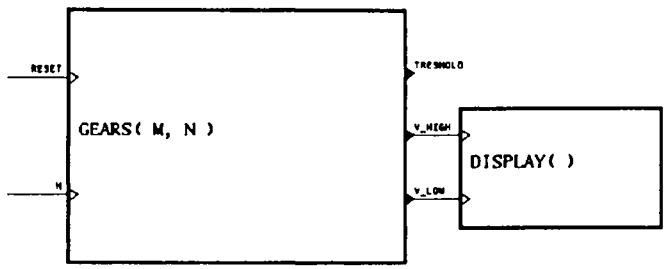


Figure B.4 : Processus TIMER

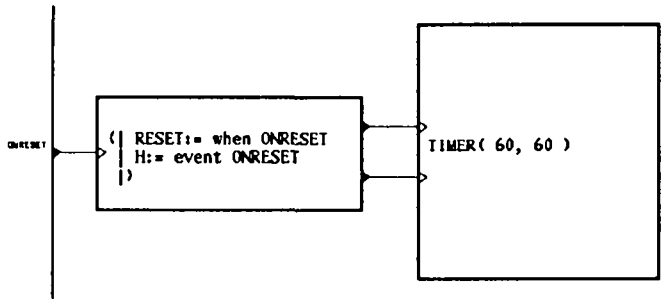


Figure B.5 : L'interface exo-endochrone

B.2 Représentation interne

B.2.1 Modélisation du contrôle

Processus WHEEL

Expression SIGNAL	Codage
IRESET := RESET **default** HS **synchro** {V,RESET **default** H} ZV := V $ 1 V := (0 **when** IRESET) **default** ZV+1 HS := **when** (ZV $\geq$ V_MODULO-1)	$ireset^2 = reset^2 + hs^2(1 - reset^2)$ $v^2 = reset^2 + h^2(1 - reset^2)$ $zv^2 = v^2$ $v^2 = ireset^2 + zv^2(1 - ireset^2)$ $hs^2 = -c - c^2$ $c \equiv ZV \geq V\_MODULO - 1$

Processus GEAR

Une copie du codage induit par le processus WHEEL est associée aux instances WHEEL(M) et WHEEL(N). Les variables x du codage du processus WHEEL sont indicées par x_m et x_n pour chacune des instances de WHEEL. Quand les indices peuvent être évités en réalisant les renommages, ceux-ci sont effectués. Par exemple, l'entrée H de WHEEL(N) n'est pas codé par la variable h_n mais par hs_m conformément au renommage.

Expression SIGNAL	Codage
IRESETm := RESET **default** HSm **synchro** {V_LOW,RESET **default** H} ZVm := V_LOW $ 1 V_LOW := (0 **when** IRESETm) **default** ZVm+1 HSm := **when** (ZVm $\geq$ M-1)	$ireset_m^2 = reset^2 + hs_m^2(1 - reset^2)$ $vlow^2 = reset^2 + h^2(1 - reset^2)$ $zv_m^2 = vlow^2$ $vlow^2 = ireset_m^2 + zv_m^2(1 - ireset_m^2)$ $hs_m^2 = -c_m - c_m^2$ $c_m \equiv ZVm \geq M - 1$
IRESETn := RESET **default** TRESHOLD **synchro** {V_HIGH,RESET **default** HSm} ZVn := V_HIGH $ 1 V_HIGH := (0 **when** IRESETn) **default** ZVn+1 TRESHOLD := **when** (ZVn $\geq$ N-1)	$ireset_n^2 = reset^2 + treshold^2(1 - reset^2)$ $vhigh^2 = reset^2 + hs_m^2(1 - reset^2)$ $zv_n^2 = vhigh^2$ $vhigh^2 = ireset_n^2 + zv_n^2(1 - ireset_n^2)$ $treshold^2 = -c_n - c_n^2$ $c_n \equiv ZVn \geq N - 1$

Interface exo-endochrone

Expression SIGNAL	Codage
RESET := **when** ONRESET H := **event** ONRESET	$reset^2 = -onreset - onreset^2$ $h^2 = onreset^2$

Processus DISPLAY

Expression SIGNAL	Codage
ZMINUTE := MINUTE $ 1 MINUTE := V_HIGH **default** ZMINUTE DISPLAY_TIME { MINUTE, SECOND }	$zminute^2 = minute^2$ $minute^2 = vhigh^2 + zminute^2(1 - vhigh^2)$ $vlow^2 = minute^2$

Les relations des tables ci-avant sont réorganisées dans la table B.1 comme suit.

- Les classes d'équivalence de variables sont nommées par un signal de type event de la classe; si aucun signal de ce type existe, un identificateur d'horloge est crée (les numéros d'identificateurs d'horloges sont ceux engendrés par le compilateur).

- Les définitions des signaux d'une même classe d'équivalence sont regroupées dans la troisième colonne de la table B.1.

- Ces définitions portant sur des variables sont réécrites dans le treillis des horloges $< C, \vee, . >$. Les relations d'horloge obtenues sont données dans la dernière colonne de la table B.1. Cette colonne possède des polynômes du type $1 - h$ qui correspondent à la négation de l'ensemble des conditions qui définissent h : $1 - reset^2 = onreset - onreset^2$.

Horloge	Classe d'équivalence	Définition	Rel. Horloge
h_4	$onreset^2, vlow^2, zv_m^2$	$reset^2 + h^2(1 - reset^2)$	
	$minute^2, zminute^2, h^2$	$ireset_m^2 + zv_m^2(1 - ireset_m^2)$	
		$vhigh^2 + zminute^2(1 - vhigh^2)$	
$reset^2$		$-onreset - onreset^2$	
h_{17}			$h_4.(1 - ireset_m^2)$
hs_m^2		$-c_m - c_m^2$	
h_{27}			$hs_m^2.(1 - reset^2)$
$ireset_m^2$	$zv_n^2, vhigh^2$	$reset^2 + hs_m^2(1 - reset^2)$	$reset^2 \vee hs_m^2$
		$ireset_n^2 + zv_n^2(1 - ireset_n^2)$	
$ireset_n^2$		$reset^2 + treshold^2(1 - reset^2)$	$reset^2 \vee treshold^2$
$treshold^2$		$-c_n - c_n^2$	

Tableau B.1 : Le contrôle

La hiérarchisation de l'ensemble des horloges se définit en deux points :

- chaque extraction de condition est placée sous l'horloge de la condition. Par exemple, l'horloge h_7 qui est l'extraction des instants à vrai de **onreset** est placée sous l'horloge de **onreset** nommée h_4. Le placement des horloges $h_7, reset^2, hs_m^2, h_{20}, treshold^2$ est ainsi réalisé.
- une horloge définie par une borne supérieure ou une borne inférieure d'horloges est placée à l'embranchement des horloges des conditions qui la définissent. Par exemple, l'horloge h_{42} est définie par la borne supérieure de h_7 et h_{20}, l'embranchement maximal dans l'arbre est h_4. Ainsi, h_{42} est une fille de h_4 dans l'arbre d'horloges.

La hiérarchie des horloges est représentée dans la figure B.6; celle-ci est définie par un arbre de racine h_4 qui est l'horloge la plus rapide du programme : le programme associé est endochrone.

B.2.2 Graphe aux dépendances conditionnées

Le graphe aux dépendances conditionnées est obtenu par analyse des dépendances de données définies dans le programme avec expansion des sous-processus. La présence des signaux xxx_m et xxx_n provient de l'expansion des instances du processus WHEEL avec les paramètres M et N.

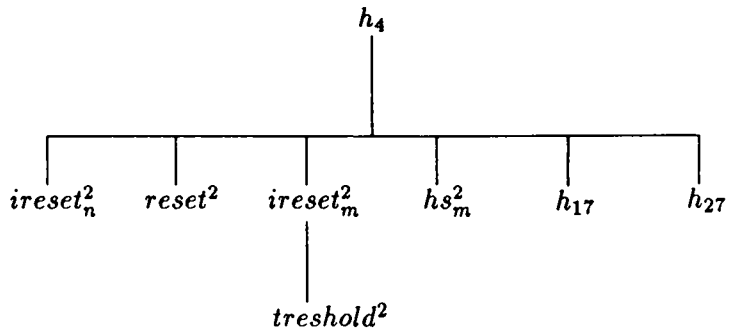


Figure B.6 : La hiérarchie d'horloges

L'application h_Γ associe la condition de validité à chaque dépendance. Elle se représente schématiquement par $x \xrightarrow{h} y$ pour (x, y) un arc tel que $h_\Gamma(x, y) = h$. L'application h_N est déduite des deux première colonnes de la table B.1.

$$\begin{aligned} h_N^{-1}(h_4) &= \{\texttt{onreset}, \texttt{v\_low}, \texttt{zv}_\texttt{m}, \texttt{minute}, \texttt{zminute}\} \\ h_N^{-1}(ireset_m^2) &= \{\texttt{v\_high}, \texttt{zv}_\texttt{n} \qquad\qquad\qquad\qquad\quad \} \end{aligned}$$

Le graphe aux dépendances conditionnées obtenu est représenté par la figure B.7

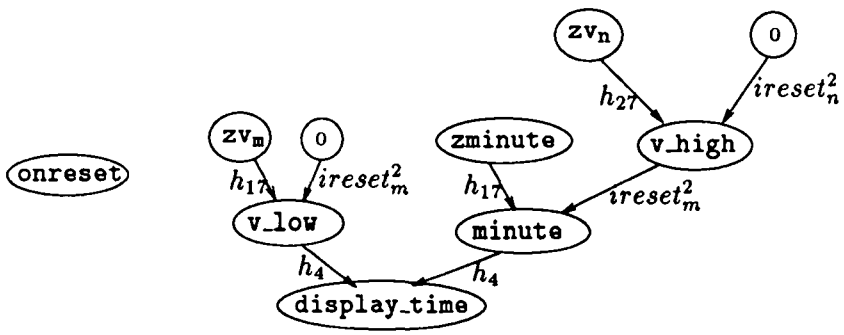


Figure B.7 : Graphe de dépendances conditionnées

B.2.3 Graphe dynamique

Le graphe dynamique de l'horloge avec remise à zéro est présenté dans la figure B.8. Ce schéma inclut (1) le graphe, (2) la hiérarchie d'horloge et (3) les relations entre ces deux structures.

(1) Le graphe aux dépendances conditionnées est partitionné selon h_N; le sous-graphe des nœuds synchrones à x est appelé G_x. L'attachement de ces sous-graphes à leur horloge est modélisée, dans la figure B.8, par les rectangles de tirets.

(2) la hiérarchie est identique à celle présentée dans la figure B.6. Les dépendances entre les définitions d'horloges ne sont pas dessinées afin de ne pas surcharger le schéma.

(3) Les connexions de (1) vers (2) sont représentées par les applications h_N et h_Γ.

De (2) vers (1), les connexions sont obtenues par les dépendances en pointillés qui expriment la définition d'une horloge sur une condition d'un signal du graphe.

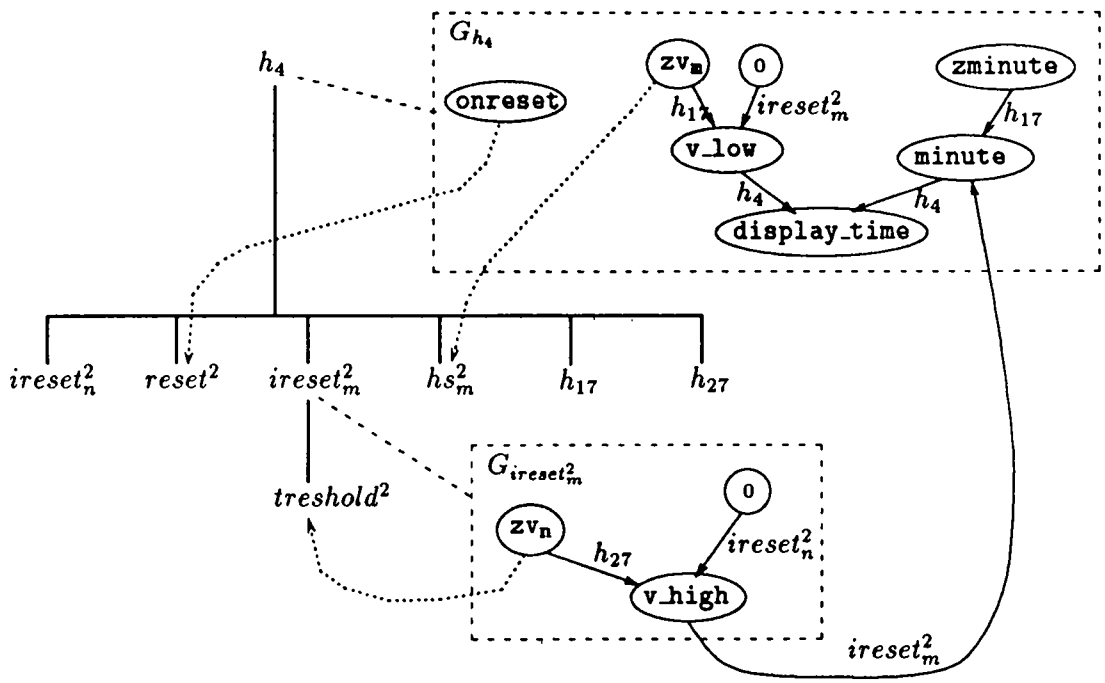


Figure B.8 : Le graphe dynamique

B.3 Décompilation en SIGNAL

Dans cette décompilation en SIGNAL d'un graphe dynamique, un processus H_i_H représente le sous-graphe dynamique de racine h_i. Pour toute horloge h_i, un identificateur h_i_h de signal est attribué; ce signal définit des événements : il est du type **event**.

```
process AS_RST_CLOCK_TRA=
   {?logical ONRESET_1
    !}
   (| (| H_4 := event ONRESET_1
      | H_4 ()
      |)
    |)
   where
      event H_4
   process H_4=
      {?event H_4;
        logical ONRESET_1
       !}
      (| synchro {H_4, V_LOW_11, ZV_23, MINUTE_34, ZMINUTE_35}
       | (| RESET_5 := when ONRESET_1
         |)
       | (| H_17_H := when ((not IRESET_21) default H_4)
         |)
       | (| HS_15 := when (ZV_23>= (60-1))
         |)
       | (| H_27_H := when ((not IRESET_29) default HS_15)
         |)
       | (| IRESET_21 := RESET_5 default HS_15
         | synchro {IRESET_21, V_HIGH_12}
         | IRESET_21 ()
         |)
       | (| IRESET_29 := RESET_5 default TRESHOLD_13
         |)
       | (| ZV_23 := V_LOW_11 $1
         | ZMINUTE_35 := MINUTE_34 $1
         | V_LOW_11 := (0 when IRESET_21) default ((ZV_23+1) when H_17_H)
         | MINUTE_34 := (V_HIGH_12 when IRESET_21) default (ZMINUTE_35 when
                  H_17_H)
         | DISPLAY_TIME {MINUTE_34, V_LOW_11}
         |)
      |)
```

```
      where
         event IRESET_29, IRESET_21, TRESHOLD_13, H_27_H, HS_15, H_17_H,
             RESET_5;
         integer V_LOW_11, V_HIGH_12, ZV_23 init 0, MINUTE_34, ZMINUTE_35
             init 0
      process IRESET_21=
         {? event IRESET_29, IRESET_21, H_27_H
          ! event TRESHOLD_13;
            integer V_HIGH_12}
         (| synchro {IRESET_21, ZV_31}
          | (| TRESHOLD_13 := when (ZV_31>= (60-1))
            |)
          | (| ZV_31 := V_HIGH_12 $1
            | V_HIGH_12 := (0 when IRESET_29) default ((ZV_31+1) when
                    H_27_H)
            |)
          |)
         where
            integer ZV_31 init 0
      end
   end;
   function DISPLAY_TIME=
      {? integer V_HIGH, V_LOW
       !}
end
```

B.4 Code séquentiel Fortran

B.4.1 Module d'instant

Ce module simule un instant; il contient deux sous-routines :

- La procédure d'initialisation IASRS qui contient l'affectation des valeurs initiales (instant $t = 0$) des signaux retardés et les affectations des signaux à valeur constante.
- La procédure centrale CASRS qui réalise une simulation d'un instant $t \geq 1$.

```
C
      SUBROUTINE SASRS
C  Declaration des signaux
      LOGICAL H4H,ONRE1
      INTEGER VLO15,VHI16,ZV29,ZV36,ZMI41,MIN42
C  Declaration des horloges
      LOGICAL TRE17,H26,H122,IRE34,IRE27
C  Corps de la procedure d'initialisations
      ENTRY IASRS
      ZV29=0
      ZV36=0
      ZMI41=0
      RETURN
C  Corps du programme
      ENTRY CASRS(H4H)
      H26= .TRUE.
      CALL RONRESET(ONRE1,H4H)
      IF ( .NOT. (H4H))RETURN
      H122=((ZV36) .GE. ((60)- 1))
      IRE34=((ONRE1) .OR. H122)
      TRE17= .FALSE.
      IF (IRE34)THEN
      VLO15=0
      ELSE
      VLO15=(ZV36)+ 1
      ENDIF
      ZV36=VLO15
      IF (IRE34)THEN
      TRE17=((ZV29) .GE. ((60)- 1))
      ENDIF
      IRE27=((ONRE1) .OR. TRE17)
      IF (IRE34)THEN
      IF (IRE27)THEN
      VHI16=0
      ELSE
      VHI16=(ZV29)+ 1
```

```
ENDIF
ZV29=VHI16
ENDIF
IF (IRE34)THEN
MIN42=VHI16
ELSE
MIN42=ZMI41
ENDIF
ZMI41=MIN42
CALL DISPLAY_TIME (MIN42,VLO15)
RETURN
END
```

B.4.2 Autres Modules

- **Modules d'itération**

  Ce module réalise l'itération des instants. Il est constitué de l'appel à la procédure d'initialisation et d'une itération sur l'appel à la procédure d'instant.

- **Module d'entrées/sorties**

  Ce module contient les procédures de lecture des entrées et d'émission des sorties. Ces E/S sont réalisées par l'intermédiaire de procédures aux noms des E/S et préfixés par "R" (resp. "W") pour les signaux en entrée (resp. en sortie). En cas de changement dans l'environnement d'exécution d'un programme, seul ce module est modifié.

Bibliographie

[1] Benveniste (A.), Le Goff (B.) et Le Guernic (P.). – *Hybrid Dynamical Systems theory and the language* SIGNAL. – Research Report n° 838, Rocquencourt, INRIA, April 1988.

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

[3] Berry (G.) et Cosserat (L.). – The ESTEREL synchronous programming language and its mathematical semantics. *In: Seminar on Concurrency*, éd. par Brookes (S. D.), Roscoe (A. W.) et Winskel (G.), pp. 389–448. – Lecture Notes in Computer Science, 197, Springer-Verlag, 1985.

[4] Besnard (L.). – *Mise en œuvre séquentielle de* SIGNAL *un langage orienté flot de données, synchrone pour applications temps-réel.* – FRANCE, Thèse de PhD, Université de Rennes 1, 1991 A paraître.

[5] Caspi (P.), Pilaud (D.), Halbwachs (N.) et Plaice (J. A.). – LUSTRE: A declarative language for programming synchronous systems. *In: 14th ACM Symposium on Principles of Programming Languages*, pp. 178–188. – Munich, 1987.

[6] Chéron (B.). – *Transformations syntaxiques de programmes* SIGNAL. – FRANCE, Thèse de PhD, Université de Rennes 1, Septembre 1991.

[7] Couronné (P.), Saint (J.) et Plaice (J.). – *The Esterel-Lustre OC Portable Format.* – Technical report, Sophia-Antipolis, France, Ecoles des mines / INRIA, 1990.

[8] Gautier (T.), Le Guernic (P.) et Besnard (L.). – SIGNAL*: a declarative language for synchronous programming of real-time systems.* – Research report n° 761, Rocquencourt, INRIA France, November 1987.

[9] Ghezal (N.), Matiatos (S.), Piovesan (P.), Sorel (Y.) et Sorine (M.). – *SYNDEX un environnement de programmation pour multi-processeur de traitement du signal. Mécanismes de communication.* – Research report n° 1236, Rocquencourt, France, INRIA, Juin 1990.

[10] Gondran (M.) et Minoux (M.). – *Graphs et algorithms.* – Wiley and Sons, John, 1984, *Wiley-Interscience series in discrete mathematics.*

[11] Gondran (M.) et Minoux (M.). – *Graphes et algorithmes.* – 61,Bd Saint-Germain, Paris 5, France, EYROLLES, 1985, second édition, *Direction des études et recherches d'électricité de France.*

[12] Harel (D.). – STATECHARTS: A visual formalism for complex systems. *Science of Computer Programming*, vol. 8, n° 3, June 1987, pp. 231–274.

[13] Le Borgne (M.), Benveniste (A.) et Le Guernic (P.). – Polynomial ideal theoretic methods in discrete event, and hybrid dynamical systems. *In: Proc. 1989 28th IEEE Conf. on Decision and Control.* pp. 2695–2700. – Tampa, FL, 1989.

[14] Le Goff (B.). – *Inférence de contrôle hiérarchique : application au temps-réel.* – Thèse de PhD, Université de Rennes 1, 1989.

[15] Le Maire (C.). – *Le langage* SIGNAL *: Un Exemple en Segmentation Automatique de la Parole Continue.* – Research Report n° 1217, Rennes, INRIA, Avril 1990.

[16] Maffeïs (O.). – Traduction de programmes SIGNAL en OCCAM. – Septembre 1989. DEA informatique, IRISA, Université de Rennes 1.

LISTE DES DERNIERES PUBLICATIONS INTERNES IRISA

PI 610 SYNCHRONIZATION AND CONCURRENCY MEASURES FOR DISTRIBUTED COMPUTATIONS
Michel RAYNAL
Octobre 1991, 20 pages.

PI 611 MALI v06 - TUTORIAL AND REFERENCE MANUAL
Olivier RIDOUX
Octobre 1991, 86 pages.

PI 612 SENSITIVITY COMPUTATION IN NETWORK RELIABILITY ANALYSIS
Gerardo RUBINO
Octobre 1991, 38 pages.

PI 613 OPAC : A FLOATING-POINT COPROCESSOR DEDICATED TO COMPUTE-BOUND KERNELS
André SEZNEC, Karl COURTEL
Octobre 1991, 28 pages.

PI 614 CONTROLLING AND SEQUENCING AN HEAVILY PIPELINED FLOATING-POINT OPERATOR
André SEZNEC, Karl COURTEL
Octobre 1991, 28 pages.

PI 615 ON FAULT-TOLERANT SYMBOLIC COMPUTATIONS
Bernard DELYON, Oded MALER
Novembre 1991, 18 pages.

PI 616 USING COHERENCE TO ACCELERATE RADIOSITY
Pierre TELLIER, Eric MAISEL, Kadi BOUATOUCH, Eric LANGUENOU
Novembre 1991, 16 pages.

PI 617 INTERVAL APPROXIMATIONS OF MESSAGE CAUSALITY IN DISTRIBUTED EXECUTION
Claire DIEHL, Claude JARD
Novembre 1991, 44 pages.

PI 618 RETOUR SUR LE RESEAU SYSTOLIQUE DU PALINDROME
Hervé LE VERGE, Patrice QUINTON
Novembre 1991, 14 pages.

PI 619 TRANSFORMATIONS DU GRAPHE DES PROGRAMMES SIGNAL
Olivier MAFFEIS, Bruno CHERON, Paul LE GUERNIC
Novembre 1991, 82 pages.

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

ISSN 0249 - 6399