N° d'ordre: 1967

THÈSE

présentée

DEVANT L'UNIVERSITÉ DE RENNES 1

pour obtenir

le grade de : *Docteur de l'Université de Rennes 1*

Mention : Informatique

par

Irina Mădălina SMARANDACHE

Équipe d'accueil : IRISA
École Doctorale : Informatique, Traitement du Signal et Télécommunications
Composante universitaire : IFSIC

Transformations affines d'horloges :
application au codesign de systèmes temps-réel en utilisant les langages
SIGNAL *et* ALPHA

SOUTENUE LE 1 Octobre 1998 devant la commission d'Examen

M. :	Jean-Pierre	BANÂTRE	Président
MM. :	Nicolas	HALBWACHS	Rapporteurs
	Éric	MARTIN	
MM. :	Yves	ROBERT	Examinateurs
	Thierry	GAUTIER	
	Paul	LE GUERNIC	

Remerciements

Le travail présenté dans ce document est le fruit de trois années d'exploration dans un monde plein de défis, celui de la recherche. Il est le résultat principalement d'une grande passion et de beaucoup de volonté, mais il n'aurait pas été possible dans la forme d'aujourd'hui sans des contributions particulières auxquelles je dédie cette page de remerciements.

Je tiens à remercier premièrement Jean-Pierre Banâtre pour m'avoir fait l'honneur de présider le jury de ma soutenance. Je remercie tout particulièrement Nicolas Halbwachs pour la patience avec laquelle il a parcouru la première variante du manuscrit et pour ses commentaires précieux. Je remercie ensuite Éric Martin pour avoir accepté de rapporter un travail très formel et d'évaluer son importance dans le contexte pratique proposé. Je tiens à remercier enfin Yves Robert pour sa disponibilité à juger ce travail.

Un grand merci, Paul, pour m'avoir accueilli dans le projet et m'avoir offert un sujet de thèse à la foi intéressant et plein de défis. Je tiens à te remercier tout particulièrement pour la confiance que tu m'as accordée pendant toutes ces années. Finalement, ce travail n'aurait jamais pu exister dans la forme d'aujourd'hui sans l'aide de Thierry. L'ambiance détendue et pleine d'humour qui régnait dans notre bureau pendant ma dernière année de thèse, ses encouragements permanents, sa lecture attentive des moindres détails de mon document et nos discussions permanentes m'ont permis de garder la tête claire, de progresser et d'aboutir à un travail complet dans la forme présente.

Mais cette thèse n'aurait pas existé sans le soutien moral de tous les membres du projet EP-ATR. Collègues et amis, je tiens à remercier particulièrement : Loïc, pour nos longues discussions sur le calcul d'horloges et la patience dont il m'a fait preuve, Patricia pour son amitié, son sourire permanent et l'optimisme qu'elle dégage par sa présence, Yan-Mei pour la confiance qu'elle a eu en moi et ses encouragements constants aux moments les plus difficiles, Pascal pour m'avoir montré comment on peut personnaliser sa soutenance et pour sa disponibilité à me fournir les tuyaux nécessaires pour réussir la mienne, Bernard pour les moments partagés dans le même bureau au soleil au début de ma thèse, Apos pour son soutien "balkanique" accompagné toujours d'un bon repas, Hervé pour sa disponibilité permanente, Jean-René pour l'ambiance relaxée et à la foi provocatrice de nos repas de midi, Sylvain pour m'avoir "illuminé" le chemin...

Je tiens tout spécialement à mentionner les membres de l'équipe API grâce auxquels la "coopération" SIGNAL-ALPHA s'est prouvée un succès : Patrice pour avoir vérifié ma compréhension du langage ALPHA et de son environnement, Tanguy et Sanjay pour leur présence et disponibilité sans lesquelles le project CAIRN n'aurait pu exister, Franck pour sa contribution à l'interfaçage des deux langages...

Je ne saurai jamais remercier assez ma sœur Roxi et mes parents qui, même de loin, n'ont cessé de penser à moi et de m'encourager pendant cette période. Je ne serais jamais arrivée là sans leur amour et confiance.

Enfin, c'est grâce à Peter que j'ai commencé cette thèse que je finis aujourd'hui. Il n'a pas cessé un seul instant de m'encourager, de me soutenir dans les moments les plus difficiles, de me motiver à me dépasser, de croire en moi... son amour, son courage, son optimisme, sa force, sa détermination ont été des sources inépuisables dans lesquelles j'ai puisé toute ma volonté. Un grand merci du fond du cœur.

À mes parents, à Roxi et à Peter...

Table des matières

Table des figures

Introduction

Le travail décrit dans ce document s'inscrit parmi les actions de recherche effectuées dans le cadre du projet EP-ATR de l'IRISA. L'objectif principal du projet est la mise en œuvre, autour du langage SIGNAL, d'un environnement de conception et d'implémentation d'applications temps-réel critiques [LLGL91]. Il s'agit de fournir un cadre unifié de spécification, validation et implémentation d'applications, en s'appuyant sur les fortes propriétés du langage induites par son caractère synchrone et flots de données. Cela a donné naissance à un ensemble d'outils basés sur des méthodes et techniques extrêmement variées visant la preuve formelle, la compilation de programmes, la génération de code synthétisable et l'implémentation logicielle, la distribution sur des architectures multi-processeurs hétérogènes, la conception conjointe matériel/logiciel etc.

Dans ce contexte, nos travaux ont visé l'extension des concepts formels sur lesquels repose le langage SIGNAL dans le but principal d'augmenter ses capacités de spécification et validation de programmes [GLQ+97] [SL97a]. Pour cela, nous avons défini en SIGNAL des mécanismes théoriques et pratiques de support pour les *transformations affines* d'horloges qui ont une application immédiate dans la conception conjointe matériel/logiciel d'applications de traitement du signal ou des images.

Contexte

Les actions menées dans le cadre du projet EP-ATR visent à construire des méthodes et logiciels pour la conception et la mise en œuvre de systèmes embarqués, spécialement dans le domaine du temps-réel. Le domaine d'application va des petits circuits intégrés spécialisés et micro-contrôleurs, jusqu'aux applications de l'aéronautique, des télécommunications, du domaine militaire ou des processus industriels. Les systèmes mis en œuvre sont souvent critiques du point de vue des propriétés de sûreté, ont un comportement temporel complexe, et peuvent être implémentés sur une variété d'architectures support : circuits intégrés, architectures multi-processeurs hétérogènes, processeurs de traitement du signal, etc.

Le modèle de programmation synchrone – qui comprend des langages comme STATECHARTS [Har87], ESTEREL[BD91], LUSTRE[HCRP91] et SIGNAL[LGLL91] – a été créé dans le but de fournir une approche unitaire et structurée pour la conception de systèmes embarqués réactifs, c'est-à-dire de systèmes qui interagissent continuellement avec leur environnement en exécutant diverses actions comme réponses aux stimuli externes. Comme exemples, un système de compression/décompression d'images, le système informatique d'un automobile ou le contrôleur d'une photocopieuse représentent des systèmes embarqués réactifs. Si, de surcroît, les réponses d'un tel système doivent parvenir dans un délai de temps borné et spécifié à l'avance, il s'agit de systèmes temps-réel.

Le traitement du temps physique et le non-déterminisme soulèvent des problèmes complexes dans la conception et l'implémentation de systèmes embarqués temps-réel. Ceci est le cas dans des langages temps-réel (ADA[Geh84], OCCAM[INM87]) dans lesquels la durée effective d'une instruction est difficile à prévoir. Dans l'approche synchrone, le temps physique a été remplacé par un temps *logique* caractérisé uniquement par l'ordre relatif des instants (leur succession) et leur simultanéité. Concrètement, dans un langage de programmation synchrone on admet les hypothèses suivantes :

1) les actions (communications et calculs) s'effectuent en temps nul ;

2) deux ou plusieurs actions peuvent avoir lieu à un même instant logique (elles sont simultanées).

Les hypothèses de synchronisme confèrent aux langages synchrones une sémantique formelle qui permet la vérification des propriétés logiques d'un système avant de prendre en considération les propriétés physiques liées à une implémentation particulière [BB91a].

En Signal, la spécification d'une application se réalise de manière modulaire par composition de systèmes d'équations [LLGL91]. Un programme Signal peut ensuite subir des transformations syntaxiques ou architecturales dans le but d'adapter une première description fonctionnelle à une architecture cible logicielle ou matérielle [Aub97] [Bel94a] [KW96]. A toutes les phases du processus de conception, il est possible de vérifier formellement des propriétés concernant le contrôle – synchronisations et logique – et les dépendances de données [Bes92] [Ama95] [Mar97]. L'environnement de Signal intègre ainsi une méthodologie de conception d'applications temps-réel critiques par : 1) raffinement progressif de spécifications afin de tester diverses implémentations possibles par simulation/vérification formelle à différents niveaux d'abstraction ; 2) implantation de l'architecture logicielle finale sur des supports éventuellement asynchrones et distribués, en relâchant au besoin la contrainte de synchronisme, ou sur des composants matériels ou hybrides matériel/logiciel, en utilisant Signal conjointement avec des langages de description de matériel.

En ce qui concerne les techniques de preuve formelle, un calcul sur les synchronisations des variables est effectué dès les premières phases de compilation d'un programme [Ama95]. Le compilateur devient ainsi un outil formel adapté particulièrement – par les techniques qu'il intègre – pour la spécification et la validation d'applications dans lesquelles prédominent les parties contenant du contrôle. Comme on a pu le constater dans un contexte de conception conjointe d'applications de traitement des images à base de Signal, les synchronisations des parties contenant des calculs numériques complexes nécessitent un calcul formel particulier qui prendrait en compte le comportement temporel régulier des variables impliquées [GLQ+97]. Un calcul formel dédié à ce problème est apparu nécessaire afin d'étendre le domaine d'application du langage et dans le but aussi de son intégration dans une chaîne de conception mixte matériel/logiciel (codesign) à côté de langages de description de matériel ou d'autres langages flots de données comme Alpha [SL97a] [SL97b].

Buts et motivations

Une application de traitement du signal ou des images suffisamment complexe contient des parties importantes de traitement numérique reliées par du contrôle. La programmation d'une application de compression/décompression d'images – la chaîne de codage vidéo à 34 Mbits/s [Sch92] – en Signal a permis de relever certaines des « faiblesses » actuelles du langage. Signal dispose de mécanismes réduits pour la description et la manipulation de structures de données multidimensionnelles, comme les matrices ou les vecteurs, et pour leur implémentation efficace. De plus, le raffinement de spécifications correspondant aux diverses implémentations de ces traitements est rendu difficile actuellement en Signal par un calcul formel sur les synchronisations adapté plutôt pour la manipulation du contrôle. En conséquence, le but de notre travail de thèse a été de proposer des solutions adaptées aux problèmes ci-dessus et d'intégrer nos solutions dans l'environnement de conception existant autour de Signal.

Nous avons ainsi proposé un contexte de conception conjointe d'applications en utilisant les langages Signal et Alpha. Développé au sein du projet Api de l'Irisa, Alpha est apparu comme un langage complémentaire à Signal conçu pour la description et l'implémentation d'algorithmes qui peuvent être décrits par des récurrences affines, parmi lesquels se situent la majorité des algorithmes de calcul sur des structures multi-dimensionnelles [Wil94] [CFG+92]. Outre le grand nombre de traitements numériques qui peuvent être décrits en Alpha, le langage dispose d'un environnement de synthèse automatique ou semi-automatique d'architectures intégrées spécialisées, ainsi que de techniques de génération de code optimisé pour l'implémentation logicielle d'une application [WS94] [QRW95a].

Les langages Signal et Alpha sont apparus ainsi comme des langages complémentaires pour la spécification et l'implémentation d'applications contenant des traitements numériques importants et du contrôle. Cette observation a été le point de départ d'une étude sur l'interfaçage de ces deux langages dans un contexte de conception conjointe matériel/logiciel (codesign) d'applications irrégulières et régulières à plusieurs niveaux d'abstraction. Outre les problèmes liés à la définition d'un protocole de communication entre les deux langages (gestion des transferts des données, des buffers de mémorisation), le raffinement des spécifications mixtes et leur validation ont fait apparaître des *transformations affines* sur les horloges des variables Signal et des synchronisations sur les horloges transformées.

Dans ce cadre, notre travail de thèse a porté essentiellement sur la définition et l'implémentation du calcul formel sur les synchronisations des horloges obtenues par transformations affines, qui n'étaient pas traitées par les techniques de preuve formelle existantes dans le compilateur de Signal. Les nouvelles horloges représentent en fait des *sur-échantillonnages réguliers* d'horloges correspondant au fonctionnement en parallèle des circuits intégrés spécialisés asynchrones synthétisés à partir de Alpha. Du point de vue de Signal, il s'agit ainsi de synthétiser l'horloge globale du système en sa totalité, ou de vérifier les re-synchronisations des résultats produits indépendamment par chaque circuit. Nous avons conçu le calcul de ces synchronisations dans un contexte formel défini par le *modèle sémantique des traces* du langage Signal et nous avons implémenté un prototype de vérification de nos résultats théoriques dans le compilateur actuel du langage Signal.

Contributions de notre thèse : Calcul formel sur les transformations affines d'horloges en Signal et application au codesign de systèmes mixtes réguliers et irréguliers

Le compilateur de Signal effectue des calculs formels sur la *synchronisation* des signaux et les *dépendances de données* afin de vérifier leur consistance et de produire du code exécutable. En ce qui concerne les synchronisations, il s'agit de résoudre un système d'équations sur horloges [1] dans le *treillis booléen*, en prenant aussi en compte les propriétés des horloges définies comme sous-échantillonnages par conditions booléennes [Bes92]. Au cours du processus de résolution (calcul d'horloges), il est nécessaire de vérifier que deux expressions d'horloges h_1 et h_2 peuvent être *synchronisées*. Dans ce but, les propriétés de l'espace des horloges peuvent s'exprimer sous la forme d'un système de *règles de synchronisabilité* sur lequel est basée toute vérification de la *synchronisabilité* de deux variables d'horloges h_1 et h_2.

1. L'horloge d'une variable représente l'ensemble des instants logiques de présence de la variable.

De manière générale, le problème de la synchronisabilité de deux horloges arbitraires h_1 et h_2 est non-décidable ; il peut être résolu sur des sous-ensembles de l'espace des horloges, comme c'était le cas pour le calcul d'horloges *booléen* existant dans le compilateur. Nous avons identifié dans cet espace un nouveau sous-ensemble : les horloges *sur-échantillonnages réguliers* qui peuvent être décrits comme les résultats des *transformations affines* d'horloges. Les propriétés des *relations affines* induites entre horloges par ces transformations nous ont permis de mettre en évidence un nouveau système de *règles de synchronisabilité* qui augmente la puissance formelle du compilateur.

Pour cela, nous avons proposé un modèle mathématique d'études de propriétés : *le modèle sémantique des traces* de SIGNAL introduit dans [LG91]. Dans ce modèle, nous avons déduit une série de propriétés des relations affines dans le contexte des *flots* (des traces particulières) et des processus (qui sont des sous-ensembles de flots associés aux programmes SIGNAL). Ces propriétés nous ont permis de prouver l'ensemble des théorèmes (règles) de synchronisabilité que nous avons ajouté aux règles existantes. Dans ce contexte, un apport important a été la définition d'une *forme canonique* des relations affines par rapport à la relation d'identité (qui est une relation d'équivalence) [SL97a] [SL97b]. Nous avons défini des opérations d'union, intersection et différence sur relations affines correspondant aux opérations homonymes qui peuvent être effectuées sur les horloges. Ceci nous a permis de retrouver des conditions simples de synchronisabilité basées sur un calcul d'entiers (n, φ, d) – φ positif et n, d strictement positifs – qui représentent les paramètres d'une transformation affine et de la relation affine induite par celle-ci.

Finalement, nous avons montré que le calcul d'horloges booléen est équivalent à un calcul de synchronisabilité et nous avons défini formellement l'intégration de notre calcul de synchronisabilité dans le calcul actuel. Nous avons proposé un prototype d'intégration de ces résultats dans le compilateur du langage SIGNAL et nous avons expérimenté notre prototype avec l'application de codage d'images vidéo à 34 Mbits/s.

L'étude des transformations affines d'horloges en SIGNAL représente un premier pas vers l'intégration dans le langage des techniques d'optimisation de code rencontrées dans les paralléliseurs automatiques [DPR94]. D'un côté, le calcul formel que nous avons proposé est basé sur la manipulation de fonctions affines de type $at + b,\ t \in \mathbb{N}$, et nous avons retrouvé des concepts et problèmes similaires à ceux existant dans le monde des $\mathbb{Z}$-polyèdres [Anc91] [QRR96b]. D'un autre côté, notre travail est une contribution au développement d'un environnement de synthèse de systèmes temps-réel critiques et nous avons pu tester nos résultats en utilisant SIGNAL conjointement avec ALPHA dans un contexte de codesign [GLQ+97]. L'idée était de pouvoir bénéficier en SIGNAL des techniques d'implémentation optimale de programmes existantes en ALPHA et qui ont été inspirées par les méthodes de parallélisation automatique de programmes [CFG+92] [BQR+90].

Finalement, nous avons proposé un calcul formel statique simple et élégant qui peut constituer le point de départ vers l'intégration en SIGNAL d'autres techniques de manipulation de structures de données régulières ; nous avons en vue les calculs formels sur les polyèdres convexes et $\mathbb{Z}$-polyèdres, comme ils apparaissent dans les méthodes d'implémentation optimale de programmes [DPR94] [QRR96a] [Fea96].

Organisation du document de thèse

Notre thèse est axée sur trois parties principales dont nous présentons succinctement le contenu:

Conception conjointe d'applications régulières et irrégulières

Nous commençons par un chapitre dédié aux langages synchrones en général et à la présentation de SIGNAL en particulier. Nous introduisons brièvement les principaux opérateurs de

SIGNAL et les outils existants dans son environnement. Nous concluons avec une analyse de l'adaptabilité de SIGNAL à la spécification et validation de systèmes temps-réel contenant des parties importantes de traitement de données, qui constituera le prélude et une motivation de notre travail de thèse sur les transformations affines de variables en SIGNAL.

Après un état de l'art dans le domaine des langages et formalismes orientés flots de données, nous introduisons le langage ALPHA et son environnement basé essentiellement sur un système semi-automatique de transformations de programmes en vue de la synthèse de circuits.

Le dernier chapitre de cette partie est dédié aux aspects concernant le codesign. Nous décrivons brièvement les principaux problèmes relatifs au codesign et proposons un cadre de spécification, validation et implémentation de systèmes temps-réel orientés traitement du signal ou des images en utilisant les langages SIGNAL et ALPHA. Nous présentons ensuite les problèmes dégagés suite à la mise en place de la coopération des deux langages et concluons sur l'introduction des transformations affines en SIGNAL.

Transformations affines d'horloges en SIGNAL

Le premier chapitre est dédié à la présentation du modèle sémantique des traces de SIGNAL; nous commençons par fixer les propriétés des traces et des flots et montrons comment chaque construction syntaxique de SIGNAL induit un *processus* défini comme un ensemble de flots qui satisfont un certain nombre de *contraintes*. Nous concluons avec un certain nombre de notions et propriétés dans le monoïde des processus : synchronisabilité, signaux identiques, horloges, vérification de propriétés.

Le chapitre suivant est entièrement dédié aux relations affines. Nous définissons un ensemble de propriétés liées à l'identité des relations affines et aux opérations d'union, intersection et différence de relations affines dans le contexte des flots que nous généralisons ensuite dans le contexte des processus. Suite à ce premier calcul, nous introduisons un ensemble de règles de synchronisabilité d'horloges qui permettront d'étendre le calcul *booléen* d'horloges existant. Nous montrons comment le concept de synchronisabilité se retrouve dans le calcul booléen et faisons apparaître le système de règles de synchronisabilité sur lequel celui-ci est basé. Nous proposons le calcul *affine* d'horloges basé sur le système de règles de synchronisabilité que nous avons déduit. Nous concluons avec le modèle de représentation d'une relation affine et la définition des opérateurs affines en termes de flots.

Implémentation des transformations affines d'horloges en SIGNAL

Le premier chapitre est une introduction à l'environnement de SIGNAL, avec une brève présentation des principales étapes de la compilation d'un programme SIGNAL, dont le calcul d'horloges en est une.

Dans le chapitre qui suit, nous décrivons la mise en œuvre existante du calcul booléen d'horloges – basée sur une représentation des équations d'horloges sous la forme d'une forêt d'arbres booléens – et décrivons l'intégration du calcul *affine* dans le calcul booléen, en utilisant comme support les axiomes du treillis d'ordre $\preceq_{aff}$.

Nous concluons avec un chapitre d'exemples simples d'utilisation du calcul affine d'horloges, parmi lesquels la chaîne de codage vidéo à 34 Mbits/s, afin d'illustrer l'apport de notre travail dans la spécification et validation d'applications de traitement du signal et des images avec SIGNAL.

Première partie

Conception conjointe d'applications régulières et irrégulières

Chapitre 1

Introduction

Nous précisons dans cette partie les éléments de base de la conception conjointe (codesign) d'applications temps-réel en utilisant les langages Signal (projet Ep-Atr) et Alpha (projet Api) développés à l'Irisa.

Le terme "codesign" dénote la spécification, la validation et l'implémentation d'un système spécialisé faisant appel à une partie matérielle spécifique, constituée d'un ou de plusieurs circuits intégrés spécialisés, et à une partie logicielle s'exécutant sur une architecture à base de processeurs programmables (par exemple, microprocesseurs ou processeurs de traitement du signal). Il s'agit d'explorer différentes architectures d'un système mixte matériel/logiciel en vue d'optimiser des critères de coût et/ou de performance.

Gérer les nombreux problèmes qui apparaissent dans le codesign – parmi lesquels la spécification et la modélisation de systèmes mixtes à différents niveaux d'abstraction ou l'implémentation des parties matérielle et logicielle et la génération de leur interface – nécessite la mise en place d'une méthodologie appropriée. Nous nous sommes proposé d'aborder une partie des problèmes du codesign, essentiellement la spécification et la vérification de systèmes et la synthèse matérielle, en s'appuyant sur les environnements existants autour de Signal et Alpha et en tirant parti des propriétés des deux langages qui les rendent adaptés au codesign.

1.1 Conception conjointe en utilisant les langages Signal et Alpha

Signal est un langage synchrone et équationnel utilisé pour la spécification, la validation et l'implémentation de systèmes réactifs et temps réel. A part son compilateur, qui constitue un outil de preuve de propriétés statiques, l'environnement de Signal contient, entre autres, un générateur de code C pour la simulation fonctionnelle, un générateur de code intermédiaire pour l'accès aux outils de vérification de propriétés dynamiques et des outils d'implémentation matérielle et logicielle de programmes. L'expérience a montré que Signal n'est particulièrement bien adapté que pour la programmation d'applications temps-réel dans lesquelles les parties contenant du contrôle sont prépondérantes. Cette caractéristique provient de l'absence d'une structure adéquate pour la programmation, la vérification et l'implémentation d'algorithmes de calcul spécialisés pour des traitements complexes, particulièrement concernant les structures de données multi-dimensionnelles.

Alpha est un langage fonctionnel pour la spécification et l'implémentation optimale d'algorithmes de traitement de vecteurs, matrices et généralement de structures régulières multi-dimensionnelles; il possède ainsi les propriétés complémentaires à Signal. Cette constatation a représenté le point de départ pour l'utilisation conjointe de Signal et Alpha pour la conception mixte matérielle/logicielle de systèmes. Notre approche repose sur le constat qu'une application de traitement du signal ou de l'image est composée de parties de traitements numériques que l'on peut spécifier en ALPHA et de parties contenant du contrôle qu'on peut décrire avec SIGNAL. Le contexte de conception conjointe que nous proposons sera basé sur deux points essentiels. En première instance, la spécification des traitements numériques en ALPHA permettra de disposer de l'environnement de programmation de ce langage en ce qui concerne la spécification, la simulation et l'implémentation – en circuits systoliques ou sur des processeurs programmables généraux – des calculs sur des structures multi-dimensionnelles. Le deuxième aspect concerne la validation de l'application entière à plusieurs niveaux d'abstraction, réalisée dans l'environnement du langage SIGNAL. Afin de rendre ce processus plus simple qu'il ne l'est actuellement en SIGNAL, il est apparu nécessaire d'introduire en SIGNAL le concept de *transformations affines* d'horloges et de définir un calcul formel sur les synchronisations des horloges transformées. Ce calcul permet le raffinement de spécifications SIGNAL induit par la prise en compte, tant du côté SIGNAL que du côté ALPHA, des détails architecturaux correspondant aux diverses implémentations possibles. Une spécification hétérogène SIGNAL - ALPHA peut être ensuite partitionnée conformément aux critères de coût et performances désirés et implémentée en circuits ou sur des processeurs généraux conformément au partitionnement. La synthèse des implémentations tirera parti des outils adaptés de SIGNAL et ALPHA. Nous n'avons pas traité les problèmes du partitionnement et de la génération de l'interface entre parties matérielle et logicielle, qui peuvent constituer à eux-mêmes des sujets de thèse indépendants.

1.2 Organisation de cette partie

Le plan de présentation de cette partie est le suivant :

Nous commençons par un chapitre dédié aux langages synchrones en général et à la présentation de SIGNAL en particulier. Nous introduisons brièvement le langage SIGNAL et son environnement. Nous présentons ensuite une étude de cas concernant l'utilisation de SIGNAL pour la programmation de systèmes temps-réel et introduisons les principales motivations de notre travail de thèse concernant les *transformations affines* d'horloges en SIGNAL.

Après un état de l'art dans le domaine des langages et formalismes orientés flots de données, nous introduisons le langage ALPHA et son environnement basé essentiellement sur un système semi-automatique de transformations de programmes en vue de la synthèse de circuits.

Le dernier chapitre de cette partie est dédié aux aspects concernant le codesign. Nous décrivons brièvement les principaux problèmes relatifs au codesign et proposons un cadre de spécification, validation et implémentation de systèmes temps-réel orientés traitement du signal ou de l'image en utilisant les langages SIGNAL et ALPHA. Nous présentons ensuite les problèmes dégagés suite à la mise en place de la coopération des deux langages et concluons sur l'introduction des transformations affines en SIGNAL.

Chapitre 2

Langages synchrones

Le langage SIGNAL a été développé dans le projet EP-ATR de l'IRISA dans le souci de disposer d'une méthodologie de conception de systèmes temps-réel qui garantirait leur sécurité et le respect des contraintes de coût et performances qui leurs sont associées. SIGNAL est basé sur le modèle de la programmation synchrone dans lequel le temps physique a été remplacé par un temps *logique* caractérisé uniquement par l'ordre relatif des instants (leur succession) et leur simultanéité. Concrètement, les hypothèses du modèle synchrone sont : 1) Les actions (communications et calculs) d'un système s'effectuent en temps logique nul; 2) Deux ou plusieurs actions peuvent avoir lieu à un même instant logique (elles sont simultanées). Il ne s'agit pas d'exclure complètement le traitement du temps physique; l'étude de son influence sur le comportement général du système est remise à plus tard, au moment de l'implémentation. Les hypothèses de synchronisme permettent d'éliminer le non déterminisme induit par le temps physique et de vérifier des contraintes de sécurité exprimées sous la forme de propriétés logiques [BB91a].

Deux classes de langages ont émergé basées sur le concept de la programmation synchrone : les langages impératifs – comme STATECHARTS, CSML, ESTEREL, ARGOS – dont le modèle sous-jacent est celui des machines à transitions d'états finis, et les langages déclaratifs ayant comme support le modèle des récurrences multi-horloges ; parmi ces derniers langages, SIGNAL et LUSTRE.

Contrairement à LUSTRE, qui est fonctionnel, SIGNAL est un langage relationnel : la programmation d'une application en SIGNAL est réalisée par la spécification d'un système d'équations qui représentent des contraintes (relations) sur les signaux (variables de SIGNAL) impliqués. Afin de vérifier que le programme est correct par rapport aux contraintes exprimées et de produire du code exécutable, le compilateur de SIGNAL doit effectuer des calculs formels ; il devient ainsi un outil de preuve. Nous citons seulement quelques uns des avantages de SIGNAL : programmation par composition de systèmes d'équations, interface graphique pour la saisie de programmes, vérification de propriétés statiques et dynamiques, implémentation séquentielle et distribuée de programme SIGNAL, profiling (évaluation de coût de programmes SIGNAL) et génération de VHDL.

Dans ce chapitre, nous proposons une présentation rapide du langage SIGNAL dans le contexte de la programmation synchrone. Après un passage en revue des langages synchrones existants, nous décrivons les opérateurs élémentaires du noyau de SIGNAL qui permettent de définir complètement le langage. Nous expliquons ensuite le principe des calculs formels effectués par le compilateur sur la *synchronisation* de signaux et les *dépendances de données*. Nous concluons ce chapitre avec une analyse des possibilités de spécification et de simulation en SIGNAL de systèmes VLSI synchrones périodiques [McC94] [KMR94] et de circuits asynchrones plus généraux; nous décrivons aussi la possibilité d'effectuer des transformations de spécification pour la validation de diverses implémentations d'un système. Nous déduirons de cette analyse la motivation principale de notre travail, i.e.

la définition et la mise en place d'un contexte de conception conjointe d'applications de traitement du signal et de l'image en utilisant les langages SIGNAL et ALPHA. Dans ce contexte, il est apparu nécessaire d'étendre le langage SIGNAL avec la possibilité d'exprimer des transformations affines sur des variables et de fournir le calcul formel support de celles-ci.

2.1 Modèle de programmation synchrone

2.1.1 Présentation générale

C'est dans le contexte de la conception de systèmes réactifs que le modèle synchrone est apparu, avec le développement des langages synchrones ESTEREL [BD91], STATECHARTS [HP85] [Har87], CSML [CLM91], LUSTRE [HCRP91] et SIGNAL [LGLL91].

Généralement, on distingue trois types de systèmes (programmes) [Ber89]. Les *programmes transformationnels* produisent un certain nombre de résultats à partir d'un ensemble d'entrées; comme exemples, les compilateurs ou les programmes de traitement numérique. Les *programmes interactifs* interagissent à leur propre rythme avec les utilisateurs ou les autres programmes ; du point de vue de l'utilisateur, un système "time-sharing" est interactif. Les *programmes réactifs* interagissent continuellement avec l'environnement, mais à une vitesse imposée par ce dernier, et non par le programme lui-même. Tandis que les programmes interactifs doivent principalement traiter des communications avec l'extérieur à leur propre vitesse, les programmes réactifs doivent répondre à des stimuli extérieurs et disposer d'un mécanisme fiable de traitement d'interruption.

Les systèmes temps-réel sont en général réactifs. Néanmoins, il existe des systèmes réactifs qui ne sont normalement pas considérés comme temps-réel ; parmi ceux-ci les protocoles, les drivers de système et les interfaces hommes-machine. L'approche synchrone apparaît comme une méthode de conception de systèmes réactifs par l'introduction du langage ESTEREL comme une alternative aux langages très généraux GPL (General Purpose Languages) dans [Ber89]. [HP85] propose en même temps la méthode *Statecharts* comme un formalisme visuel pour la description du comportement des systèmes réactifs complexes. Dans les deux cas, il apparaît clairement la nécessité d'un langage de description doté d'une sémantique simple et rigoureuse qui permettrait la vérification de propriétés de sûreté dans des systèmes (réactifs et/ou temps-réel) dans lesquels un simple défaut peut avoir des conséquences très graves.

L'idée de l'approche synchrone, telle qu'elle est présentée dans [BB91b], est d'éliminer le non déterminisme introduit par la prise en compte du temps physique dans la conception d'un système, et de se concentrer uniquement sur les aspects logiques des réactions et sur la vérification que le système considéré est correct du point de vue logique. Le modèle de programmation synchrone est basé sur un ensemble de concepts et méthodes qui possèdent des fondements mathématiques très rigoureux et élégants. L'idée est de ne considérer que des systèmes "idéalisés" qui réagissent de manière *instantanée* à leurs entrées. De tels systèmes peuvent être composés facilement et leur comportement peut être décrit et analysé de manière simple.

L'approche synchrone permet d'aborder et de résoudre certains problèmes qui apparaissent dans la conception de systèmes réactifs : la spécification et la programmation de systèmes concurrents hiérarchisés, la génération de code efficace et contrôlable, l'utilisation d'outils automatiques de vérification de propriétés. Dans la section suivante, nous proposons un bref descriptif des principaux langages synchrones impératifs et déclaratifs existants.

2.1.2 Langages synchrones

STATECHARTS [Har87] [SN97] est un langage graphique pour la spécification de systèmes réactifs basé sur le modèle des diagrammes à transitions d'états dotés de mécanismes de composition et communication. Dû essentiellement à l'existence de ces mécanismes, le formalisme STATECHARTS a été largement utilisé dans l'industrie. Un grand nombre de concepts existant dans le langage complet posent néanmoins des problèmes de syntaxe et de sémantique. Pour une description complète de ces problèmes, nous renvoyons le lecteur à [M.94].

Une sémantique du formalisme STATECHARTS a été définie pour la première fois dans [HPSS87]. Depuis, plusieurs autres approches ont été proposées dans la littérature. Le point clé de plusieurs d'entre elles a été de fixer une sémantique au niveau des communications en décidant sur l'instant d'exécution de toute communication (le *step* (pas) courant ou le suivant). Dans [SN97], une nouvelle approche est proposée en considérant que la décision appartient à l'utilisateur et en fournissant des opérateurs adéquats pour le choix de la sémantique désirée.

Une énumération détaillée des sémantiques existantes est proposée dans [M.94]. La sémantique STATEMATE définie dans [HN96] et commercialisée par i–Logix [Inc90] a constitué le point de départ pour la conception d'un environnement de programmation de systèmes réactifs englobant plusieurs formalismes – comme STATECHARTS et SIGNAL – dans le cadre du projet européen SACRES [CON97b] [CON97a]. Un des problèmes étudiés est l'interopérabilité entre SIGNAL et STATECHARTS. Des schémas de traduction de STATECHARTS en SIGNAL sont en cours d'élaboration en tenant compte des différents modes d'interprétation de la sémantique STATECHARTS (*step* et *super-step*), et pour lesquels on tire parti des horloges de SIGNAL et du calcul associé [BGL+98].

ARGOS [Mar91] est un langage synchrone de syntaxe graphique similaire à celle de STATECHARTS. Un programme peut être un automate ou le résultat de l'application d'un opérateur n–aire aux opérandes du programme. Chaque automate est représenté comme un ensemble de blocs de diagrammes interconnectés par des fils qui décrivent leurs communications. Contrairement à STATECHARTS, la représentation graphique détermine complètement la syntaxe et la sémantique de chaque opérateur et du programme entier.

Dans la famille des langages synchrones impératifs, CSML (Compositional State Machine Language) [CLM91] a été défini pour la description de contrôleurs à états finis. Le compilateur du langage génère une table d'états qui sert comme entrée pour des outils de vérification formelle qui utilisent des techniques de model-cheking. La table d'états permet aussi la génération de code pour diverses implémentations.

Le langage ESTEREL [BD91] [BG92] est un langage impératif qui manipule soit des variables classiques locales à des blocs d'instructions concurrents, soit des *signaux* utilisés pour la communication avec l'environnement ou entre des processus concurrents. Un signal est un couple formé d'une variable qui caractérise la présence ou l'absence du signal dans une réaction donnée, ainsi que d'une valeur d'un type arbitraire. Les signaux sont partagés par diffusion instantanée vers tous les processus. Les événements qui déclenchent la réaction des instructions d'un programme peuvent être composés d'occurrences simultanées de signaux. On retrouve en ESTEREL deux classes d'instructions: 1) des instructions impératives standards ;2) des instructions temporelles comme les *triggers*, les *watchdogs* et les boucles temporelles.

ESTEREL possède une sémantique comportementale et une sémantique opérationnelle; cette dernière a été utilisée pour la compilation de programmes en automates séquentiels. Le langage ESTEREL a été utilisé dans des domaines comme le contrôle des processus temps-réel, le graphique et les protocoles de communication. Les travaux les plus récents vise à rendre la validation des systèmes transparente et donc accessible aux non-spécialistes du domaine de la preuve formelle.

Finalement, LUSTRE [HCRP91] et SIGNAL [LGLL91] sont des langages déclaratifs basés sur le modèle des récurrences multi-horloges: les variables manipulées – les signaux – représentent des séquences de valeurs auxquelles sont associés leurs instants logiques de présence : *échelles de temps* ou horloges. En ce sens, “seconde” et “milliseconde” deviennent des horloges parmi d'autres, d'où l'aspect multi-horloges des langages. Un calcul formel est nécessaire pour la vérification des synchronisations exprimées sur les horloges; ce calcul représente le support pour toute vérification ultérieure de propriétés statiques ou dynamiques d'un programme. Il peut contenir des aspects différents entre les deux langages, induits par l'existence en LUSTRE d'une horloge unique contenant les instants logiques de toutes les autres horloges et par le caractère fonctionnel du langage, propriétés que l'on ne retrouve pas en SIGNAL. LUSTRE étant basé sur un point de vue strictement fonctionnel [CPHP87], il peut être compilé en automates et soumis aux outils existants de vérification formelle et génération de code.

2.2 Le langage SIGNAL

SIGNAL est un langage synchrone et flots de données qui permet une programmation aisée par spécification de *contraintes (relations)* sur les signaux impliqués. Le compilateur de SIGNAL doit effectuer des calculs formels sur la *synchronisation* des signaux et les *dépendances de données* afin de vérifier leur consistance et de produire du code exécutable.

Le caractère flots de données de SIGNAL se manifeste en ce qu'il manipule des séquences (éventuellement infinies) de données appartenant toutes à un même domaine de valeurs (leur type commun) : les *signaux*. A chaque valeur d'un signal est associé un entier positif qui dénote l'instant logique de présence de la valeur dans le signal respectif. L'ensemble des instants logiques de présence d'un signal représente l'*horloge* du signal.

Les signaux de mêmes horloges sont dits *synchrones*. Les opérateurs de SIGNAL sont définis par des contraintes (relations) entre les valeurs et les horloges des signaux impliqués. Nous employons le terme *processus* pour dénoter un tel système de relations. En SIGNAL, nous distinguons un ensemble de quatre opérateurs (processus) *primitifs* ou *élémentaires*. Ils constituent le *noyau* de SIGNAL : ils permettent de définir complètement le langage. Des constructions plus complexes peuvent être ensuite obtenues à partir des processus primitifs par composition et restriction. Elles peuvent être combinées dans des modules et utilisées dans une programmation structurée comme il sera montré dans la suite.

Parmi les premières références sur SIGNAL, nous citons [LBBG86]. Dans [LG91] SIGNAL apparaît pour la première fois dans la variante dans laquelle il existe aujourd'hui, avec un premier aperçu du modèle sémantique des traces qui lui est associé. Pour une bonne introduction au langage, avec un exemple de son utilisation – un système de reconnaissance de la parole – nous renvoyons le lecteur à [LLGL91]. Un aperçu plus complet de SIGNAL et de son environnement dans une méthodologie de programmation de systèmes temps-réel, avec un exemple de son utilisation sur la programmation d'une montre digitale, se trouve dans [GLM94].

Finalement, syntaxiquement SIGNAL a subi plusieurs transformations. Dans toute référence ultérieure, nous utiliserons la dernière version du langage – la version V4 – définie dans [GLD94].

2.2.1 Les opérateurs primitifs

Parmi les opérateurs de base de SIGNAL nous distinguons les opérateurs *monochrones* – qui manipulent des signaux synchrones – et les opérateurs *polychrones*, dont les signaux peuvent avoir des horloges différentes.

2.2.1.1 Opérateurs monochrones

Expressions fonctionnelles : x_{n+1} `:=` `g(` $x_1, ..., x_n$ `)`

L'expression SIGNAL précédente dénote l'extension aux séquences de valeurs (signaux) des fonctions définies sur les valeurs. Pour une fonction `g` agissant sur les signaux (`g` peut être une fonction arithmétique ou booléenne, une fonction définie par l'utilisateur), on note g la fonction correspondante sur les valeurs. L'expression SIGNAL x_{n+1} `:=` `g(` $x_1, ..., x_n$ `)` est équivalente à

$$x_{n+1\,t} = g(\ x_{1t}, \ldots, x_{nt}\ )$$

pour tout instant t de présence du signal x_{n+1}; les signaux $x_1, \ldots, x_{n+1}$ doivent être présents ou absents au mêmes instants de temps.

Retard : a_2 `:=` a_1 `$ init` x_0

Le processus ci-dessus introduit un retard sur les valeurs du signal a_1 : cette expression dénote un processus élémentaire satisfaisant

$$\begin{aligned} \forall\ t > 0,\ \ a_{2t} &= a_{1(t-1)} \\ t = 0,\ \ a_{2t} &= x_0 \end{aligned}$$

L'opérateur de retard contraint les signaux a_2 et a_1 à être synchrones. La première valeur de a_2 est une *constante* x_0 qui doit être associée à la spécification de l'opérateur de retard. Nous présentons sur la Figure 2.1 une suite de valeurs de a_2 retardée d'un instant par rapport à la suite de a_1 ($x_0 = 0$).

a_1 :	2	5	1	0	4	1	3	7	9	$\cdots$
a_2 :	0	2	5	1	0	4	1	3	7	$\cdots$

FIG. 2.1 – *Illustration de l'opérateur de retard.*

2.2.1.2 Opérateurs polychrones

Extraction sous condition booléenne : a_3 `:=` x_1 `when` a_2

Le processus `when` permet de construire a_3 comme un *sous-échantillonnage* de x_1 par la condition booléenne a_2. Le signal a_3 est présent quand a_2 est présent et a la valeur *true*; sinon, a_3 est absent. Une séquence de valeurs pour a_3, x_1 et a_2 est présenté sur la Figure 2.2 :

x_1 :	1	2	$\perp$	3	4	$\perp$	5	6	9	$\cdots$
a_2 :	t	f	t	f	t	f	$\perp$	f	t	$\cdots$
a_3 :	1	$\perp$	$\perp$	$\perp$	4	$\perp$	$\perp$	$\perp$	9	$\cdots$

FIG. 2.2 – *Illustration de l'opérateur* `when`.

On peut remarquer que les signaux a_3, x_1 et a_2 ont des horloges différentes.

	Syntaxe	Description
Processus primitifs		
Expressions f fonctionnelles	a := g($x_1, ..., x_n$)	g: une fonction arithmétique ou booléenne, une fonction définie par l'utilisateur, etc.
Retard	a_2 := a_1 $ init x_0	a_2 : la précédente valeur de a_1
Extraction sous c condition booléenne	a_3 := x_1 when a_2	a_3: x_1 quand a_2 est présent et vrai
Entrelacement de signaux	a_3 := a_1 default x_2	a_3: a_1 si a_1 est présent; sinon x_2
Composition de processus	P | Q	Les processus P et Q communiquent par leurs signaux communs
Restriction de processus	P/a	Le signal a n'est pas visible à l'extérieur du processus P
Opérateurs dérivés		
	c := ^x	L'horloge de x
	c := when b	L'horloge des instants à vrai de b
	x ^= y	Contrainte de synchronisation explicite : l'horloge de x est égale à l'horloge de y

TAB. 2.1 – **Le noyau de SIGNAL et opérateurs dérivés.**

Entrelacement de signaux : a_3 := a_1 `default` x_2
Le processus `default` construit a_3 comme un entrelacement *avec priorité* des valeurs des signaux a_1 et x_2. A chaque instant logique, la valeur de a_3 est égale à celle de a_1, si a_1 est présent; sinon, a_3 prend la valeur de x_2. Par la priorité accordée aux valeurs de a_1, l'opérateur `default` est déterministe. Nous présentons sur la Figure 2.3 une exécution particulière du processus `default`.

$$\begin{array}{lcccccccccc} a_1: & 1 & 2 & \perp & 3 & 4 & \perp & 5 & \perp & 9 & \cdots \\ x_2: & \perp & \perp & 3 & 4 & 10 & 8 & 9 & 2 & \perp & \cdots \\ a_3: & 1 & 2 & 3 & 3 & 4 & 8 & 5 & 2 & 9 & \cdots \end{array}$$

FIG. 2.3 – *Illustration de l'opérateur* `default`.

Pour tous les processus précédents, les signaux situés en partie gauche du signe `:=` constituent les sorties du processus et ceux situés en partie droite ses entrées. Chaque processus représente ainsi une équation (par l'utilisation de `:=`) entre ses entrées et ses sorties.

2.2.1.3 Opérateurs sur tableaux

Le langage SIGNAL possède un mécanisme d'itération sur des structures de données multidimensionnelles qui permet l'implémentation d'un répertoire d'algorithmes simples pour leur manipulation. Le constructeur de base est `array` à partir duquel on peut définir un ou plusieurs tableaux et/ou effectuer un traitement itératif, sur des signaux non nécessairement synchrones, par la replication d'un même processus SIGNAL. Cette replication peut former un réseau où les entrées d'un nœud sont les sorties des nœuds voisins immédiats.

Un exemple d'utilisation du constructeur `array` est représenté dans la Figure 2.4 où nous avons illustré le calcul de la somme des éléments du vecteur `V`. Une façon classique de procéder est de

```
array I to N of
  ACC := ACC + V
with ACC[0]:ZERO, V
        end
```

FIG. 2.4 – *Exemple d'utilisation du constructeur* `array`.

définir une boucle sur le vecteur et de sommer à chaque itération i la somme des $i-1$ premiers éléments calculée à l'itération précédente avec le i^{eme} élément. Ce mécanisme se traduit en SIGNAL par l'utilisation du constructeur `array` qui réalise la replication du processus de sommation de deux signaux : le processus i reçoit en entrée un signal du processus $i-1$ qui représente la somme des $i-1$ éléments du vecteur `V`, ainsi que le i^{eme} élément de `V`. Le processus i effectue la somme des ses deux entrées et transmet le résultat au processus $i+1$. L'exemple de la Figure 2.4 peut être lu : $ACC[i] := ACC[i-1] + V[i]$, pour tout $i > 0$; pour $i = 0$, $ACC[0] = 0$. Le tableau `ACC` joue le rôle d'accumulateur des résultats partiels ; la somme des éléments de `V` se retrouve dans $ACC[N+1]$.

L'indexation des signaux est implicite : toute sortie est indicée par l'indice courant du constructeur ; dans l'exemple de la Figure 2.4, la sortie `ACC` est implicitement indicée en $ACC[i]$. L'indexation des entrées est spécifié à travers la clause `with` ; pour un constructeur `array` d'indice `I` et de dimension `N` on peut avoir trois indexations possibles :

- `V` signifie que le signal d'entrée `V` est indicé en $V[i]$;
- `V[0] : VINIT` signifie une indexation en $V[i-1]$ et $V[0]$ est initialisé à $VINIT$;
- `V[.] : VINIT` implique une indexation en $V[i+1]$ et $V[N+1]$ est initialisé à $VINIT$.

L'implémentation actuelle des tableaux en SIGNAL est minimale car limitée, comme on peut le remarquer, du point de vue des relations qui peuvent être exprimées entre les domaines d'indices associés aux structures de données multi-dimensionnelles. Aussi, les extensions à considérer dans le futur proche devront permettre la définition et la manipulation de tableaux dont les éléments peuvent avoir des horloges différentes, ce qui n'est pas le cas actuellement.

2.2.2 Composition de processus

Les processus primitifs de SIGNAL peuvent être composés par l'opérateur de composition parallèle `|` ; la composition `P | Q` des processus `P` et `Q` consiste dans l'union des équations correspondant aux processus `P` et `Q` : les processus communiquent par leurs signaux communs, i.e. les sorties de `P` peuvent être des entrées de `Q` et vice-versa.

Un signal ne peut pas être défini dans les deux processus à la fois; le langage SIGNAL respecte le principe de l'assignation unique. Les entrées de `P | Q` sont les signaux qui ne sont définis dans aucun des processus `P` et `Q`; les sorties de `P | Q` peuvent être les sorties de `P` ou de `Q`.

La composition de processus est commutative et associative.

2.2.3 Restriction de processus

Le confinement d'un signal a à un processus `P` est réalisé par l'opérateur de masquage `/` ; la sortie a n'est plus visible à l'extérieur du processus `P /` a ainsi construit.

2.2.4 Modularité et opérateurs dérivés

Les processus primitifs de SIGNAL peuvent être combinés par composition et restriction pour produire des processus plus complexes appelés *modèles*: les modèles représentent des systèmes d'équations caractérisés par une interface décrivant leurs entrées et sorties avec les types associés. La définition de modèles en SIGNAL permet une programmation structurée par création et utilisation de bibliothèques de modèles.

Nous résumons dans le tableau 2.1 les processus primitifs de SIGNAL et les principaux opérateurs dérivés (macros SIGNAL). Nous illustrons ensuite leur utilisation, ainsi que les opérations de composition ou restriction de processus, par la programmation d'un exemple simple (modèle) dans la section suivante.

2.2.5 Exemple d'utilisation de SIGNAL

La Figure 2.5 contient un exemple de modèle construit avec des processus parmi les opérateurs élémentaires et dérivés présentés dans le tableau 2.1. En associant au système obtenu par la composition des équations :

```
S := E default (ZS - 1)
ZS := S $ init 1
E ^= when (ZS <= 1)
```

une interface décrivant ses entrées et ses sorties (une seule entrée et une seule sortie dans ce cas) on obtient un nouveau processus `OVERSAMPLE` dont l'utilisation sera

S := `OVERSAMPLE(E)`

A chaque appel, le processus `OVERSAMPLE` sera remplacé par son corps. Le terme *modèle* est utilisé pour mettre en évidence la possibilité d'une programmation modulaire entraînée par la définition de ce type de processus.

```
(1)   process OVERSAMPLE =
(2)    ( ? integer E;    % E is an input signal %
(3)      ! integer N; )  % S is an output signal %
(4)     (| S := E default (ZS - 1)
(5)      | ZS := S $ init 1
(6)      | E ^= when (ZS <= 1)  |) / ZS
(7)    where integer ZS;
(7)   end;
```

FIG. 2.5 – *Exemple de programme* SIGNAL.

Le programme SIGNAL de la Figure 2.5 implémente un *sur-échantillonnage* : pour chaque valeur de l'entrée `E`, le processus `OVERSAMPLE` produit `E` valeurs de sortie pour `S` (cf. au schéma d'exécution présenté sur la Figure 2.6). `ZS` contient la valeur précédente de `S` (définie par l'équation SIGNAL (5)); cf. à la ligne (4) du programme SIGNAL, la valeur de `S` est égale, pour chaque instant logique auquel `S` est présent, à celle de `E` si `E` est présent, ou à sa valeur précédente décrémentée de 1 (`ZS` $- 1$), si `E` est absent. A chaque instant t du programme une nouvelle valeur pour l'entrée `E` est acceptée seulement si ($ZS_t <= 1$) est vrai. La variable `ZS` est locale au processus `OVERSAMPLE` ; ses valeurs ne peuvent pas être utilisées à l'extérieur de ce processus.

Il apparaît du schéma d'exécution présenté sur la Figure 2.6 que la sortie `S` est plus fréquente que l'entrée `E`; le processus `OVERSAMPLE` décrit ainsi une relation – et non une fonction – entre son entrée et sa sortie.

	t_0	t_1	t_2	t_3	t_4	t_5	$\cdots$
`E` :	3	$\perp$	$\perp$	2	$\perp$	5	$\cdots$
`ZS` :	1	3	2	1	2	1	$\cdots$
`S` :	3	2	1	2	1	5	$\cdots$

FIG. 2.6 – *Schéma d'exécution : les t_i représentent les instants logiques du programme.*

2.2.6 Calcul d'horloges

Les opérateurs primitifs (élémentaires) de SIGNAL sont définis par des contraintes entre les valeurs et les horloges des signaux impliqués; ces contraintes s'organisent en *systèmes* pour des processus plus complexes construits à partir d'opérateurs primitifs par composition ou restriction. Afin de vérifier qu'un programme est correct du point des contraintes ainsi exprimées et de produire du code exécutable, le compilateur de SIGNAL doit effectuer des calculs formels sur la synchronisation des signaux (contraintes d'horloges) et les dépendances de données (obtenues à partir des contraintes sur les valeurs).

A chaque processus SIGNAL sont associés une équation sur les horloges des signaux impliqués et un graphe aux dépendances conditionnées par les horloges auxquelles les dépendances sont effectives [BL87] [BLL88].

En ce qui concerne les horloges, on obtient pour les opérateurs SIGNAL élémentaires les équations suivantes :

- x_{n+1} `:= g(` $x_1, ..., x_n$ `)` $\qquad h(x_1) = h(x_2) = \ldots = h(x_n) = h(x_{n+1})$
- a_2 `:=` a_1 `$ init` x_0 $\qquad h(a_2) = h(a_1)$
- a_3 `:=` x_1 `when` a_2 $\qquad h(a_3) = h(x_1) \wedge [a_2]$
- a_3 `:=` a_1 `default` x_2 $\qquad h(a_3) = h(a_1) \vee h(x_2)$

Pour tout signal x on note $h(x)$ son horloge. Les équations sur horloges associées aux processus monochrones – expressions fonctionnelles et retard – expriment le fait que les signaux impliqués doivent être synchrones. Pour le processus a_3 `:=` x_1 `when` a_2, l'horloge $h(a_3)$ de la sortie a_3 est définie comme l'intersection de l'horloge $h(x_1)$ de x_1 et l'horloge $[a_2]$ des instants à vrai de la condition booléenne a_2. L'horloge de la condition booléenne a_2 est partitionnée d'après les valeurs de a_2 ($[\neg a_2]$ représente l'horloge des instants à faux de a_2) conformément aux *équations de partition* suivantes :

$$[a_2] \vee [\neg a_2] = h(a_2)$$
$$[a_2] \wedge [\neg a_2] = \emptyset$$

Finalement, dans le cas du processus `default`, l'horloge de la variable a_3 est définie comme l'union des horloges de a_1 et x_2.

Considérant H l'ensemble des horloges d'un programme SIGNAL, $(H, \vee, \wedge)$ est un treillis booléen d'ordre $\leq$: $h_1 \leq h_2$ si l'ensemble des instants de h_2 contient l'ensemble des instants de h_1 (on notera aussi $h_1 \subseteq h_2$ puisqu'une horloge est assimilée à un ensemble d'entiers positifs). Les propriétés des horloges associées aux conditions booléennes – déduites essentiellement à partir des équations de partition – permettent de définir sur l'espace H un ordre $\preceq$ "plus grossier" que l'ordre $\leq$ du treillis booléen [Bes92]. Afin de vérifier qu'un programme SIGNAL est correct du point de vue des contraintes exprimées sur ses horloges, il est nécessaire de résoudre le système d'équations sur horloges qui lui est associé. La résolution de ce système – *le calcul d'horloges* – repose sur les propriétés de l'ordre $\preceq$. L'idée est d'englober l'ensemble des horloges de H dans un treillis complet d'ordre $\preceq$; la borne supérieure de ce treillis constituera l'horloge *maîtresse* du programme qui cadencera son exécution et à partir de laquelle toutes les autres horloges peuvent être calculées.

Le treillis $(H, \vee, \wedge)$ est représenté en une forêt d'arbres sur les sous-échantillonnages par conditions booléennes (un arbre est un treillis complet d'ordre $\preceq$) ; cette représentation est appelée *hiérarchie d'horloges*. Un programme SIGNAL à plusieurs arbres – plusieurs maxima locaux par rapport à l'ordre $\preceq$ – est appelé *exochrone*. La spécification de ses synchronisations est incomplète pour un schéma d'exécution déterministe. Dans le cas d'une horloge maîtresse unique, le programme est *endochrone* et son horloge maîtresse définit complètement les instants de son exécution.

Nous reprenons le processus `OVERSAMPLE` de la section précédente en indiquant le système d'équations sur horloges qui lui est associé par le compilateur de SIGNAL.

`process OVERSAMPLE =`	
`( ? integer E;`	<u>Équations sur horloges</u>
`! integer S; )`	
`(4) (| S := E default (ZS - 1)`	(4) $\hat{\mathtt{S}} = \hat{\mathtt{E}} \vee \hat{\mathtt{ZS}}$
`(5) | ZS := S $ init 1`	(5) $\hat{\mathtt{ZS}} = \hat{\mathtt{S}}$
`(6) | E ^= when (ZS <= 1) |) / ZS`	(6) $\hat{\mathtt{E}} = [\mathtt{ZS} \leq 1]$
`(7) where integer ZS;`	
`end;`	

FIG. 2.7 – *Le processus* `OVERSAMPLE` *et le système d'équations sur horloges associé.*

A partir des équations ci-dessus il résulte que l'horloge de la variable `S` est "plus fréquente" que celle de `E` : effectivement, comme $\hat{\mathtt{S}}$ représente l'union des horloges $\hat{\mathtt{E}}$ et $\hat{\mathtt{ZS}}$, $\hat{\mathtt{E}}$ est au plus égale à $\hat{\mathtt{S}}$: $\hat{\mathtt{E}} \preceq \hat{\mathtt{S}}$ (voir aussi le schéma d'exécution représenté sur la Figure 2.6). L'équation (6) $\hat{\mathtt{E}} = [\mathtt{ZS} \leq 1]$ raffine la condition précédente en imposant que les instants de $\hat{\mathtt{E}}$ coïncident avec les instants où `ZS` est présent et inférieur à 1.

Conformément aux équations de partition

$$[\mathtt{ZS} \leq 1] \vee [\neg(\mathtt{ZS} \leq 1)] = \hat{\mathtt{ZS}}$$
$$[\mathtt{ZS} \leq 1] \wedge [\neg(\mathtt{ZS} \leq 1)] = \emptyset$$

on a $[\mathtt{ZS} \leq 1] \preceq \hat{\mathtt{ZS}} = \hat{\mathtt{S}}$, et on retrouve $\hat{\mathtt{E}} \preceq \hat{\mathtt{S}}$, ainsi que la manière de calculer $\hat{\mathtt{E}}$ en fonction des valeurs de `ZS`. Nous avons illustré sur la Figure 2.8 l'arbre d'horloges qui représente le treillis complet $\{\hat{\mathtt{S}}, \hat{\mathtt{ZS}}, \hat{\mathtt{E}}\}$ d'ordre $\preceq$ et de borne supérieure $\hat{\mathtt{S}}$. On remarque ainsi que $\hat{\mathtt{S}}$ est le seul maximum du programme SIGNAL ; le programme est endochrone.

Nous avons esquissé de manière informelle le calcul d'horloges dans le cas de l'exemple précédent. Dans la suite, nous allons revenir sur les principes de la méthode de résolution du système d'équations sur horloges et sur son implémentation, au moment où nous décrirons l'extension de cette méthode pour la vérification de contraintes de synchronisation sur des horloges obtenues par

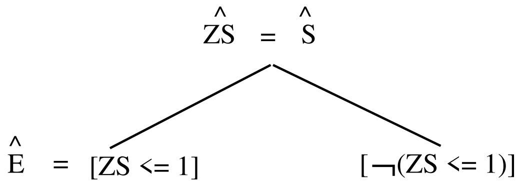

FIG. 2.8 – *L'arbre d'horloges associé au processus* `OVERSAMPLE`.

transformations affines. Nous concluons en soulignant le fait que $(H, \vee, \wedge)$ est isomorphe avec le treillis $(Exp(\mathcal{B}), +, *)$ des expressions booléennes ($\mathcal{B}$ est l'ensemble des variables booléennes).

2.3 Spécification, validation et implémentation de systèmes avec SIGNAL

Dans cette section nous présentons le contexte dans lequel s'intègre notre travail de thèse. Il s'agit de pouvoir décrire et valider en SIGNAL des architectures synchrones ou asynchrones dans le cadre d'une implémentation matérielle ou mixte matérielle-logicielle d'un système temps-réel. Nous analysons le problème posé par ce type de descriptions qui est celui des synchronisations d'horloges régulières, c'est-à-dire les horloges des entrées/sorties des composants VLSI périodiques. Dans cet esprit, nous décrivons une première expérience engendrée par la modélisation de systèmes VLSI synchrones et périodiques avec SIGNAL [KMR93] [McC94]. Les architectures modélisées étaient considérées synchrones et il s'agissait de vérifier des contraintes de synchronisation sur des horloges exprimées comme sous-échantillonnages réguliers sur l'horloge globale du système. De manière plus générale, on veut pouvoir traiter en SIGNAL les synchronisations des horloges régulières plus générales (sur-échantillonnages réguliers également). Comme on le verra dans la suite, ces horloges peuvent être décrites comme des *transformations affines* d'horloges dans un contexte de transformation de spécifications pour l'exploration des diverses implémentations possibles d'un système. Le problème de la définition et la mise en place du calcul formel nécessaire pour la prise en compte des synchronisations sur les nouvelles horloges et devenu l'objet principal de notre thèse et notre principale contribution au développement de l'environnement du langage SIGNAL.

2.3.1 Modélisation et validation d'architectures VLSI en SIGNAL

Comme présenté précédemment, le compilateur de SIGNAL réalise des calculs formels sur la synchronisation des signaux et leurs dépendances. En ce qui concerne les synchronisations, le calcul d'horloges utilise les propriétés du treillis booléen et des horloges définies comme sous-échantillonnages par conditions booléennes afin de vérifier la synchronisation des expressions (unions, intersections et différences) construites sur ces sous-échantillonnages.

La description et la validation d'architectures VLSI en SIGNAL entraîne des contraintes de synchronisation sur des horloges *régulières* – des horloges sous-échantillonnages affines (leurs instants peuvent être décrits par des entiers de la forme $\{ni + \varphi\}_{i \geq 0}$, où $n > 0$ et $\varphi \geq 0$ représentent la période et la phase de l'horloge) ou sur-échantillonnages par nombre constant d'instants – et des expressions sur ce type d'horloges.

Considérons l'exemple d'un système synchrone d'horloge globale h composé de deux circuits C_1 et C_2 et d'un additionneur (le système représenté sur la Figure 2.9). Les circuits C_1 et C_2 reçoivent

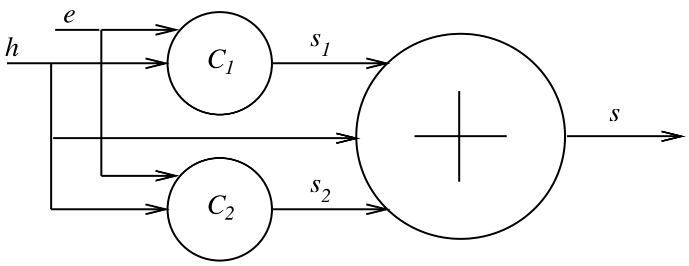

FIG. 2.9 – *Les circuits C_1, C_2 et l'additionneur sont synchrones (leur horloge est h).*

l'entrée e et produisent les sorties s_1 et s_2 de manière périodique sur h, c'est-à-dire les horloges de e, s_1 et s_2 sont des sous-échantillonnages affines sur h. Nous spécifions en SIGNAL le système représenté sur la Figure 2.9[1] ; le rôle de C_1 et C_2 est de retarder l'entrée e d'un nombre d'instants égal à n_1 et n_2.

```
process SYSTEME = { integer n1, n2; }
  ( ? integer e; event h ! integer s; )
   (| s1 := C { n1 }(e, h) | s2 := C { n2 }(e, h)
    | s := s1 + s2  |) / s1, s2
  where integer s1, s2;
        process C = { integer n; }
          ( ? integer ec; event hc; ! integer sc; )
           (| cc ^= hc
            | cc := ((n - 1) when ^ec) default (zcc - 1)
            | zcc := cc $ init 0
            | ec ^= when (zcc = 0)
            | memec := ec cell hc
            | sc := memec when (cc = 0)|)/ cc, zcc, memec
          where integer cc, zcc, memec;
        end;
end;
```

Le processus `C` modélise le comportement des circuits C_1 et C_2; l'instanciation de `C` pour chaque circuit se matérialise dans l'instanciantion du paramètre n aux valeurs spécifiques fournies par n_1 et n_2. Le fonctionnement de `C` est tel que l'entrée ec est récupérée sur les instants de l'horloge hc multiples de n. L'entrée ec est ensuite réproduite par la sortie sc après $n-1$ instants de l'horloge hc suivant l'arrivée de ec.

Les entiers n_1 et n_2 spécifient les paramètres des sous-échantillonnages réguliers : si $\{t_i\}_{i\geq 0}$ représente l'ensemble des instants de h, alors $\{t_{n_1 i}\}_{i\geq 0}$, $\{t_{n_1 i+2}\}_{i\geq 0}$ (pour le circuit C_1) et $\{t_{n_2 i}\}_{i\geq 0}$, $\{t_{n_2 i+2}\}_{i\geq 0}$ (pour le circuit C_2) représentent les descriptions des ensembles d'instants pour e et s_1,

1. `cell` est un processus SIGNAL dérivé qui a le rôle d'une mémoire ; `X : = Y cell C`, avec `C` signal booléen, se réécrit en :

```
(| X ^ = (^Y) default (when C)
 | X : = Y default (X $ init x0) |)
```

respectivement pour e et s_2, correspondants aux circuits C_1 et C_2. C_1 et C_2 fournissent s_1 et s_2 à l'additionneur ; pour cette opération, les sorties s_1 et s_2 doivent être synchronisées. La contrainte implicite de synchronisation imposée par l'équation SIGNAL `s : = s1 + s2` doit être validée par le compilateur de SIGNAL, en fonction de diverses valeurs que peuvent prendre n_1 et n_2. Par exemple, pour $n_1 = n_2 = 3$, un schéma d'exécution possible pour le processus `SYSTEME` est le suivant :

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	$\cdots$
`h` :	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	$\cdots$
`e` :	3	$\perp$	$\perp$	2	$\perp$	$\perp$	5	$\perp$	$\perp$	6	$\cdots$
$\mathtt{s}_1$:	$\perp$	$\perp$	3	$\perp$	$\perp$	2	$\perp$	$\perp$	5	$\perp$	$\cdots$
$\mathtt{s}_2$:	$\perp$	$\perp$	3	$\perp$	$\perp$	2	$\perp$	$\perp$	5	$\perp$	$\cdots$

Dans ce cas, on observe que les instants de présence des signaux s_1 et s_2 coïncident et que la contrainte de synchronisation sur leurs horloges respectives peut être validée par le compilateur (calcul d'horloges) de SIGNAL. Le calcul d'horloges doit prendre en compte les valeurs des paramètres n_1 et n_2 afin de décider de la *synchronisabilité* de s_1 et s_2 (i.e. s_1 et s_2 peuvent être synchronisés) quand leurs horloges sont des sous-échantillonnages affines comme précédemment et, plus généralement, quand leurs horloges peuvent être des expressions – unions, intersections et/ou différences – sur des sous-échantillonnages affines. Comme on le verra dans la suite, la synchronisation des expressions sur des sous-échantillonnages affines entraîne des calculs sur leurs phases et périodes qui reposent sur des techniques de calcul de nombres entiers (calcul de ppcm, pgcd, etc.) comme celles qui sont à la base du modèle VSDF [KMR93] [McC94]. Dans notre thèse on s'est proposé d'aller plus loin, vers la vérification de contraintes de synchronisation sur des sur-échantillonnages réguliers disjoints correspondant aux circuits complémentaires et asynchrones d'une architecture.

Pour illustrer cette idée, nous considérons le processus `FONCTIONNEL` ci-dessous. En SIGNAL, tout modèle précédé par l'identificateur *function* possède la propriété que ses entrées/sorties sont synchrones ; en d'autres termes, l'exécution de la *fonction* `FC` est considérée comme instantanée en SIGNAL. Les deux appels de cette fonction dans `FONCTIONNEL` génèrent deux processus s'exécutant en parallèle dans un seul instant logique de SIGNAL (car le synchronisme de l'entrée et de la sortie de `FC` entraîne le synchronisme de s_1 et s_2).

```
process FONCTIONNEL = ( ? integer e ! integer s1, s2; )
   (| s1 := FC (e) | s2 := FC (e)  |)
  where  function FC = ( ? integer ec; ! integer sc; );
  end;
end;
```

Dans l'exemple précédent, la fonction `FC` n'est pas spécifiée ; elle est considérée comme une fonction externe décrite dans un langage autre que SIGNAL (par exemple, elle peut être une fonction C).

On suppose maintenant que l'on implémente la fonction `FC` par un circuit qui prend l'entrée *ec* et produit la sortie *sc* après un certain nombre de cycles d'horloges. Nous souhaitons modéliser le comportement d'un tel circuit en SIGNAL en associant à chaque instant physique du circuit un instant logique de SIGNAL. La nouvelle modélisation pourrait être utilisée ensuite pour la validation du fonctionnement du circuit par vérification formelle ou par simulation. L'implémentation SIGNAL de ce circuit sera le processus $\mathtt{C}'$ équivalent au processus `C` précédent pour lequel l'horloge *hc* a été masquée ($\mathtt{C}'$ est équivalent à `C/hc`) et $\mathtt{IMPL}'$ représente une architecture particulière qui implémente le processus `FONCTIONNEL` :

```
process IMPL' = { integer n1, n2; }
   ( ? integer e ! integer s1, s2; )
    (| s1 := C' { n1 }(e) | s2 := C' { n2 }(e) |)
  where process C' = { integer n; }
           ( ? integer ec ! integer sc; )
            (| cc := ((n - 1) when ^ec) default (zcc - 1)
             | zcc := cc $ init 0
             | hc := ^cc
             | ec ^= when (zcc = 0)
             | memec := ec cell hc
             | sc := memec when (cc = 0)  |) / cc, zcc, memec, hc
          where integer cc, zcc, memec; event hc;
         end;
end;
```

Le système $\mathtt{IMPL}'$ a été représenté sur la Figure 2.10 par les circuits C_1' et C_2' d'horloges internes h_1 et h_2 (ces horloges correspondent aux deux instances de l'horloge $\mathtt{h_c}$ générées par les instanciations du processus $\mathtt{C}'$ correspondant à chaque appel dans $\mathtt{IMPL}'$). Ces horloges sont des sur-échantillonnages réguliers par rapport à l'horloge de e. Les sorties s_1 et s_2 sont fournies de façon périodique sur les horloges h_1 et h_2 et peuvent être ainsi décrites également par des sur-échantillonnages réguliers par rapport à e. Dans ces conditions, il est parfois nécessaire de re-synchroniser les signaux s_1 et s_2 ; cela correspond à pouvoir décrire en SIGNAL des composants hardware asynchrones en fixant les points de communication avec les autres parties du système. Ainsi, il est nécessaire d'introduire en SIGNAL un calcul d'horloges qui permette la vérification des contraintes engendrées par les nouvelles synchronisations, en tenant compte des propriétés des horloges sous-échantillonnages et/ou sur-échantillonnages par un nombre constant d'instants et des expressions sur ce type d'horloges.

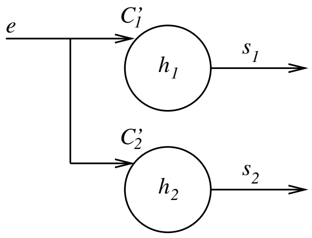

FIG. 2.10 – *Les circuits C_1' et C_2' sont asynchrones (d'horloges respectives h_1 et h_2).*

Dans l'idée de pouvoir décrire et valider des systèmes VLSI avec SIGNAL, nous présentons dans la section suivante l'expérience engendrée par la spécification et simulation d'un ensemble de composants synchrones et périodiques en utilisant le langage SIGNAL. Ensuite, le point de départ de notre travail a été la description haut niveau d'un système et l'exploration de diverses implémentations des processus composants dans un contexte de conception conjointe en utilisant les langages SIGNAL et ALPHA.

2.3.2 Modélisation et validation d'architectures Vlsi synchrones et périodiques avec Signal – un exemple

A partir du modèle des *flots de données synchrones – Synchronous Data Flow (*Sdf*)* [LM87b] [LM87a] – développé par Lee à Berkeley, Kerihuel et McConnell [KMR94] proposent un modèle abstrait étendu avec des propriétés temporelles – Vlsi Synchronous Data Flow (Vsdf) – pour la représentation de circuits Vlsi synchrones et périodiques. Les graphes Sdf représentent des graphes flots de données dans lesquels le nombre de données consommées et produites par chaque nœud du graphe est fixé pour chaque entrée et sortie du nœud. Des propriétés de *consistance* peuvent être vérifiées sur des graphes Sdf ; celles-ci garantissent ensuite l'existence d'une implémentation séquentielle ou parallèle de l'application modélisée par un graphe de ce type.

En associant à chaque nœud d'un graphe Sdf une spécification temporelle de ses communications avec les autres nœuds du système, on obtient le modèle Vsdf d'un système constitué de plusieurs circuits synchrones et périodiques. La spécification temporelle comprend la *période* du circuit en nombre de tops de l'horloge système, la repartition des entrées et sorties du circuit pendant cette période (la spécification des événements de communication), ainsi que l'intervalle de temps entre la consommation de la première entrée et la production de la première sortie, qui représente la *latence* du circuit. Cette modélisation temporelle permet de vérifier que le système Vsdf est correct vis-à-vis des contraintes de synchronisation sur les événements de communication de chaque nœud, c'est-à-dire que les instants où une valeur est consommée correspondent strictement aux instants où une valeur est disponible suite à sa production.

Un modèle Vsdf [KMR93] [McC94] pour une chaîne de codage vidéo à 34 Mbits/s [ETS90] a été spécifié et simulé en Signal. Cette application est intéressante dans le domaine du traitement des images : la compression des images est basée sur la transformation en cosinus discrète (Dct) et l'estimation de mouvement, opérations utilisées dans des standards comme Mpeg [Pur93].

Dans la spécification Signal, chaque processus décrit un nœud Vsdf dans lequel on identifie une partie comportementale, décrite en C, une partie de gestion temporelle des communications et une partie de vérification. Nous illustrons ces idées avec le processus `DCT_FLOT_Y` qui décrit la transformation en cosinus discrète appliquée au bloc de luminance de chaque composante d'une image :

```
process DCT_FLOT_Y= ( ? integer Y_IN; logical H_IN;
                      ! integer Y_OUT; logical H_OUT; )
  (| DCT_STREAM( )
   | H( 128, 64, 64, 64 )
   | ERROR:= ( false when( ^Y_IN )when READ )default( true when
            READ )default( true when( ^Y_IN ))
   | PRINT_ERR( )
   | DCT_IN:= ( Y_IN when READ )default( 0 when WRITE )
   | ZDCT_OUT:= DCT_OUT $64
   | Y_OUT:= ZDCT_OUT when WRITE
   |)/ WRITE, ERROR
  where logical READ, WRITE, ERROR;
        integer DCT_IN, DCT_OUT, ZDCT_OUT init[ { to 64 }: 0 ];
        function DCT_STREAM= ( ? integer DCT_IN ! integer DCT_OUT );
end;
```

La fonction `DCT_STREAM` décrit le comportement de la composante Dct ; c'est une fonction décrite en C. Le processus `H` associe à la composante Dct une modélisation temporelle décrite par la période $\Theta = 64$, le nombre d'entrées (64), le nombre de sorties (64) et la latence du circuit $\delta = 128$; la repartition des entrées (le flot `Y_IN`) et des sorties (le flot `Y_OUT`) sur l'horloge globale du système représentée par `H_IN` est donnée par les instants à vrai des signaux logiques `READ` et `WRITE` (sorties du processus `H`).

Les horloges des entrées et sorties des composants VSDF sont des sous-échantillonnages affines, ou des expressions sur ces sous-échantillonnages, sur l'horloge globale du système `H_IN`. L'utilisation des variables `ERROR` permet de spécifier les synchronisations sur les entrées/sorties des composants qui communiquent et d'utiliser l'outil de preuve formelle SIGALI [Dut92] [ALMR95] afin de vérifier les contraintes de synchronisation. Des considérations sur la dynamique du système permettent de vérifier l'accessibilité des états d'erreur : si un état d'erreur est accessible à partir de l'état initial du système, le système risque d'avoir des transferts de données incorrects.

Afin de pouvoir appliquer l'outil SIGALI, la bibliothèque des processus de gestion temporelle (processus `H`) a été remplacée par une version qui n'utilise que des opérations booléennes. Les conditions d'erreur ont été exprimées avec des équations booléennes. A partir de la nouvelle spécification, une vérification statique a pu être effectuée avec SIGALI pour une partie de la chaîne de codage : l'explosion du nombre de variables booléennes a posé des problèmes de temps de vérification et d'espace mémoire.

Or, la vérification de la consistance d'un graphe VSDF est basée sur un calcul de nombre entiers qui peut être effectué sans aucune considération sur la dynamique du système et qui peut être intégré dans le calcul d'horloges statique existant en SIGNAL. Plus généralement, on s'intéresse à élargir ce calcul pour la synchronisation des horloges qui représentent des sur-échantillonnages réguliers disjoints correspondants aux circuits complémentaires d'une architecture et ceci dans un contexte de conception mixte matérielle/logicielle en utilisant les langages SIGNAL et ALPHA.

2.3.3 Intégration de SIGNAL et ALPHA

Le processus `DCT_FLOT_Y` présenté dans la section précédente décrit le calcul de la DCT sur chaque pixel d'un bloc (de luminance ...) d'une image. Cette opération correspond à une décomposition particulière du calcul de la DCT du bloc d'une image en opérations élémentaires sur les pixels constituant le bloc. Cette décomposition n'est pas unique. Si on s'imagine une fonction SIGNAL *instantanée* `DCT_C` qui calcule la transformation en cosinus discrète d'une matrice correspondant au bloc de luminance d'une image, l'implémentation de cette fonction par un composant matériel pourrait se réaliser de plusieurs manières. La diversité provient des algorithmes de calculs de DCT existants, ainsi que des implémentations qui pourront être envisagées à partir de ceux-ci.

La description, la validation et l'implémentation de divers composants de calcul comme la DCT en SIGNAL doit prendre en considération les aspects suivants : la programmation d'algorithmes de calcul réguliers en SIGNAL, la synthèse de circuits à partir de spécifications algorithmiques et finalement l'interaction d'une composante algorithmique comme la DCT avec les autres éléments du système au niveau de la synchronisation des horloges.

On se situe dans un contexte dans lequel on veut effectuer des transformations rapides de spécifications afin de modéliser et simuler des implémentations variées pour des sous-systèmes composants ou pour le système dans son intégralité. Pour cela, nous sommes confrontés au problème de la spécification d'algorithmes de calcul régulier et la génération d'implémentations optimales – sur des architectures parallèles intégrées ou sur des processeurs programmables généraux – à partir de ces spécifications algorithmiques. Or, SIGNAL dispose actuellement de mécanismes réduits pour la manipulation de structures régulières [Ché91] : il n'est pas possible, par exemple, de décrire des dépendances affines générales, mais uniquement des dépendances uniformes de pas unitaire ; le code C généré n'est pas optimal, puisqu'aucune analyse n'est réalisée au niveau des dépendances de données, comme c'est le cas dans la parallélisation automatique de code [DR95] [DR96] [DRR96].

Finalement, toujours dans le contexte des traitements sur des structures régulières, aucune optimisation n'est réalisée au niveau de la synthèse de circuits intégrés : du code VHDL comportemental est généré à partir du graphe de flots synchrones qui constitue le modèle sous-jacent de tout programme SIGNAL.

Autant de problèmes qui peuvent être résolus en utilisant ALPHA, un langage fonctionnel pour la spécification et l'implémentation matérielle et logicielle d'algorithmes de traitement numérique qui peuvent être décrits par des équations de récurrence affine sur des domaines convexes (ce que l'on qualifiera de *domaines polyédraux* dans la suite). Le modèle mathématique sur lequel le langage ALPHA est basé constitue le support formel pour des transformations correctes de programmes en vue d'une implémentation matérielle ou logicielle optimisée. Les techniques qui se trouvent à la base de l'environnement du langage ALPHA sont inspirées des méthodes de parallélisation automatique de programmes [Fea92] [Fea96]. ALPHA devient ainsi un langage complémentaire à SIGNAL qui résout une partie des problèmes de spécification et de simulation existants. Il reste à résoudre le problème de la modélisation en SIGNAL des synchronisations entre les divers composants d'un système décrits en ALPHA et d'une navigation efficace entre plusieurs spécifications correspondant aux diverses implémentations possibles.

Les transformations de spécifications engendrent des *transformations affines* sur les horloges manipulées et il devient nécessaire de fournir un cadre formel pour la résolution de contraintes de synchronisation sur les nouvelles horloges obtenues par transformations affines. Nous avons introduit les transformations affines d'horloges dans le contexte du codesign avec les langages SIGNAL et ALPHA, plus particulièrement afin de décrire les transformations induites sur les horloges suite à l'interfaçage des deux langages. Dans ce cadre, la définition du calcul formel des synchronisations sur les nouvelles horloges a constitué la principale contribution de notre travail de thèse. De manière plus générale, nos résultats peuvent s'appliquer à la spécification, validation et implémentation d'applications temps-réel flots de données – comme les systèmes de traitement du signal ou de l'image – dans un cadre où SIGNAL est utilisé seul ou en conjonction avec d'autres langages flots de données.

2.4 Conclusion

Le modèle de programmation synchrone est extrêmement favorable pour la conception d'applications temps-réel sûres ; la remise à plus tard, dans des phase ultérieures de la conception, du temps physique et la manipulation d'un temps uniquement logique permet le développement d'outils de vérification qu'un système est correct du point de vue logique, c'est-à-dire qu'il respecte la spécification initiale concernant ses communications avec l'environnement (entrées/sorties). Le respect des contraintes temps-réel est assuré par la propriété d'opération à durée limitée pour toute architecture sous-jacente.

Le monde synchrone a vu l'émergence d'un grand nombre de formalismes qui, vu la difficulté du problème général, se sont spécialisés en fonction des domaines d'application ciblés. Dans ce sens, SIGNAL est un langage déclaratif basé sur le modèle des récurrences multi-horloges ; le compilateur du langage doit être un outil puissant qui effectue des calculs formels pour vérifier qu'une spécification est correcte et qui propose des implémentations optimisées pour des programmes déterministes. Le compilateur est un outil qui peut évoluer ; les méthodes formelles qui le composent peuvent être étendues et améliorées pour cibler un nombre de plus en plus grand d'applications qui peuvent être spécifiées et simulées en utilisant SIGNAL. C'est essentiellement le but que l'on se fixe par l'introduction d'opérateurs de transformation affine d'horloges. Nous avons ciblé les applications temps-réel flots de données que l'on retrouve dans le domaine du traitement du signal ou de l'image, et nous nous sommes intéressés à un contexte de conception conjointe matérielle/logicielle utilisant un ou plusieurs formalismes. Dans ce cadre, nous avons abordé le problème du calcul des synchronisations sur des horloges obtenues par transformations affines et de la synthèse de l'horloge la plus rapide d'un système, en fournissant les concepts de base pour la mise en place d'un calcul d'horloges dédié à ces tâches.

Chapitre 3

Modèles "flots de données"

Dans ce chapitre nous présentons ALPHA, un langage fonctionnel basé sur le modèle des *graphes de dépendances restreints*, qui constituent eux-mêmes un cas particulier de graphes de dépendances dans le modèle "flots de données". Contrairement à SIGNAL, dans lequel la manipulation des flots de données est liée au concept d'horloge et aux traitements des synchronisations entre horloges, dans ALPHA on manipule essentiellement les fonctions qui caractérisent les dépendances de données. Parmi celles-ci, les fonctions affines possèdent des propriétés intéressantes quand elles sont appliquées aux polyèdres. Par ses propriétés, ALPHA est un langage de spécification et implémentation de systèmes complémentaire à SIGNAL. Les deux langages peuvent être utilisés dans un contexte de conception conjointe d'applications temps-réel, comme il apparaîtra dans le chapitre suivant où nous présenterons les ingrédients nécessaires pour la coopération des deux langages.

Nous avons structuré la présentation de ce chapitre comme suit : nous proposons d'abord un survol rapide du monde des langages flots de données. Nous décrivons ensuite les principaux aspects qui concourent à la définition du langage ALPHA. Finalement, nous présentons l'environnement d'implémentation optimale de programmes développé autour de ALPHA et qui exploite le modèle mathématique sous-jacent au langage.

3.1 Contexte

Les langages (modèles) "flots de données" sont apparus comme une bonne solution pour l'exploitation du parallélisme inhérent dans les multi-processeurs, les machines vectorielles ou toute autre architecture parallèle [Ack82] [WP94]. Initialement, les efforts des concepteurs de systèmes ont été orientés vers la construction de compilateurs spécialisés dans l'optimisation de programmes décrits dans des langages conventionnels – i.e. impératifs – existants ; ainsi sont nés, par exemple, les compilateurs vectoriels pour Fortran. Or, il s'est trouvé rapidement que certains des langages existants possédaient une grande partie des propriétés indispensables pour constituer un modèle de spécification de programmes pour des systèmes flots de données. Parmi ces propriétés, Ackerman cite les plus importantes dans [Ack82] :

- Absence d'effets de bord et localité des actions;
- Équivalence entre le flot d'exécution des instructions et les dépendances de données ; ceci atteste la nécessité d'une analyse du graphe des dépendances de données pour extraire des informations utiles à l'exécution. Cette propriété est largement utilisée aujourd'hui dans les techniques d'optimisation de code pour la construction de compilateurs paralléliseurs [DPR94]

ou pour la synthèse d'architectures parallèles intégrées dans l'environnement du langage ALPHA;

- Le principe de l'assignation unique;
- Une notation et un traitement particulier des structures multi-dimensionnelles et des itérations, déterminés principalement par les contraintes de l'absence d'effets de bord et du respect du principe de l'assignation unique.

Initialement, les langages *fonctionnels* – qui modélisent les programmes comme des appels de fonctions mathématiques – partageaient certaines des propriétés énumérées ci-dessus – par exemple, le principe de l'assignation unique – avec les langages connus ultérieurement sous le nom de langages *flots de données*. Ces derniers étaient initialement basés sur le modèle des *graphes flots de données*, c'est-à-dire la modélisation d'un programme sous la forme d'un ensemble de nœuds interconnectés par un réseau d'arcs qui sont porteurs des données communiquées entre les nœuds [WP94]. Dans ce modèle il n'existe pas d'opération courante : chaque opération s'exécute quand ses entrées sont disponibles. Le langage décrit ainsi un modèle de concurrence au niveau opération.

Au cours du temps, les langages flots de données modernes se sont éloigné du modèle graphe flots de données initial, plus particulièrement en ce qui concerne la description et la manipulation des itérations ou des structures de données multi-dimensionnelles. Malgré ces écarts, les concepteurs de langages ont préservé essentiellement une sémantique qui respectait le principe de l'assignation unique.

En examinant d'un œil attentif le développement des langages flots de données, on peut remarquer une forte influence entre ces langages et les langages fonctionnels. Cela est probablement dû à leur développement parallèle, au partage de certaines propriétés fondamentales – qui contrastaient profondément avec celles des langages impératifs – ainsi qu'aux problèmes similaires concernant l'implémentation efficace de programmes dans les deux classes de langages.

Le langage LUCID [AW77] [Ski91] est un langage fonctionnel qui manipule des suites infinies de valeurs dans un nombre fini de dimensions. Plusieurs implémentations ont été proposées pour les programmes écrits en LUCID ; néanmoins, il est considéré comme un langage flots de données : les fonctions spécifiées en LUCID correspondent aux nœuds d'un graphe flots de données ; les données traitées par ces fonctions sont des suites (éventuellement infinies) de valeurs échangées par l'intermédiaire des arcs qui réalisent les connections entre les nœuds.

La motivation initiale de LUCID a été de fournir un système formel pour la programmation et la vérification de systèmes. Les programmes écrits en LUCID constituaient en fait un ensemble d'axiomes à partir desquels les preuves de propriétés étaient réalisées par raisonnement logique basé sur un ensemble de règles d'inférence et d'axiomes généraux associés aux opérateurs du langage. Initialement, LUCID manipulait des séquences infinies de valeurs dans une seule dimension [AW77] ; la nécessité de décrire des applications contenant des calculs importants sur des structures multi-dimensionnelles a déterminé l'extension de LUCID aux suites infinies de valeurs dans un nombre fini de dimensions [Ski91] [FJ93]. Un environnement de programmation et implémentation d'applications de traitement du signal a été développé autour de LUCID : la syntaxe fonctionnelle de GLU [Agi95] [JDA96] et ses notions de causalité et d'espace multi-dimensionnel représentent un bon support pour l'implémentation logicielle d'applications ; GLU dispose aussi d'un mécanisme de parallélisation de code comprenant des procédures d'ordonnancement, allocation et équilibrage de charge et traitement des communications.

Historiquement, LUCID est un des plus anciens langages flots de données ; il représente par ailleurs la quintessence des concepts que l'on souhaite intégrer dans un langage de spécification, validation et implémentation de systèmes dans un domaine très large d'applications (traitement du signal et de l'image, applications temps-réel, etc.). La complexité du problème général a induit, comme dans bien des cas, la spécialisation du problème en fonction des domaines d'applications ciblés. SIGNAL est un langage flots de données basé sur l'hypothèse du synchronisme ; Ptolemy [KL93] est un outil de description et de simulation de systèmes basé, entre autres, sur le modèle des *flots de données synchronisés* (*Synchronous Data Flow* – SDF) ; ALPHA est un langage fonctionnel de description de programmes basé sur le formalisme des *graphes de dépendances restreints*.

Le modèle de programmation flots de données a été considéré comme un paradigme intuitif et convenable pour la description de systèmes de traitement de signal, i.e. des systèmes qui contiennent des importants traitements numériques. Néanmoins, quand de tels systèmes doivent respecter des contraintes temps-réel, on ne peut pas considérer un ordonnancement dynamique. Les modèles et langages flots de données ont été adaptés afin de prendre en compte cet aspect. Dans [LM87a] [LM87b] les graphes de flots de données synchronisés (SDF) représentent des graphes flots de données particuliers dans lesquels le nombre de données produites et consommées par chaque nœud du graphe est spécifié statiquement pour chaque entrée et chaque sortie du nœud. Sur les graphes SDF on peut ainsi vérifier l'existence d'ordonnancements séquentiels ou parallèles : la *cohérence* d'un graphe SDF peut être vérifiée au moment de la compilation. De plus, dans [Lee91a] [Lee91b] le modèle SDF a été adapté à une classe plus large d'applications de traitement du signal par l'introduction de conditions booléennes dans un modèle SDF généralisé.

Le modèle SDF a été intégré dans un environnement de programmation graphique et interactif dénommé Ptolemy décrit pour la première fois dans [LHG+89] (pour plus de détails sur Ptolemy, voir [LM93]). Ptolemy permet de décrire une application dans des *domaines* différents qui correspondent chacun à un modèle de traitement ; parmi ces modèles, on retrouve les SDFs qui modélisent typiquement des applications de traitement du signal synchrones. Une application peut être décrite dans un ou plusieurs domaines, vérifiée par une analyse statique et simulée. A partir d'une spécification on peut générer une mise en œuvre logicielle (en C ou en Assembleur) ou générer une description dans un HDL (VHDL ou SILAGE). Il n'est pas possible de passer de n'importe quel domaine à toutes les mises en œuvres possibles [LM93]. On ne dispose pas d'un vrai outil de vérification formelle, mis à part l'analyse statique effectuée pour certains modèles.

Un autre exemple de graphes flots de données est constitué par les *graphes de dépendances restreints* qui sont des spécifications de systèmes en termes de graphes de dépendances acycliques et périodiques dans lesquels le comportement du système est précisé pour une période et les dépendances entre périodes spécifiées par indexation [KMW67]. Les graphes de dépendances restreints possèdent des ordonnancements optimaux qui peuvent être calculés au moment de la compilation. Néanmoins, leur domaine d'application est plus restreint par rapport aux SDFs : les graphes de dépendances restreints sont utilisés généralement pour décrire des algorithmes itératifs réguliers, qui peuvent être mis en œuvre de manière systématique sur des architectures linéaires de circuits intégrés ou processeurs programmables.

A partir de cette approche, le langage ALPHA a été développé comme un langage fonctionnel basé sur le formalisme des équations récurrentes affines [Mau89a]. ALPHA permet la description d'algorithmes réguliers, comme la multiplication de matrices ou vecteurs, et la transformation de programmes pour optimisation. Un environnement de transformations de programmes a été construit autour de ALPHA pour la synthèse d'architectures parallèles régulières [Qui84] [WS94]. Les techniques implémentées dans l'environnement de transformations de programmes de ALPHA s'apparentent aux méthodes de parallélisation automatique de code utilisées dans la conception des compilateurs vectoriels spécialisés [DPR94] [Fea96]. Ces techniques reposent sur une analyse statique et des transformations correctes de programmes modélisés par des graphes statiques de dépendances affines dont les variables sont définies sur des polyèdres convexes.

En conclusion, il n'existe pas de langage flots de données général, mais la tendance est de définir des modèles particuliers adaptés à des domaines d'applications restreints et d'étendre ensuite ces modèles pour plus de généralité. Dans la même idée, nous envisageons l'extension du langage SIGNAL dans un contexte de coopération avec ALPHA, et ceci étant données les propriétés complémentaires des deux langages.

Nous proposons d'abord un survol du formalisme ALPHA à l'appui de l'idée de complémentarité.

3.2 Le langage ALPHA

Basé sur le modèle des *graphes de dépendances restreints*, ALPHA [Mau89b] [Mau89a] [Wil94] est un langage fonctionnel pour la spécification d'algorithmes de calculs réguliers utilisant le formalisme des équations récurrentes. Un système d'équations de récurrence sur un ensemble convexe de coordonnées permet une programmation facile et intuitive d'un grand nombre d'algorithmes réguliers, ainsi que des transformations de spécifications afin de déduire une paire d'indices représentant, d'une part, l'indice de temps dans une exécution dirigée par les dépendances de données, et d'autre part l'indice du processeur sur lequel le (les) calcul(s) seront exécutés [Qui84]. Or, il a été observé qu'une grande partie des algorithmes considérés pour la synthèse de circuits systoliques pouvaient être décrits par des systèmes d'équations récurrentes. Basé sur ces observations, un environnement de synthèse de circuits à partir de spécifications de haut niveau a été développé autour de ALPHA [CFG$^+$92] [WS94]. Un certain nombre de transformations peuvent être effectuées sur la spécification ALPHA initiale afin de déduire un programme dans lequel les indices de temps et de processeur seront mis en évidence; ces transformations garantissent des programmes sémantiquement équivalents aux programmes d'origines. A partir de la description finale, il est possible de produire du code VHDL pour l'accès aux outils de synthèse [Le 97]. Il est également possible de simuler les descriptions d'algorithmes par une traduction en C [QRW95a] [QRW95b] [QR95].

Dans la suite, nous introduisons le langage ALPHA à travers le formalisme des équations récurrentes et la programmation d'un exemple simple. Nous décrivons ensuite brièvement les principes du processus de synthèse de circuits réguliers basé sur des transformations correctes de programmes, ainsi que les possibilités de simulation fonctionnelle existant dans l'environnement du langage. Nous concluons avec une proposition d'extension du modèle formel sur lequel le langage ALPHA est basé et qui va permettre d'accroître le domaine d'applicabilité du langage.

3.2.1 Définition du langage

ALPHA est basé sur le formalisme des systèmes d'équations récurrentes affines. Les idées et concepts que nous présentons dans cette section sont tirés essentiellement du rapport de Wilde [Wil94] et basés sur le travail de Mauras présenté dans [Mau89b] et [Mau89a].

3.2.1.1 Systèmes d'équations récurrentes

Dans la description de systèmes d'équations récurrentes, deux points de vue peuvent être adoptés. Le premier est purement fonctionnel : chaque identificateur est une fonction et une équation de récurrence définit la fonction située en partie gauche du signe de l'égalité en termes des fonctions situées à droite. Un identificateur peut aussi être vu comme une variable à assignation unique et chaque équation définit la variable de sa partie gauche en fonction des variables de sa partie droite.

Nous adoptons le deuxième point de vue dans la suite.

Définition 1 *(Équation de récurrence)*

Une équation de récurrence sur un domaine $D \subset \mathbb{Z}^n$ est une équation de la forme

$$f(z) = g(f_1(I_1(z)), f_2(I_2(z)), \ldots, f_k(I_k(z))) \tag{3.1}$$

avec

- $f(z)$ *est une variable indexée par* z *;*
- $z \in D$*, avec* D *le domaine (éventuellement parametré) de la variable* f*;*
- $f_1, f_2, \ldots, f_k$ *sont les variables situées en partie droite de l'équation; elles peuvent inclure la variable* f*, ainsi que des instances multiples d'une même variable ;*
- I_i *sont des fonctions d'index qui associent à un* $z \in D$ *un élément* $I_i(z) \in D_i$*, avec* $D_1, D_2, \ldots, D_k$ *des domaines (éventuellement parametrés) des variables* $f_1, f_2, \ldots, f_k$*.*
- *Et finalement* g *est une fonction de complexité* $\mathcal{O}(1)$ *qui définit la partie droite de l'équation.*

□

Une équation de récurrence peut être de la forme :

$$f(z) \quad = \quad \begin{cases} z \in D_1 & \Rightarrow \quad g_1(\ldots f_1(I_1(z)) \ldots) \\ z \in D_2 & \Rightarrow \quad g_2(\ldots f_2(I_2(z)) \ldots) \\ \ldots & \end{cases}$$

dans laquelle la variable f est définie par un nombre fini de cas disjoints constitués de sous-domaines disjoints de D: le domaine de la variable f est D tel que $D = \cup_i D_i$ et pour toute paire (i, j), $i \neq j$, $D_i \cap D_j = \emptyset$.

Un système d'équations de récurrence définit un graphe de dépendances ; chaque dépendance est définie comme suit : une variable f_i dans un point $p \in D_i$ dépend de la variable f_j dans le point $q \in D_j$ (noté $p_i \to q_j$) si $f_i(q)$ se trouve dans la partie droite d'une équation qui définit $f_i(p)$. La fermeture transitive de la relation définie précédemment représente la **relation de dépendance** notée $p_i \to q_j$.

Pour une relation de dépendance $p_i \to q_j$, la fonction I qui associe la valeur q_j à p_i est nommée **fonction de dépendance**. La relation de dépendance peut être réécrite en $p_i \to I(p_i)$.

Une équation de récurrence définie comme précédemment est **affine** (ERA) si toutes les fonctions de dépendance I_i sont affines, i.e. de la forme $I_i(z) = Az + b$, avec A une matrice constante et b un vecteur constant (éventuellement paramétrisé) de même dimension que le vecteur produit Az. Si la matrice A est l'identité, l'équation de récurrence est uniforme.

Un système d'équations de récurrence affine (SERA) est un ensemble d'équations de récurrence affine (de type 3.1) qui définissent les variables $f_1, f_2, \ldots, f_m$ sur les domaines $D_1, D_2, \ldots, D_m$ (les variables f_i représentent les parties gauches des équations). Les équations peuvent être mutuellement récursives. Les variables qui apparaissent dans le système seront désignées comme entrées, sorties ou variables locales du système. Toute variable qui n'est pas une entrée peut apparaître dans la partie gauche d'une équation une seule fois.

3.2.1.2 Alpha – un langage équationnel

On peut remarquer qu'il existe une correspondance entre l'ensemble des variables définies en partie gauche d'une équation et l'ensemble des équations, donc les termes "variable" (qui n'est pas une entrée) et "équation" peuvent être interchangés. Les équations de récurrence du système peuvent constituer une définition fonctionnelle d'un calcul ; le système lui-même a la forme d'un **programme statique**, c'est-à-dire un programme pour lequel le graphe des dépendances peut être déterminé statiquement et analysé. Pour qu'un système SERA définisse un programme statique, il suffit que les fonctions g_i soient complètement définies de manière statique et que les expressions conditionnelles sur les indices des variables qui apparaissent en partie gauche des équations soient limitées aux inégalités linéaires. Également, toutes les valeurs d'entrée du système doivent être présentes au moment de l'évaluation de la fonction qui définit le calcul.

Un SERA satisfaisant les conditions ci-dessus apparaît comme un bon formalisme pour la description de programmes statiques. Or, on a observé [Qui84] que la majorité des algorithmes considérés pour la synthèse de circuits réguliers peuvent être représentés sous la forme de SERAs et une méthode systématique de synthèse peut être déduite, basée sur une analyse du graphe des dépendances associé à un SERA, quand ce graphe peut être déduit statiquement.

Alpha a été introduit comme un langage de spécification d'algorithmes réguliers par écriture d'équations de récurrence affine dont l'ordre de définition n'est pas important; l'ordre des calculs est induit par l'ordre des dépendances des données. Tout programme Alpha définit un programme *statique* ; le graphe des dépendances qui lui est associé peut être déduit et analysé statiquement. Alpha respecte le principe de l'assignation unique – toute variable peut apparaître dans la partie gauche d'une équation une seule fois – ainsi que le principe de substitution selon lequel la variable située dans la partie gauche d'une équation est synonyme de l'expression située à droite et donc toute instance de la variable peut être remplacée par son expression de définition. La notion de variable globale n'existe pas en Alpha. L'exécution d'un système d'équations écrit en Alpha se répercute seulement sur les variables de sorties du système; il n'y a aucun effet de bord. Un système sans sorties est sans effet (il peut être assimilé à du code mort).

Alpha est aussi un langage fortement typé : chaque variable doit être déclarée avec le domaine et le type qui lui sont associés.

Finalement, étant donné le point de vue dual que l'on peut avoir pour un identificateur dans un système d'équations de récurrence – variable ou fonction – tout programme Alpha est une fonction entre ses sorties et ses entrées ; cette fonction est déclarée par le mot clé *system*, puisqu'il s'agit en fait de la déclaration d'un *système* d'équations récurrentes affines.

3.2.1.3 Déclaration de systèmes en Alpha

Un programme Alpha est constitué d'une *déclaration de système* – contenant le nom du système, une liste de déclarations des variables d'entrée du système et une liste de déclarations des variables de sortie – et d'une *définition de système* qui comprend la liste de déclarations pour les variables locales et un système d'équations qui définissent les variables de sortie, ainsi que les variables locales, en fonction des variables d'entrée.

Le système `prodVect` ci-dessous décrit le produit d'une matrice carrée `NxN` par un vecteur de dimension `N`. Nous retrouvons les notions spécifiées ci-dessus :

```
------- Déclaration de système -------

system prodVect : { N | 2 <= N }
                 ( A : { i, j | 1 <= i, j <= N } of integer ;
                   B : { i    | 1 <= i <= N } of integer )
        returns  ( C : { i    | 1 <= i <= N } of integer );

------- Définition de système  -------

var c : { i, j | 1 <= i <= N; 0 <= j <= N } of integer;
let
    c = case
          { | j = 0 } : 0.(i, j ->);
          { | j >= 1} : c.(i, j -> i, j - 1) + A * B.(i, j -> j);
        esac;
    C = c.(i -> i, N);
tel;
```

Le vecteur C résultat de la multiplication d'une matrice carrée A par un vecteur de dimension correspondante B est donné par la formule :

$$\begin{array}{ll} \text{for } i = 1, \ldots, N: & C_0 = 0 \\ & \text{for } j = 1, \ldots, N: \quad C_j = C_{j-1} + a_{ij} * b_j \end{array}$$

Le programme `prodVect` représente la spécification de cette formule, avec l'utilisation d'une variable locale `c` qui joue le rôle d'accumulateur des résultats partiels au cours du calcul du produit matrice-vecteur.

La notation $\mathtt{B.(i,j} \rightarrow \mathtt{j)}$ est équivalente à `B[j]`; la fonction $f : (i,j) \rightarrow j$ représente la fonction de dépendance entre le domaine $\{\mathtt{i,j} \mid \mathtt{1 <= i <= N;\ 0 <= j <= N}\}$ de la variable `c` et le domaine $\{\mathtt{i} \mid \mathtt{1 <= i <= N}\}$ de `B`: il s'agit d'une fonction **affine** de la forme $f(z) = Az + b$, avec $z = (i,j)$, $A = (0\ 1)$ et $b = 0$.

ALPHA est un langage fortement typé dans lequel chaque variable doit être déclarée avec un type composé de la forme

$$< \texttt{domaine} \longrightarrow \texttt{type des valeurs} >$$

Cette déclaration est en concordance avec l'équivalence entre *variable* et *fonction* dans une équation récurrente, comme il a été précisé dans la section 3.2.1.1. En effet, une variable peut être vue comme une fonction qui associe à chaque point entier d'un `domaine` une valeur dans un ensemble défini par le `type des valeurs`:

$$X : z \in < \texttt{domaine} > \longrightarrow X[z] \in < \texttt{type des valeurs} >$$

Le type composé associé à la variable X peut être vu comme le prototype de la fonction associée à X: il spécifie le domaine et le co-domaine de la fonction.

Les valeurs que peut prendre une variable peuvent être entières, réelles ou booléennes :
$< \texttt{type des valeurs} > \in \{\texttt{integer}, \texttt{boolean}, \texttt{real}\}$. Chaque variable doit avoir un `domaine` de définition et une valeur (appartenant au `type des valeurs`) associée à chaque point dans le domaine. Les variables scalaires sont définies sur le domaine $\mathbb{Z}^0$ (voir l'utilisation de la variable scalaire 0 dans le programme précédent).

Toute utilisation d'une variable doit être conforme à sa déclaration. ALPHA dispose d'un contrôle de types qui permet de détecter des incohérences entre la déclaration d'une variable et la manière dont elle est utilisée ultérieurement. Plus généralement, les `domaines` des variables en ALPHA représentent des structures particulières – *domaines polyédraux* – et une bibliothèque de fonctions d'accès et manipulation de structures de ce type [Wil93] a été définie et implémentée ; elle permet non seulement une vérification de type classique, mais teste que chaque fonction associée à une variable est correctement définie : à chaque élément du `domaine` est associé un unique élément du co-domaine `type des valeurs`.

3.2.1.4 Domaines de variables

Chaque variable en ALPHA peut être identifiée avec une fonction dont le domaine et le co-domaine sont définis par le type composé de la variable. Le domaine de la fonction est un sous-ensemble de $\mathbb{Z}^n$ nommé *domaine polyédral* : il représente l'intersection de $\mathbb{Z}^n$ avec une union finie de *polyèdres*.

Pour une étude plus détaillée des polyèdres, nous renvoyons le lecteur à [Mau89a] et [Wil93]. Nous adoptons ici la *définition implicite* d'un polyèdre comme l'espace des solutions d'un système mixte de contraintes linéaires:

$$\mathcal{P} = \{x \mid Ax = b, Cx \geq d\}$$

spécifié comme un ensemble d'équations et d'inégalités impliquant les matrices A, C et les vecteurs b, d, x.

Tous les polyèdres sont convexes. Un polyèdre contient un nombre infini de coordonnées rationnelles ; les domaines des variables du langage ALPHA constitue le treillis des sous-ensembles de coordonnées entières contenu dans un polyèdre ou une union finie de polyèdres:

Définition 2 *Un* **domaine polyédral** *de dimension n est défini comme*

$$\mathcal{D} : \{z \mid z \in \mathbb{Z}^n, z \in \mathcal{P}\} = \mathbb{Z}^n \cap \mathcal{P}$$

avec $\mathcal{P}$ une union finie de polyèdres de dimension n. □

Un programme ALPHA représente un système d'équations récurrentes affines dans lequel chaque variable est déclarée sur un domaine polyédral comme défini ci-dessus. Les variables du système `prodVect` sont définies sur les domaines polyédraux suivants:

```
{ i, j | 1 <= i, j <= N } - un carré dans l'espace de dimension 2
{ i | 1 <= i <= N } - un segment dans l'espace uni-dimensionnel
{ i, j | 1 <= i <= N; 0 <= j <= N } - un rectangle dans l'espace de
dimension 2
```

Des opérations d'union, d'intersection et de différence peuvent être définies sur les domaines polyédraux en utilisant les opérateurs `|` pour l'union, `&` pour l'intersection et `&~` pour la différence. L'ensemble des **domaines polyédraux** forme un treillis fermé par intersection, union, preimage par fonction affine (la classe de fonctions de dépendance acceptée en ALPHA) et image par fonctions affines unimodulaires. Ces propriétés représentent le support des transformations correctes de programmes définies dans l'environnement du langage ALPHA et utilisées pour la synthèse d'implémentations optimales pour des architectures VLSI ou pour des machines (séquentielles ou parallèles) plus générales.

3.2.2 Un environnement d'implémentation optimale de programmes en utilisant ALPHA

Les points forts du langage ALPHA sont liés aux propriétés des **domaines polyédraux** et à la restriction des fonctions de dépendance à la classe des **fonctions affines**. Même si d'un côté cela peut contraindre la puissance d'expression du langage, ce choix a été dicté par plusieurs raisons. La plus grande partie des algorithmes que l'on trouve dans de nombreuses applications de traitement du signal ou de l'image peuvent être décrits par des systèmes d'équations récurrentes affines dont les variables sont définies sur des domaines polyédraux. Il est connu aussi que les boucles qui apparaissent dans les programmes impératifs (de type `for,do`) et qui manipulent uniquement des fonctions d'indices affines peuvent être modélisées par un formalisme similaire aux programmes ALPHA [Fea91] [Fea96]; les techniques de parallélisation de code existantes sont basées sur les propriétés du modèle sous-jacent [Ban93]. Finalement, il se trouve que les propriétés du modèle choisi coopèrent harmonieusement pour former le support formel du système de transformations définies et mises en œuvre dans l'environnement du langage ALPHA.

Basée sur les propriétés de l'espace des domaines polyédraux, une bibliothèque de manipulation de domaines a été définie et mise en œuvre en C [Wil93]; elle est intégrée dans l'environnement du langage ALPHA. Un grand nombre d'opérations sur des domaines sont disponibles: test d'égalité de domaines, évaluation d'une fonction sur un domaine, union, intersection, etc.

La bibliothèque polyédrale constitue le support des principales opérations effectuées dans l'environnement de synthèse de ALPHA : le test d'équivalence de programmes et le changement de repère. Toute expression ALPHA possède une sémantique dénotationnelle induite par le typage des variables et qui permet de comparer deux programmes. Comme ALPHA respecte le principe de l'assignation unique et de la substitution, il est possible de réécrire tout programme ALPHA dans une **forme normale** unique et non réductible par les règles de réécritures existantes. Le test d'équivalence de deux programmes en forme normale se réduit à la vérification de l'égalité des domaines et co-domaines des variables qui se correspondent dans les deux programmes. Toute transformation effectuée dans l'environnement de ALPHA garantit l'équivalence du programme transformé avec celui d'origine ; dans ce sens, les transformations sont correctes.

Le *changement de repère* représente le support des transformations de programmes effectuées dans l'environnement de ALPHA. Les principaux pas vers une implémentation en circuits consistent en une suite de changements de repère qui ont comme résultat l'*uniformisation* d'un système d'équations et la *projection des dépendances* sur les indices de temps et espace correspondants à une implémentation donnée. Le nouveau programme peut être ensuite décrit dans un sous-ensemble de ALPHA à partir duquel on peut avoir accès aux outils de synthèse de circuits ; il est également possible de générer du code optimisé exécutable sur des machines parallèles ou séquentielles.

3.2.2.1 Transformations de programmes

Un programme ALPHA est un programme statique : le graphe de dépendances qui lui est associé peut être déterminé statiquement et analysé. Or, il s'agit essentiellement de trouver un ordonnancement optimal et une fonction d'allocation des calculs sur un ensemble fini de processeurs de telle manière que tout processeur réalise un seul calcul à instant donné. Les fonctions de temps et d'allocation calculées seront respectivement des fonctions quasi-affines et quasi-linéaires [Qui84]. La mise en évidence, dans un système ALPHA, des indices de temps et d'espace correspondants à ces fonctions revient à un *changement de repère (ou de base)* du système par une matrice T *inversible* et *unimodulaire.*

Définition 3 *Le changement de repère défini par une matrice inversible et unimodulaire T appliqué à une variable (locale) V consiste en trois règles de réécriture définies comme suit :*

- *Le domaine $\mathcal{D}$ de V est remplacé par $T(\mathcal{D})$.*
- *Toutes les occurrences de V en partie droite des équations sont remplacées par $V.T$.*
- *L'expression E constituant la partie droite de la définition de V est remplacée par $E.T^{-1}$.*

□

Tout programme Q obtenu par un changement de repère appliqué à un programme P est équivalent à P. Pour vérifier cette équivalence, il suffit de réécrire les deux programmes en leurs formes normales et de vérifier l'égalité des domaines associés aux variables qui se correspondent dans les deux programmes P et Q. La transformation d'un programme par la matrice T garantit que le nombre de points entiers de tout domaine $\mathcal{D}$ d'une variable locale ne change pas après la transformation et que la fonction avec laquelle la variable peut être identifiée produit un résultat identique à celui d'avant la transformation. Dans ce sens, le changement de repère est une transformation correcte.

3.2.2.2 Synthèse

L'environnement construit autour de Alpha est essentiellement un environnement de synthèse automatique ou semi-automatique de circuits. Il est basé sur une suite de transformations correctes de programmes [Mau89a] qui introduisent graduellement, dans une spécification de haut niveau, des informations relatives au temps d'exécution des calculs, au nombre et au positionnement des processeurs, à la transmission de l'information [Qui84] [LMQ90] [BQR+90] [WS94].

Les transformations essentielles sont des changements de repère qui visent l'uniformisation du système d'équations représenté par un programme Alpha et la projection des dépendances sur les indices de temps et de l'espace correspondant à une implémentation donnée.

L'uniformisation est l'opération qui transforme un programme Alpha en un programme équivalent dans lequel les variables (locales) sont définies sur le même domaine de $\mathbb{Z}^n$ et les dépendances sont des translations [Mau89a].

L'uniformité est une hypothèse qui permet d'obtenir des architectures systoliques. Un programme uniforme peut ensuite être ordonnancé et implémenté sur des circuits Vlsi. Concernant l'ordonnancement, on est à la recherche de fonctions de temps quasi-affines. Ces fonctions spécifient l'instant de calcul de chaque valeur du programme; trouver une telle fonction revient à résoudre un problème de programmation linéaire sur un domaine convexe.

Une fois l'ordonnancement trouvé, il est nécessaire d'appliquer au système un changement de repère afin de rendre visible l'indice de temps associé à chaque variable locale. Pour illustrer ces idées, nous reprenons le produit matrice-vecteur que nous avons considéré dans les sections précédentes et présentons ci-dessous sa variante ordonnancée :

```
system prodVect : { N | 2 <= N }
                  ( A : { i, j | 1 <= i, j <= N } of integer ;
                    B : { i    | 1 <= i <= N } of integer )
        returns   ( C : { i    | 1 <= i <= N } of integer );
```

```
var c : { t, j | 0 <= t <= N; 1 <= j <= N } of integer;
let
    c[t, j] = case
                { | t = 0 } : 0[];
                { | t >= 1} : c[t - 1, j] + A[j, t] * B[t];
              esac;
    C[i] = c[N, i];
tel;
```

On remarque que pour chaque élément du vecteur résultat c il existe des dépendances temporelles **uniquement** entre son instance présente et son instance précédente et ces dépendances sont unitaires. Les éléments du vecteur résultat peuvent ainsi être calculés en parallèle sur N processeurs; le deuxième indice de la variable locale c devient l'indice du processeur :

```
system prodVect : { N | 2 <= N }
                  ( A : { i, j | 1 <= i, j <= N } of integer ;
                    B : { i    | 1 <= i <= N } of integer )
        returns   ( C : { i    | 1 <= i <= N } of integer );
var c : { t, p | 0 <= t <= N; 1 <= p <= N } of integer;
let
    c[t, p] = case
                { | t = 0 } : 0[];
                { | t >= 1} : c[t - 1, p] + A[p, t] * B[t];
              esac;
    C[p] = c[N, p];
tel;
```

De manière générale, il s'agit de déterminer une fonction d'allocation qui répartit les calculs sur un ensemble fini de processeurs en attribuant à chaque processeur un seul calcul pour un instant donné. Pour obtenir des implémentations avec de bonnes propriétés géométriques, les fonctions d'allocation considérées en ALPHA sont quasi-linéaires ; cela revient finalement à projeter le domaine d'une variable sur une direction choisie de manière convenable.

Un programme dans lequel les indices de temps et de l'espace ont été mis en évidence est ensuite transformé en une version ALPHA0 qui permet une description très proche de l'implémentation physique (registres, multiplexeurs, unités de calcul arithmétiques et éléments de la logique combinatoire). A partir de cette description il est possible de produire du code VHDL pour l'accès aux outils de synthèse [Le 97].

3.2.2.3 Génération de code

Dans l'environnement du langage ALPHA, il est possible *transformer* un programme pour une mise en œuvre logicielle optimale. Dans ce sens, des travaux ont été effectués sur la compilation et la génération de code à partir d'un programme ALPHA. Un premier code exécutable – code naïf – a été généré en utilisant une technique de cache standard rencontrée dans l'implémentation efficace de programmes récursifs [QRW95b]. Le code naïf possède deux inconvénients majeurs : l'overhead induit par le changement de contexte généré par une évaluation à la demande et la gestion inefficace de la mémoire. En conséquence, la compilation doit être vue comme un processus de *transformation* d'un programme pour rendre le code exécutable plus efficace. Les techniques de transformation de programmes utilisées sont inspirées de celles de la synthèse de circuits. Il s'agit principalement de:

- Déterminer un ordonnancement statique et éventuellement optimal des calculs. Ce problème est une généralisation du problème de l'ordonnancement dans les architectures systoliques [Raj86] [QV89] à des ordonnancements multi-dimensionnels [Fea92].
- Détecter le parallélisme inhérent au système par : allocation virtuelle [BDRR93], partitionnement et allocation physique [DR93]. Pour le choix du meilleur partitionnement, une analyse statique des communications peut être effectuée [Raj93].

Les techniques d'optimization du code généré à partir d'un programme Alpha sont présentées dans [QRW95a] [QR95] ; elles utilisent en principal des algorithmes de programmation linéaire sur des polyèdres convexes. Ces techniques aboutissent à des implémentations efficaces d'algorithmes coûteux en temps d'exécution et en nombre de ressources utilisées, pour lesquels l'analyse et la mise en œuvre s'avère un problème complexe.

3.2.2.4 Extension de Alpha

Les propriétés des domaines polyédraux sont fort importantes pour l'environnement de transformations de programmes existant: 1) Alpha est fermé par transformations *unimodulaires*; 2) les programmes Alpha peuvent être mis de manière automatique sous une *forme normale* syntaxique.

Dans certaines situations il devient nécessaire de décrire des variables sur des domaines d'indices qui ne sont plus "denses" (convexes), mais possèdent "des trous d'une manière régulière". Ceci permet de décrire en Alpha des circuits périodiques ou plus généralement à plusieurs horloges ou d'effectuer des transformations non unimodulaires de programmes.

Dans ce contexte, le langage Alpha a été étendu pour la description d'une classe d'équations récurrentes dont les variables sont définies sur un ensemble plus large de domaines. Il s'agit d'unions finies de Z-polyèdres définis comme l'intersection entre un polyèdre et un treillis [QRR96b]. Les propriétés du langage qui doivent être préservées dans toute extension sont la substitution, la normalisation et le changement de repère. Celles-ci dépendent directement des propriétés de fermeture par certaines opérations des domaines sur lesquels les variables sont définies. Il a été démontré que les Z-polyèdres possèdent les propriétés de fermeture nécessaires afin de garder les fondements sémantiques du langage dans une extension contenant des Z-polyèdres.

3.3 Conclusion

Alpha est un langage fonctionnel qui utilise le formalisme des équations récurrentes affines sur des domaines polyédraux pour la spécification d'algorithmes de traitement numérique coûteux et complexes, comme les calculs effectués sur des structures de données multi-dimensionnelles. Les propriétés mathématiques du langage ont permis de mettre en œuvre un environnement de transformations correctes de programmes pour la synthèse d'architectures Vlsi spécialisées ou pour une implémentation logicielle sur des machines parallèles ou séquentielles. Le langage Alpha représente un formalisme complémentaire à Signal en ce qu'il dispose d'un modèle simple et efficace de manipulation de structures multi-dimensionnelles.

Nous avons envisagé en conséquence de construire un contexte de conception conjointe d'applications de traitement du signal ou de l'image sûres en utilisant les environnements existants autour des langages Signal et Alpha.

Chapitre 4

Conception conjointe en utilisant les langages Signal et Alpha

Le terme codesign est utilisé pour dénoter la conception conjointe matériel/logiciel vue comme un processus complexe qui intègre essentiellement les trois étapes suivantes : 1) la spécification d'une application à plusieurs niveaux d'abstraction en vue de l'exploration des différentes alternatives d'implémentation existantes suivant des critères de coût et de performance ; 2) la validation des spécifications obtenues par simulation ou vérification formelle ; 3) la synthèse d'une implémentation hétérogène contenant des parties matérielles spécifiques, constituées d'un ou de plusieurs circuits intégrés spécialisés, et des parties logicielles s'exécutant sur des processeurs programmables (microprocesseurs ou processeurs de traitement du signal). La mise en place d'une chaîne de conception conjointe nécessite la résolution d'un grand nombre de problèmes comme : le partitionnement d'une application en parties matérielles et logicielles, la spécification et la modélisation de systèmes mixtes matériel/logiciel, l'implémentation des parties matérielles et logicielles, la génération de l'interface entre les deux parties.

Dans ce chapitre, nous proposons un contexte de conception conjointe basé sur les langages Signal et Alpha. Dans ce sens, nous développons les idées et concepts exposés initialement dans [GLQ+97]. Nous partons de l'idée qu'une application de traitement du signal ou de l'image suffisamment complexe est composée de parties de calculs numériques importantes reliées par du contrôle et peut être ainsi décrite en utilisant de manière complémentaire les langages Signal et Alpha. On dispose ainsi de deux formalismes puissants qui permettent la validation des parties constituantes d'un système par simulation ou vérification formelle. Signal et Alpha disposent aussi d'un ensemble d'outils qui permettent la synthèse d'implémentations matérielles ou logicielles à partir de descriptions de haut niveau. Il reste à étudier le problème de l'interfaçage entre les deux formalismes qui se traduit dans une re-synchronisation des diverses parties du système qui peuvent subir des transformations de spécification conformément aux divers niveaux d'abstraction souhaités.

Le chapitre est structuré comme suit : en section 4.1 nous introduisons le lecteur à la problématique du codesign et présentons notre approche de codesign en utilisant Signal et Alpha comme une solution à certains problèmes qui apparaissent dans ce contexte. En section 4.2 nous décrivons brièvement notre méthodologie de codesign, en insistant sur la cospécification et la cosimulation d'une application en utilisant les langages Signal et Alpha. Dans ce contexte, le principal problème à résoudre était celui de l'interfaçage entre les deux langages à plusieurs niveaux de description. Nous avons défini deux niveaux et avons étudié les problèmes liés à leur mise en œuvre. Nous concluons ce chapitre en section 4.3 avec un récapitulatif des problèmes que nous avons abordés et résolus dans le contexte de codesign proposé – parmi lesquels la définition des transformations affines d'horloges en SIGNAL et du calcul formel des synchronisations sur les horloges transformées – et une description des aspects qui restent à être considérés dans le futur.

4.1 Introduction à la conception conjointe (codesign)

La nécessité de produire simultanément des descriptions de type logiciel et des composants matériels a donné lieu récemment à des recherches sur ce qu'on appelle la *conception mixte matériel/logiciel* (*hardware/software codesign*). Il s'agit d'explorer les différentes alternatives de conception d'un système mixte matériel et logiciel en vue d'optimiser des critères de coût et/ou de performances. La modélisation des systèmes comprenant du matériel et du logiciel à différents niveaux d'abstraction est appelée *cospécification*. La validation des conceptions soit par simulation, soit par vérification formelle du matériel et du logiciel à différents niveaux d'abstraction est appelée *cosimulation* (dans certains contextes, le terme *vérification formelle* est également utilisé). Enfin, le partitionnement de l'application en deux parties, matérielle et logicielle, l'exploration et l'évaluation de différentes possibilités, la génération de circuits correspondant au matériel, la génération du code correspondant au logiciel, ainsi que la production de l'interface entre les deux sont désignés par le terme de *cosynthèse*.

4.1.1 État de l'art

Les concepteurs de systèmes ont été confrontés aux problèmes qui apparaissent dans le codesign depuis toujours ; les progrès obtenus récemment dans certains domaines ont déterminé un renouvellement de l'intérêt dans le sujet. Parmi ceux-ci, De Micheli cite les plus importants dans [De 94] : 1) les performances accrues offertes par les systèmes informatiques d'aujourd'hui par le biais des architectures intégrées spécialisées ou des composants reprogrammables ; 2) le développement des outils de synthèse et simulation de circuits qui constitue un premier pas vers l'intégration d'environnements de conception conjointe matériel/logiciel de systèmes assistée par ordinateur. Dans le même article, l'auteur propose une incursion dans les approches utilisées au fil du temps pour la conception de coprocesseurs et/ou de composants spécialisés pour diverses applications. La plus grande partie des problèmes rencontrés relèvent en effet du domaine du codesign.

Après avoir défini les concepts de cosimulation et cosynthèse, De Micheli conclue [De 94] sur la nécessité de définir des modèles d'abstraction pour des systèmes mixtes matériel/logiciel et leurs environnements, avec des langages consistents pour leur description. Il s'agit de pouvoir disposer d'un cadre unifié dans lequel la description d'un système sera équivalente à une spécification formelle et les méthodes primitives de simulation seront remplacées par des techniques de vérification formelle étendues au domaine de la conception mixte. Des améliorations sont souhaitables aussi dans le domaine de la cosynthèse, concernant autant la synthèse automatique d'architectures intégrées spécialisées ou la génération de code optimal pour des architectures parallèles ou séquentielles générales, que le partitionnement technologique d'une application et l'assemblage des divers composants matériels ou logiciels existants.

Dans cet ordre d'idées, plusieurs tentatives de structuration en vue de l'obtention d'une méthodologie ont été proposées. Le point de départ est l'utilisation d'un langage qui permet d'intégrer dans un seul cadre des descriptions matérielles et/ou logicielles. On peut utiliser soit un langage de description de matériel enrichi avec des possibilités de décrire du logiciel – le langage VHDL avec des possibilités d'appel à des fonctions C [SJA93] – soit un langage de programmation haut niveau (comme C, C++, Fortran) incluant des concepts pour décrire du matériel. Cette dernière idée apparaît dans [EHB93] avec l'utilisation du langage C augmenté de contraintes de temps et des concepts de tâche et d'intercommunication entre tâches – on obtient ainsi le langage C^x – et dans [GCD94] [GD93] avec le langage HardwareC, un sous-ensemble de C introduit pour la description de circuits et qui contient le type entier comme unique type de donnée.

Les langages de description de matériel comme VHDL ou ceux pour une programmation haut niveau structurée de systèmes – C, C++, Fortran – peuvent être exécutés et donc permettent une validation du système par simulation. Or il est connu qu'une simulation ne peut pas garantir l'absence d'erreurs de conception. La spécification d'un système mixte requiert l'usage de formalismes consistents qui doivent permettre d'étendre les techniques de vérification formelle existantes au domaine de la conception mixte.

Toute description, que ce soit en C ou en VHDL étendu, repose sur un modèle qui est généralement un graphe représentant les fonctionnalités du système décrit. Ce modèle sert pour l'évaluation des contraintes de coût et/ou de performance du système et pour le partitionnement technologique en matériel et logiciel. Or, l'usage des langages permettant des descriptions mixtes matérielles et logicielles comme décrit précédemment permet d'accrocher à une description des informations relatives à une implémentation physique donnée (temps d'exécution, ressources utilisées, communications), mais ne fournit pas le contexte nécessaire pour appliquer des transformations correctes de programmes correspondant aux vues différentes que l'on peut avoir d'un même objet au cours du processus de conception.

Concernant les autres aspects du codesign on remarque que, de manière générale, le partitionnement d'une application en parties matérielles et logicielles est réalisé manuellement. Des techniques de partitionnement automatique ont été proposées dans [EHB93] et [GD93] [GD92] pour les systèmes Cosyma et Vulcan. Dans le premier système, un algorithme de partitionnement qui minimise une fonction coût par l'heuristique du recuit simulé permet une navigation entre des niveaux plus ou moins fins de partitionnement ; le dernier système utilise un partitionnement automatique à un niveau très fin (opération).

Une chaîne de conception conjointe de systèmes relève d'autres aspects, particulièrement dans le domaine de la cosynthèse. Parmi ceux-ci, la génération de l'interface entre matériel et logiciel est déjà un problème complexe quand il s'agit d'une architecture contenant un seul processeur qui communique avec plusieurs circuits intégrés et une mémoire [GD93] [GCD94]. D'autre part, les langages de description de matériel, dont VHDL constitue le standard, sont beaucoup utilisés dans la spécification de sous-systèmes voués à une implémentation matérielle et permettent la construction d'une plate-forme de cosimulation du système complet [TAS93], mais il n'est généralement pas possible de synthétiser de manière automatique des circuits à partir d'une description VHDL arbitraire. Enfin, en ce qui concerne la génération de code, le problème le plus important est celui de l'ordonnancement des boucles traité dans [CWB94].

En conclusion, il n'existe pas de méthode universellement acceptée pour spécifier et mettre en œuvre un système complet en fournissant une solution à tous les problèmes mentionnés ci-dessus. Les tendances actuelles évoluent vers la mise en place d'un cadre unique, qui permet la spécification, la validation et l'implémentation d'un système par une suite de transformations qui fournissent des vues différentes d'un même objet. Un exemple est le système Ptolemy [KL93] qui offre une méthodologie de codesign basée sur la possibilité de représenter des systèmes hétérogènes à divers niveaux d'abstraction et un passage facile entre des domaines qui permettent la simulation ou la génération de code. En revanche, Ptolemy n'inclut pas des méthodes de vérification formelle.

A partir de ces constatations, notre approche du codesign offre une solution à l'absence de formalismes dans les méthodes de cospécification et cosimulation (covalidation) existantes, ainsi qu'aux problèmes de synthèse automatique d'implémentations matérielles et logicielles pour les algorithmes de calcul réguliers.

4.1.2 Notre approche du codesign

SIGNAL et ALPHA fournissent deux environnements de programmation complémentaires pour la spécification et l'implémentation de systèmes mixtes. Le compilateur du langage SIGNAL associe à chaque programme un modèle sous la forme d'un graphe dynamique qui reflète la sémantique synchrone et flots de données du langage et permet la vérification de propriétés statiques – de synchronisation temporelle ou de dépendances de données – ou dynamiques, ainsi que diverses optimisations pour la génération de code C ou VHDL pour des implémentations matérielles ou logicielles. ALPHA est un langage de spécification d'algorithmes réguliers complémentaire à SIGNAL dont le modèle sous-jacent des équations de récurrence affine sur des domaines polyédraux permet la vérification et l'implémentation optimale de programmes en conformité avec une syntaxe et une sémantique pré-définies.

SIGNAL et ALPHA peuvent être utilisés conjointement pour la spécification d'une application composée de parties de calculs numériques importantes reliées par du contrôle : les parties numériques sont décrites en ALPHA et le contrôle en SIGNAL. Il est nécessaire de fournir une interface de communication entre les deux langages afin de pouvoir disposer d'une plate-forme de cospécification et de cosimulation du système complet [GLQ+97]. En ce qui concerne la cosynthèse, nous n'avons pas abordé le problème du partitionnement automatique en parties matérielles et logicielles, ni celui de la génération de l'interface entre les deux. Ces aspects induisent des problèmes complexes et peuvent constituer en eux seuls des sujets de recherche intéressants. Dans une première instance, on peut considérer un partitionnement manuel induit par les spécifications : les sous-systèmes décrits en ALPHA seront implémentés en matériel et les parties spécifiées en SIGNAL en logiciel. Ce partitionnement est intéressant puisqu'il permet essentiellement d'utiliser l'environnement de synthèse de circuits existant autour de ALPHA.

4.2 Une méthodologie de codesign

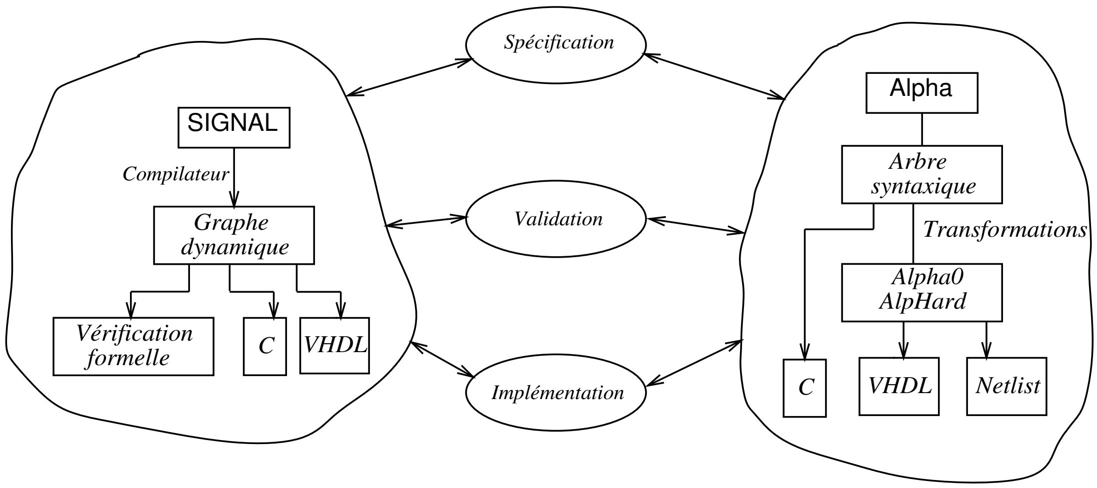

FIG. 4.1 – *Schéma de conception conjointe avec* SIGNAL *et* ALPHA.

Nous avons schématisé sur la Figure 4.1 une approche générique de conception conjointe d'applications en utilisant les langages SIGNAL et ALPHA. Nous supposons une application contenant des parties importantes de calcul numérique reliées par du contrôle; ceci est le cas de certains systèmes embarqués dans le domaine du traitement du signal ou de l'image (par exemple, un dispositif de compression/décompression de séquences vidéo). La spécification de l'application sera réalisée en utilisant un mélange des deux langages : ALPHA pour les traitements numériques sur des structures multi-dimensionnelles et SIGNAL pour le contrôle qui relie ces parties. Afin de valider la spécification hétérogène obtenue, il est nécessaire de définir une interface de communication entre les deux langages. Pour rendre ce processus plus simple qu'il ne l'est dans le cas général, nous avons commencé par considérer les calculs décrits avec ALPHA comme instantanés en SIGNAL. Ceci a permis "d'interfacer" SIGNAL et ALPHA par l'intermédiaire du langage C de la manière suivante : 1) à chaque système spécifié en ALPHA peut être associer une fonction C qui décrit le comportement du système ; 2) l'ensemble des fonctions C ainsi obtenues est inséré dans la spécification SIGNAL sous la forme de fonctions instantanées précédées par le mot clé *function*. La spécification hétérogène SIGNAL-ALPHA ainsi obtenue peut être raffinée progressivement en fonction du comportement temporel de certains sous-systèmes. On veut pouvoir, par exemple, décrire le comportement physique d'un circuit qui implémente une description algorithmique de haut niveau et qui a été obtenu par des transformations successives appliquées à certaines spécifications ALPHA. Ces transformations induisent des transformations similaires au niveau de la spécification SIGNAL et nécessitent un traitement particulier afin de permettre la validation, avec SIGNAL, de la nouvelle spécification pour l'application entière. Les transformations de spécifications ci-dessus entraînent d'une part, des *transformations affines* sur les horloges du programme SIGNAL d'origine et la nécessité de valider les synchronisations des horloges transformées, et d'autre part le besoin de mettre en place une nouvelle interface entre les langages SIGNAL et ALPHA. Cette interface a nécessité la définition de mécanismes de communication plus complexes entre la spécification SIGNAL et la fonction C décrivant le comportement du système ALPHA transformé.

Interfacer SIGNAL et ALPHA comme décrit précédemment permet de construire un contexte de cospécification et cosimulation de systèmes mixtes contrôle/calculs numériques. Les langages SIGNAL et ALPHA ne sont pas seulement des langages de spécification haut niveau, ils disposent tous les deux de générateurs de code efficace adaptés respectivement aux algorithmes contenant du contrôle ou des traitements numériques. Finalement, le graphe de dépendances obtenu comme la concatenation des graphes dynamique (SIGNAL) et celui des dépendances affines (ALPHA) peut être partitionné et évalué conformément aux contraintes de coût et performances initiales. Une implémentation mixte matérielle/logicielle peut être obtenue avec les outils de synthèse automatique de programmes fournis par ALPHA ou les mécanismes de génération de code VHDL développés en SIGNAL, ainsi que les générateurs de code C spécialisés existants dans les deux langages.

Dans les sections suivantes, nous détaillons un peu plus la présentation précédente en reprenant principalement les idées exposées dans [GLQ+97] et en insistant particulièrement sur les transformations affines d'horloges dont l'étude en SIGNAL a fait l'objet de notre travail de thèse. En section 4.2.1 nous illustrons à l'aide d'un exemple simple les travaux concernant la cospécification et la cosimulation d'une application par coopération des langages SIGNAL et ALPHA aux niveaux fonctionnel et architectural. En section 4.2.2 nous exposons uniquement les principes de la cosynthèse avec SIGNAL et ALPHA et les problèmes que cela implique.

4.2.1 Cospécification & cosimulation avec SIGNAL et ALPHA

Les principaux points forts de la méthodologie de codesign que l'on propose sont la cospécification et la cosimulation d'une application souvent complexe par la juxtaposition de deux langages qui réalisent un cadre formel complet pour des transformations et validations de spécifications à plusieurs niveaux d'abstraction. Ce contexte de cospécification et cosimulation a été rendu possible par la définition d'une interface de communication entre des spécifications SIGNAL et ALPHA. Nous avons défini deux niveaux d'interfaçage – fonctionnel et architectural – que nous détaillons dans la suite. Comme les deux langages peuvent être simulés en C, nous avons utilisé le langage C comme support de communication pour les deux niveaux d'interfaçage que nous avons introduis. L'interfaçage des deux langages a été facilité également par les mécanismes d'appels de processus externes existants en SIGNAL, plus particulièrement l'appel de *fonctions externes* (correspondant au mot clé *function*) : les processus dont les entrées et les sorties sont synchrones.

4.2.1.1 Cospécification et cosimulation au niveau fonctionnel

A ce niveau d'abstraction, la spécification $\mathcal{S}$ d'une application contient les spécifications $\mathcal{S}_1$ et $\mathcal{S}_2$, en SIGNAL et respectivement ALPHA, qui communiquent par l'intermédiaire du langage C de la manière suivante : une description fonctionnelle en C, notée $\mathcal{F}_2$, est associée à $\mathcal{S}_2$ et représentée dans $\mathcal{S}_1$ sous la forme d'un appel de fonction externe. Considérons l'exemple d'une application décrite entièrement en SIGNAL, à l'exception d'un calcul du produit entre une matrice et un vecteur que nous avons choisi de spécifier avec ALPHA. La spécification $\mathcal{S}$ consiste dans l'union des spécifications $\mathcal{S}_1$ en SIGNAL et $\mathcal{S}_2$ en ALPHA que nous avons illustré respectivement dans les Figures 4.2 et 4.3. Nous avons considéré un cas très simple afin de dégager les principaux aspects, en occurrence la spécification $\mathcal{S}_1$ contient uniquement l'appel de la fonction externe `prod` qui effectue le produit de la matrice `A` par le vecteur `B`. Le produit matrice-vecteur est un exemple de calcul numérique dont l'exécution peut s'avérer coûteuse en temps et ressources mémoire utilisées. Il peut ainsi être intéressant de spécifier ce calcul en ALPHA et de synthétiser une implémentation matérielle spécialisée ou logicielle sur des machines parallèles, ceci en utilisant l'environnement de synthèse du langage ALPHA. L'interfaçage entre $\mathcal{S}_1$ et $\mathcal{S}_2$ est réalisé de la manière suivante : le système `PROD` décrit en ALPHA (la spécification $\mathcal{S}_2$ sur la Figure 4.3) est traduit en C ; la fonction `prod` ainsi obtenue (la description fonctionnelle $\mathcal{F}_2$) est introduite dans la spécification $\mathcal{S}_1$ en SIGNAL (Figure 4.2) sous la forme d'un appel à une fonction externe. Les calcul effectués dans `prod` sont ainsi considérés comme instantanés en SIGNAL.

```
process MAT_VECT =  { integer N }
   ( ? [N,N] integer A; [N] integer B ! [N] integer C )
   (| C := prod (A, B)   % appel de fonction externe %
    |)
  where  fonction prod = ( ? [N,N] integer x; [N] integer y
                           ! [N] integer z )
end
```

FIG. 4.2 – *L'appel dans la spécification* SIGNAL *d'une fonction externe* `prod` *obtenue par génération de code C à partir de la spécification* `PROD` *décrite en* ALPHA.

On dispose ainsi d'un premier niveau d'interfa cage entre les langages SIGNAL et ALPHA qui permet de tirer parti des propriétés spécifiques à chaque langage pour la cospécification et la cosimulation de l'application décrite par $\mathcal{S}$ qui représente l'union de $\mathcal{S}_1$ et $\mathcal{S}_2$. Considérons d'abord la spécification $\mathcal{S}_2$ dans le contexte de l'environnement de programmation du langage ALPHA. Une première vérification du programme peut être réalisée basée sur le caractère fortement typé du

```
system PROD : { N | 2 <= N}
       ( a : { i, j | 1 <= i, j <= N } of integer;
         b : { i    | 1 <= i <= N } of integer )
  returns  ( c : { i    | 1 <= i <= N } of integer );
var C : { i, j | 1 <= i <= N; 0 <= j <= N } of integer;
let C[i, j] = case
               { | j = 0 }: 0[];
               { | j >= 1}: C[i, j-1] + a[i, j] * b[j];
              esac;
    c[i] = C[i, N];
tel;
```

FIG. 4.3 – *Programme* ALPHA *calculant le produit d'une matrice par un vecteur.*

langage ALPHA. Chaque variable ALPHA est en fait une fonction définie sur un domaine polyédral et prenant ses valeurs dans un ensemble restreint de types de base. A part la vérification de types classique, on vérifie que la fonction avec laquelle une variable peut être identifiée est correctement définie. L'étape de vérification de la spécification $\mathcal{S}_2$ est suivie par une étape de génération de la description fonctionnelle $\mathcal{F}_2$ qui permet la simulation fonctionnelle de $\mathcal{S}_2$.

La communication entre les spécifications $\mathcal{S}_1$ et $\mathcal{S}_2$ dans $\mathcal{S}$ se réalise par l'appel de la fonction externe $\mathcal{F}_2$ dans $\mathcal{S}_1$. A partir de ce moment, il est possible d'appliquer à $\mathcal{S}_1$ les outils de spécification et validation de programmes fournis par SIGNAL. D'abord, le compilateur de SIGNAL vérifie que les contraintes de synchronisation sur horloges ne sont pas contradictoires et qu'il n'y a pas de cycles dans le graphe aux dépendances conditionnées. Le graphe de flots synchrones obtenu après la compilation représente le support pour toute manipulation ultérieure du programme ; à partir de ce graphe, il est en effet possible de produire du code dans un format intermédiaire qui constituera ensuite l'entrée pour les outils de vérification de propriétés dynamiques. Il est également possible de générer du code C pour la simulation fonctionnelle de l'application entière spécifiée par $\mathcal{S}_1$.

4.2.1.2 Cospécification et cosimulation au niveau architectural

Nous souhaitons maintenant transformer la spécification $\mathcal{S}_2$ décrite en ALPHA afin d'explorer plusieurs implémentations envisageables. Cela est possible en utilisant l'environnement de transformations de programmes de ALPHA. Celui-ci propose principalement un ou plusieurs ordonnancements (fonctions de temps) sur les calculs en concordance avec les dépendances de données. La description ALPHA peut être transformée afin d'introduire l'indice de temps correspondant à une des fonctions de temps calculées auparavant. En conséquence, la description $\mathcal{S}_2'$ transformée n'est plus une fonction instantanée du point de vue de SIGNAL et il devient nécessaire de transformer la spécification $\mathcal{S}_1$, et ainsi $\mathcal{S}$, afin de prendre en compte le comportement temporel du programme ALPHA.

La spécification $\mathcal{S}'$ transformée est constituée des spécifications $\mathcal{S}_1'$ et $\mathcal{S}_2'$, obtenues à partir de $\mathcal{S}_1$ et respectivement $\mathcal{S}_2$, et qui communiquent par l'intermédiaire du langage C. A partir de la spécification $\mathcal{S}_2'$ obtenue dans l'environnement de ALPHA, il est possible de générer du code C ordonnancé et de simuler $\mathcal{S}_2'$; on note $\mathcal{F}_2'$ la nouvelle fonction C. La fonction $\mathcal{F}_2'$ décrit le fonctionnement de $\mathcal{S}_2'$ *pour tous les instants* de temps correspondant à l'ordonnancement calculé dans l'environnement de

ALPHA.

La spécification $\mathcal{S}_1$, décrite en SIGNAL, doit être transformée afin de rendre visible les instants d'exécution de la fonction $\mathcal{F}'_2$ et de décrire les nouvelles communications entre $\mathcal{S}'_2$ et $\mathcal{S}_1$. La nouvelle spécification $\mathcal{S}'_1$ est un programme SIGNAL englobant, d'un côté, une fonction C notée $\mathcal{F}''_2$ qui décrit le comportement fonctionnel de $\mathcal{S}'_2$ pour *chaque instant* t de temps correspondant à l'ordonnancement précédemment calculé pour $\mathcal{S}'_2$, et de l'autre côté, la gestion des communications entre $\mathcal{S}'_1$ et $\mathcal{F}''_2$. La description des communications entre $\mathcal{S}'_1$ et $\mathcal{F}''_2$ pour chaque instant de temps t correspond à l'interfaçage entre $\mathcal{S}'_1$ et $\mathcal{S}'_2$ pour toute la durée d'exécution de $\mathcal{S}'_2$ (i.e. tous les instants t correspondant au raffinement temporel de $\mathcal{S}_2$ afin d'obtenir $\mathcal{S}'_2$).

```
system PROD_ORD :{N|N>=0}
        (a : {i,j | 1<=i<=N; 1<=j<=N} of integer;
        b : {i | 1<=i<=N} of integer)
  returns  (c : {i | 1<=i<=N} of integer);
var    A : {t,p | p-1<=t<=p+N-2; 1<=p<=N} of integer;
       B : {t,p | p-1<=t<=p+N-2; p=1} of integer;
       C : {t,p | t=p+N-2; 1<=p<=N} of integer;
let    use moduleProd [0,N] (A,B) returns (C);
       A[t,p] = a[p,t-p+2];
       B[t,p] = b[t-p+2];
       c[i] = C[i+N-2,i];
tel;
```

FIG. 4.4 – *Spécification spatio-temporelle du produit matrice-vecteur en* ALPHA.

Sur les Figures 4.4, 4.5 et 4.6 nous avons illustré les principales transformations ci-dessus sur l'exemple du produit matrice-vecteur. La spécification $\mathcal{S}'_2$ du produit matrice-vecteur en ALPHA est présentée sur la Figure 4.4 par le nouveau système PROD_ORD dans lequel on a mis en évidence les informations spatio-temporelles (les indices du temps et des processeurs) correspondant à une implémentation particulière sur un circuit intégré spécialisé. L'indice de temps introduit dans PROD_ORD est illustré sur la Figure 4.6 par les micro-tops de l'horloge T' introduits entre chaque instant de T. La Figure 4.6 contient le fonctionnement d'un circuit qui pourrait réaliser les calculs décrits par PROD_ORD et aussi une description visuelle de l'interface de communication entre les spécifications $\mathcal{S}'_2$, décrite en ALPHA, et $\mathcal{S}'_1$ décrite en SIGNAL. La spécification $\mathcal{S}'_1$ est représentée sur la Figure 4.5 par le processus PROD_TR; la fonction TRANSFER décrit les calculs effectués dans $\mathcal{S}'_2$ à chaque instant t de l'horloge T' et correspond à la fonction externe $\mathcal{F}''_2$ obtenue à partir de $\mathcal{S}'_2$.

Les instants t (Figure 4.5) de l'horloge T' (Figure 4.6) apparaissent comme un sur-échantillonnage régulier sur les instants de l'horloge T de l'arrivée des entrées a et b (voir Figure 4.6). Sur la nouvelle horloge T', tous les éléments de la sortie c sont récupérés aussi de manière régulière. Autrement dit, l'horloge de sur-échantillonnage T' et celle de la sortie c peuvent être décrite comme des *transformations affines* sur l'horloge des entrées a et b.

Le processus SIGNAL PROD_TR représenté sur la Figure 4.5 est un exemple d'une spécification SIGNAL raffinée afin de prendre en considération les détails d'une implémentation particulière. Dans le processus de raffinement, l'horloge de la sortie c a subie une transformation affine par rapport à sa valeur dans la spécification $\mathcal{S}_1$ initiale: les horloges de a, b et c étaient synchrones dans $\mathcal{S}_1$; dans la spécification $\mathcal{S}'_1$ transformée l'horloge de c peut être décrite comme le résultat d'une transformation

```
process PROD_TR = { integer N }
  ( ? [N,N] integer a; [N] integer b ! [N] integer c )
   (| t:=(0 when ^a) default (zt+1)  % mise en place de l'horloge T' %
    | zt:= t$ init (2 * N - 1) | a ^= b ^= zt>= (2 * N - 2)}
    | mema := a cell ^t | memb := b cell ^t
    | array p to N of
      taba := mema[p,N-p+2] when ((1 <= t-p+2) and (t-p+2 <= N)) default 0
      end
    | array p to 1 of
      tabb :=  memb[t-p+2] when ((1<= t-p+2) and (t-p+2 <= N)) default 0
      end
    | TRANSFER( ) ? x:taba, y:tabb ! z:tabc  % communications %
    | taba ^= tabb ^= tabc | c := memc when ^a
    | zmemc := memc$1
    | array i to N of
      memc := tabc[i] when (i = (t-N)+2) default zmemc[i]
      end
    |)
    where  [N,N] integer mema; [N] integer memb, memc,
           zmemc init  [{i in 1 to N } : 0 ], taba, tabc;
           [1] integer tabb; integer t, zt init (2*N-1)
           function TRANSFER = ( ? [N] integer x; [1] integer y
                                 ! [N  integer z )
end
```

FIG. 4.5 – *Exemple d'intégration des spécifications* SIGNAL *et* ALPHA *au niveau architectural.*

affine sur l'horloge des entrées a et b.

Dans un programme SIGNAL suffisament complexe on peut imaginer plusieurs sous-processus $\mathcal{S}_1, \mathcal{T}_1, \mathcal{U}_1$, ... subir des transformations comme celle décrite précédemment pour $\mathcal{S}_1$. Le programme $\mathcal{P}'$ obtenu par raffinement contiendra les nouvelles spécifications $\mathcal{S}_1'$, $\mathcal{T}_1'$, $\mathcal{U}_1'$, ..., ainsi que la description de leur nouvelles communications. Ce dernier aspect peut entraîner la re-synchronisation des horloges obtenues suite à des transformations affines de paramètres différents dans $\mathcal{S}_1'$, $\mathcal{T}_1'$, $\mathcal{U}_1'$, ... respectivement. La validation dans $\mathcal{P}'$ des nouvelles synchronisations nécessite un calcul formel spécialisé qui, rajouté au calcul d'horloges existant dans le compilateur du langage, doit réposer sur les propriétés des transformations affines d'horloges et des *relations affines* entre horloges engendrées par ces transformations. Si on suppose les entrées a, b et c, d de $\mathcal{S}_1'$, respectivement $\mathcal{T}_1'$, dans $\mathcal{P}'$ telles que $h(a) = h(c) = H$ et $h(b) = H_1$, $h(d) = H_2$, la Figure 4.7 décrit les transformations affine subies par H_1 et H_2 suite à un raffinement particulier. Comme on verra dans le chapitre suivant, les paramètres des transformations affines permettent dans ce cas la synchronisation des horloges H_1 et H_2 (intuitivement, on peut remarquer que le positionnement relatif de H_1 et H_2 par rapport à H est le même, ce qui représente l'hypothèse de synchronisation).

La définition du calcul formel pour la re-synchronisation des horloges obtenues par transformations affines a constitué l'objet principal de nos recherches et notre principale contribution au développement de l'environnement de programmation du langage SIGNAL. Notre principale pré-

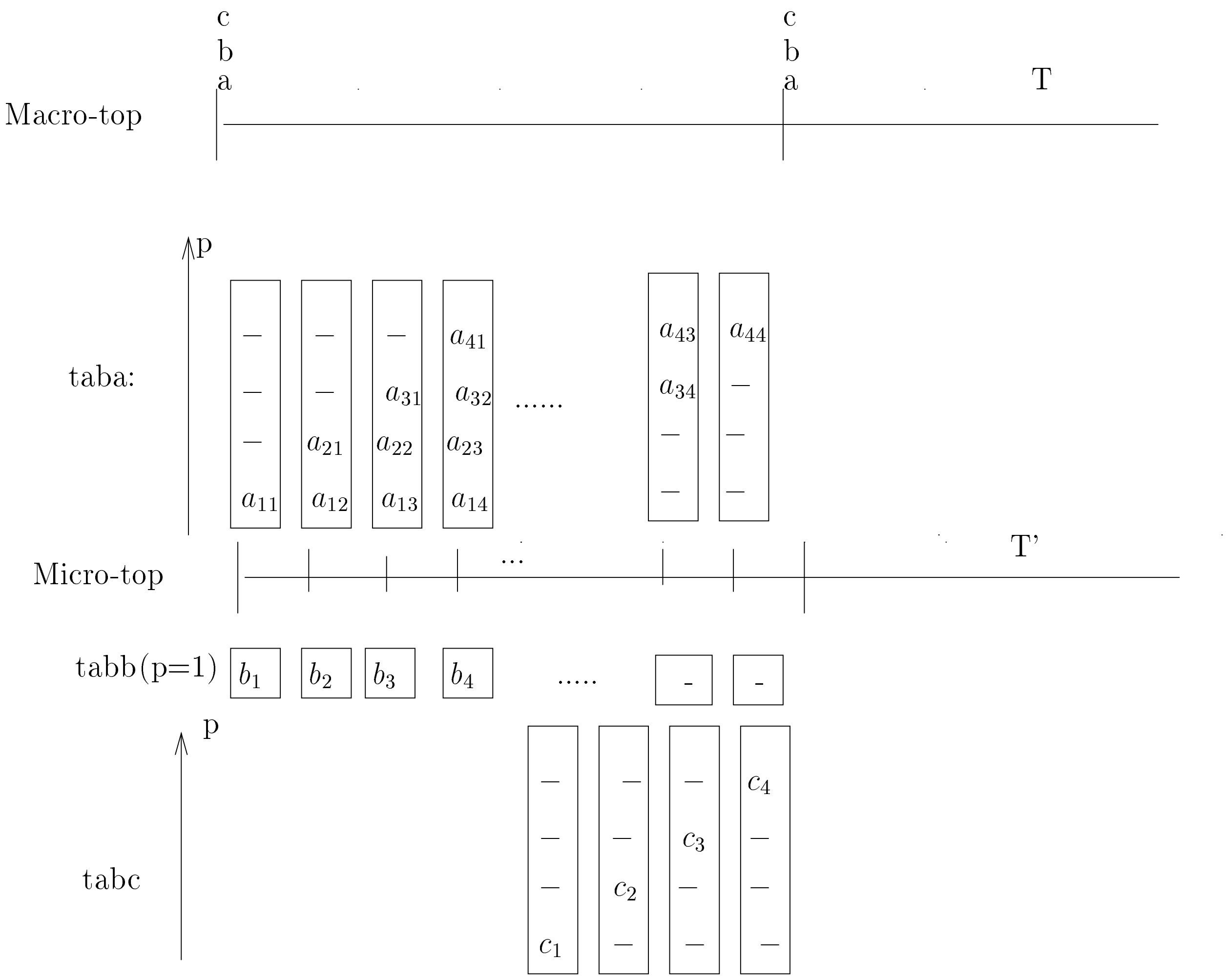

Fig. 4.6 – *Visualisation de l'interfaçage* Signal-Alpha *au niveau architectural.*

occupation a été de fournir des conditions préliminaires pour rendre possible la validation de la nouvelle spécification $\mathcal{P}'$ à l'aide de Signal (par vérification formelle en utilisant le compilateur de Signal et/ou par simulation fonctionnelle en C).

4.2.2 Cosynthèse avec Signal et Alpha

Les problèmes à résoudre dans la cosynthèse sont le partitionnement matériel/logiciel, l'évaluation du respect des contraintes de coût et de performances sur la partition obtenue, la synthèse des circuits et la génération de code C, ainsi que l'implémentation de l'interface entre les parties matérielles et logicielles. Le graphe de flots synchrones de l'application en étude peut être soumis aux outils de partitionnement et d'évaluation du temps d'exécution disponibles en Signal [KL96]. En ce qui concerne la synthèse de circuits ou la génération de code, nous utilisons au choix les outils de Signal ou de Alpha: s'il s'agit d'implémenter en circuits des traitements numériques sur des struc-

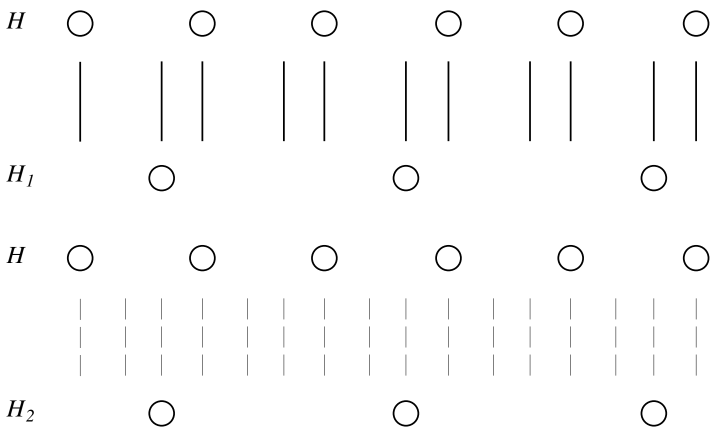

FIG. 4.7 – *H_1 et H_2 sont obtenues par transformations affines de paramètres* $(2, 1, 4)$ *et* $(3, 2, 6)$ *appliquées à H.*

tures multi-dimensionnelles, c'est-à-dire des spécifications ALPHA, on utilise les outils spécialisés de ALPHA ; autrement, pour l'implémentation matérielle du contrôle d'une application, on fait appel à SIGNAL. Dans les deux cas on peut obtenir une description VHDL, modulaire pour les descriptions ALPHA, pour l'accès aux outils de synthèse standard. Pour les parties à implémenter en logiciel, nous disposons d'implémentations efficaces en C pour les traitements numériques (génération de code C dirigé par les dépendances de données dans l'environnement de ALPHA) et d'implémentations séquentielles ou distribuées pour la partie contrôle (générateur de code exécutable par un parcours, dirigé également par les dépendances de données, du graphe de flots synchrones en SIGNAL).

Nous n'avons pas traité les problèmes du partitionnement automatique ou de l'interfaçage des parties matérielles et logicielles. Ceux-ci sont des problèmes complexes qui ont fait et continuent à faire en eux seuls l'objet de beaucoup d'investigations. En première instance, on peut considérer un partitionnement matériel/logiciel correspondant aux spécifications : les traitements numériques, coûteux en temps de calcul, seront implémentés en matériel, tandis que la partie contrôle peut être réalisée en logiciel. En ce qui concerne l'interfaçage matériel/logiciel, il n'existe pas de méthode universellement acceptée, des solutions cas par cas ont été proposées. Une méthode générique simple a été proposée dans [KW96] : la solution consiste à utiliser un contrôleur de bus qui gère les communications matériel/logiciel.

4.3 Conclusion

Nous avons étudié les éléments de base qui permettent la mise en place d'une méthodologie de conception conjointe matériel/logiciel en utilisant les langages SIGNAL et ALPHA. Nous sommes partis de la constatation que les deux langages possèdent des propriétés complémentaires qui peuvent être utilisées dans la spécification, validation et implémentation matérielle/logicielle d'une application contenant de parties importantes de traitements numériques reliées par du contrôle, comme c'est le cas dans la majorité des applications de traitement du signal ou des images embarquées. Le problème le plus important à étudier était celui de l'interfaçage des structures irrégulières décrites en SIGNAL avec celles régulières spécifiées en ALPHA, et cela à plusieurs niveaux d'abstraction en fonction des raffinements des spécifications induites par l'exploration de diverses architectures logicielles ou matérielles. L'étude de l'interfaçage SIGNAL-ALPHA a fait apparaître en SIGNAL le problème du calcul formel des synchronisations sur des horloges régulières, i.e obtenues par *transfor-*

mations affines. Intégrer un tel calcul dans l'environnement actuel du langage SIGNAL permettrait d'accroître la puissance des preuves formelles sur lequel ce dernier repose et d'élargir le domaine d'applicabilité du langage.

La définition de ce calcul a constitué l'objet principal de notre travail que nous décrivons dans la deuxième partie de notre document de thèse. Nous avons mis en œuvre un prototype de notre calcul formel qui nous a permis de valider les principes de cospécification et cosimulation d'une application de traitement d'images vidéo au niveau architectural. Le contexte de codesign avec SIGNAL et ALPHA que nous avons décrit peut être ensuite étendu avec des mécanismes de partitionnement automatique et de génération d'interfaces entre parties matérielles et logicielles, problèmes que nous n'avons pas traités dans le présent document.

Deuxième partie

Transformations affines d'horloges en Signal

Chapitre 5

Introduction

Les transformations affines de variables induisent des relations affines entre les variables opérandes de la transformation. Il est dans ce cas intéressant d'envisager une étude des propriétés des relations affines pour pouvoir construire un cadre formel dans lequel on peut vérifier que certaines contraintes de synchronisation sur les horloges transformées ne sont pas contradictoires dans le contexte d'un programme SIGNAL. Or, il s'agit essentiellement de contraintes concernant le positionnement relatif des horloges, positionnement qui dépend directement de la forme des relations qui peuvent être définies entre horloges. Nous introduisons ainsi de manière informelle les concepts de *transformation affine* effectuée sur une variable et de *relation affine* – induite par la dite transformation – entre les variables opérandes de la transformation affine.

5.1 Transformation affine d'horloges

Pour deux horloges physiques H_1 et H_2, on considère que H_2 est le résultat d'une transformation affine de paramètres (n, φ, d) appliquée à H_1, si $H_1 \times H_2$ est le domaine des valeurs de la fonction f suivante :

$$\begin{array}{l} f : \mathbb{N} \times \mathbb{N} \ \rightarrow \ H_1 \times H_2 \\ f(t,t') = (t, \frac{dt'+\varphi}{n}) \end{array}$$

H_1 et H_2 sont assimilées à des sous-ensembles de $\mathbb{N}$, respectivement $\mathbb{Q}$, qui permettent de décrire leurs instants équidistants sur une échelle arbitraire du temps physique (secondes, minutes, heures, etc.).

L'application f permet de définir les ensembles d'instants des horloges H_1 et H_2 respectivement par les points (entiers ou rationnels) marqués par des cercles et des carrés dans la Figure 5.1 ($n = 3$, $\varphi = 1$ et $d = 2$). Autrement, on peut construire H_2 en introduisant $n - 1$ instants entre chaque paire d'instants consécutifs de H_1, et en comptant ensuite tous les d instants, à partir de l'instant φ.

On observe que f peut s'écrire

$$f(t,t') = (f_1(t), f_2(t')), \ \forall \ (t,t') \in \mathbb{N} \times \mathbb{N}$$

avec

$$\begin{array}{l} f_1 : \mathbb{N} \ \rightarrow \ \mathbb{N}, \ f_1(t) = t, \ \forall \ t \in \mathbb{N} \\ f_2 : \mathbb{N} \ \rightarrow \ \mathbb{Q}, \ f_2(t) = \frac{dt+\varphi}{n}, \ \forall \ t \in \mathbb{N} \end{array}$$

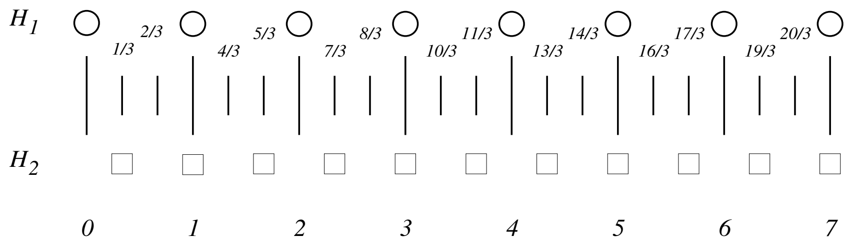

FIG. 5.1 – *Transformation affine de paramètres* $(3, 1, 2)$ *sur* H_1.

et que les instants de l'horloge H_2 peuvent être assimilés aux éléments du domaine de valeurs (co-domaine) de la fonction *affine* f_2 appliquée aux instants de H_1, d'où la dénomination de transformation affine appliquée à H_1 pour induire H_2.

En multipliant les valeurs de f par n – autrement dit, en effectuant une translation de n instants sur chaque instant de H_1 et H_2 – on obtient la fonction

$$\begin{array}{l} g : \mathbb{N} \times \mathbb{N} \ \rightarrow \ H_1 \times H_2 \\ g(t, t') = (nt, dt' + \varphi) \end{array}$$

Les instants de H_1 et H_2 peuvent être assimilés aux instants illustrés par les cercles, respectivement les carrés de la Figure 5.2 (nous avons illustré uniquement les valeurs entières correspondant aux instants des horloges H_1 et H_2).

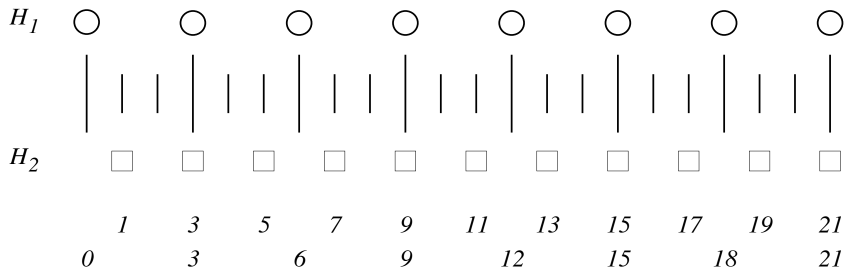

FIG. 5.2 – *Transformation affine de paramètres* $(3, 1, 2)$ *sur* H_1.

Généralement, les paramètres d'une transformation affine doivent respecter les conditions suivantes : $n, d \in \mathbb{N}^*$ et $\varphi \in \mathbb{Z}$. Pour respecter les conditions d'un temps physique qui est toujours positif, pour φ négatif il suffit d'effectuer une translation de valeur $max(0, \varphi)$:

$$\begin{array}{l} g : \mathbb{N} \times \mathbb{N} \ \rightarrow \ H_1 \times H_2 \\ g(t, t') = (nt + max(0, \varphi), dt' + \varphi + max(0, \varphi)) = (nt + max(0, \varphi), dt' + max(0, -\varphi)) \end{array}$$

Nous adoptons la même définition pour une transformation affine de paramètres (n, φ, d) appliquée à une horloge H_1 en SIGNAL. La différence fondamentale par rapport aux horloges physiques consiste dans le fait que H_1 et H_2 ne sont plus forcément des ensembles d'instants *équidistants*. Dans ce cas, une possible représentation graphique des horloges H_1 et H_2 peut être celle de la

Figure 5.3 où nous avons défini la fonction g comme suit :

$$\begin{aligned} g : \mathbb{N} \times \mathbb{N} &\rightarrow H_1 \times H_2 \\ g(t, t') &= (h(nt), h(dt' + \varphi)), \;\; \varphi \geq 0 \end{aligned}$$

avec $h : \mathbb{N} \rightarrow \mathbb{N}$, croissante sur $\mathbb{N}$ et strictement croissante sur $(n\mathbb{N}) \cup (d\mathbb{N} + \varphi)$, satisfaisant plus particulièrement pour le cas antérieur :

$$\begin{aligned} h(n\mathbb{N}) &= \{0, 5, 9, 14, 19, \ldots\} \\ h(d\mathbb{N} + \varphi) &= \{2, 5, 8, 10, 14, 17, 20, \ldots\} \end{aligned}$$

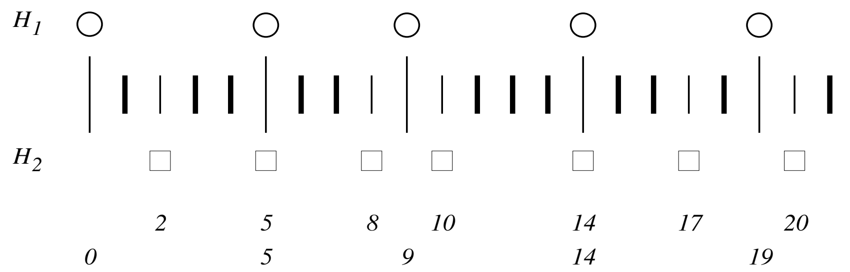

FIG. 5.3 – *Transformation affine de paramètres* $(3, 1, 2)$ *sur* H_1 *en* SIGNAL.

(nous avons représenté en gros un ensemble d'instants arbitraires complémentaire à l'union des instants de H_1 et H_2).

En conclusion, pour SIGNAL, construire l'horloge H_2 comme une transformation affine de paramètres (n, φ, d), avec $\varphi \geq 0$, sur l'horloge H_1 revient à définir une fonction g

$$\begin{aligned} g : \mathbb{N} \times \mathbb{N} &\rightarrow \mathbb{N} \times \mathbb{N} \\ g(t, t') &= (h(nt), h(dt' + \varphi)), \;\; \varphi \geq 0 \end{aligned}$$

dont le domaine des valeurs représente exactement les instants de H_1 et H_2. L'application h satisfait : $h : \mathbb{N} \rightarrow \mathbb{N}$, h *au moins* croissante sur $\mathbb{N}$ et strictement croissante sur $(n\mathbb{N}) \cup (d\mathbb{N} + \varphi)$.

Dans le cas de horloges physiques, la nécessité d'équidistance implique h strictement croissante sur $\mathbb{N}$. En SIGNAL, par la manipulation d'instants logiques, des fonctions h *croissantes* sur $\mathbb{N}$ peuvent être acceptées.

Généralement, la définition d'une transformation affine en SIGNAL ne pose pas de problème particulier pour $\varphi < 0$. H_2 est une horloge obtenue par transformation affine de paramètres (n, φ, d) sur H_1 si il existe $h : \mathbb{Z} \rightarrow \mathbb{N}$, h croissante sur $\mathbb{Z}$ et strictement croissante sur $(n\mathbb{N}) \cup (d\mathbb{N} + \varphi)$, telle que

$$\begin{aligned} g : \mathbb{N} \times \mathbb{N} &\rightarrow H_1 \times H_2 \\ g(t, t') &= (h(nt), h(dt' + \varphi)) \end{aligned}$$

La manipulation d'instants logiques en SIGNAL fait que nous nous intéressons à la *relation affine* de paramètres (n, φ, d) entre H_1 et H_2 induite par la transformation affine qui a produit H_2.

5.2 Relation affine

Dans la section précédente, nous avons introduit le concept de transformation affine de paramètres entiers positifs (n, φ, d) à l'aide d'une fonction $h : \mathbb{N} \to \mathbb{N}$ supposée croissante sur $\mathbb{N}$ et strictement croissante sur $(n\mathbb{N}) \cup (d\mathbb{N} + \varphi)$. Si on considère h strictement croissante sur $\mathbb{N}$, on obtient la définition suivante : une transformation affine de paramètres entiers positifs (n, φ, d) appliquée à l'horloge H_1 va produire une horloge H_2 en insérant $n-1$ instants fictifs entre deux instants successifs de H_1, puis en prenant chaque d^{eme} instant à partir de l'instant φ (nous avons représenté sur la Figure 5.4 un exemple de transformation affine comme appui à cette définition).

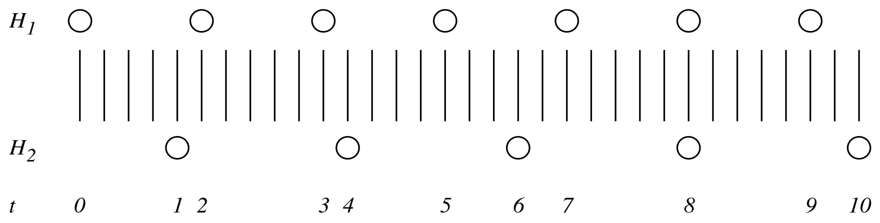

FIG. 5.4 – *L'horloge* H_2 *est le résultat de la transformation affine de paramètres* $(5, 4, 7)$ *appliquée à l'horloge* H_1 .

Sur l'exemple de la figure 5.4 une horloge *plus fréquente* que H_1 et H_2 a été exhibée ; elle constitue l'horloge *support* de la transformation affine et nous permet d'introduire le concept de *relation affine* entre les horloges H_1 et H_2 . Toute transformation affine de paramètres (n, φ, d) induit une relation affine de paramètres identiques entre les horloges opérandes de la transformation affine. La relation affine entre H_1 et H_2 décrit en fait la position relative des instants de H_1 par rapport aux instants de H_2 . La même position relative peut être obtenue par une transformation affine de paramètres différents de $(5, 4, 7)$ et en utilisant une horloge support différente. Deux relations affines qui définissent la même position relative des instants de H_1 et H_2 seront considérées comme *équivalentes*. Nous allons revenir sur ces concepts dans la section du modèle des traces en SIGNAL dédiée aux relations affines. Pour le moment, nous insistons sur le fait que :

- Une relation affine de paramètres (n, φ, d) entre deux signaux définit une position relative *régulière* entre les horloges des signaux et a été obtenue suite à une transformation affine de paramètres (n, φ, d) ;
- La même position relative peut être obtenue par une transformation affine de paramètres différents (n', φ', d') qui induira respectivement la relation affine caractérisée par le triplet de paramètres (n', φ', d') .

  Sur la Figure 5.5 nous avons représenté les horloges H_1 et H_2 obtenues par les transformations affines de paramètres $(2, 1, 4)$ et $(3, 2, 6)$ appliquées à l'horloge H. Le positionnement des instants de H_1 par rapport aux instants de H est le même que le positionnement des instants de H_2 par rapport à la même horloge H. Aucune contrainte n'existe sur la position relative des instants de H_1 et H_2 (conformément à la vision chronologique du temps en SIGNAL : le temps est caractérisé uniquement par l'ordre relatif des instants (leur succession) et leur simultanéité).
- Les relations affines de paramètres (n, φ, d) et (n', φ', d') seront alors considérées équivalentes.

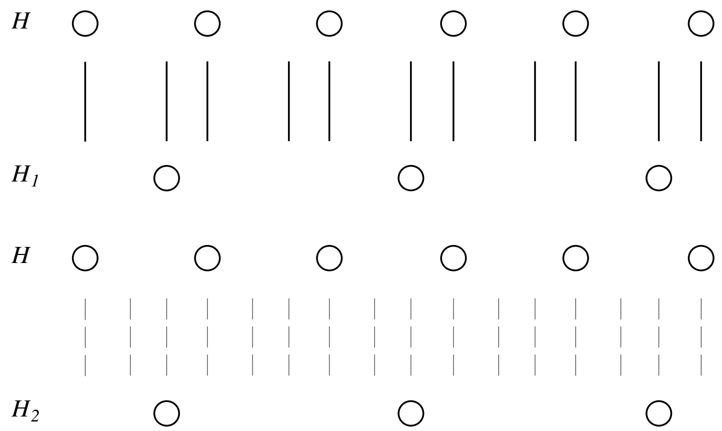

FIG. 5.5 – *Les relations affines de paramètres* $(2, 1, 4)$ *et* $(3, 2, 6)$ *existantes entre* H *et* H_1*, respectivement* H *et* H_2*, sont équivalentes.*

L'équivalence de relations affines est une relation importante de grande utilité dans le calcul d'horloges entre lesquelles il existe une relation affine. Par la relation d'équivalence entre relations affines et par la mise en évidence d'un élément unique dans la classe d'équivalence d'une relation affine – la forme canonique de la relation affine – il sera possible de vérifier des contraintes de synchronisation entre horloges obtenues par transformation affine sur une même horloge de base.

Plus généralement, une relation affine de paramètres (n, φ, d) existe entre *les signaux* x et y si et seulement si :

- l'horloge H_y de y a été obtenue par une transformation affine sur les indices temporels de l'horloge H_x de x ou vice-versa;
- les valeurs des signaux x et y coïncident sur leurs instants de présence communs (intersection de H_x et H_y).

Une relation affine entre les signaux x et y induit une relation affine de mêmes paramètres entres leurs horloges H_1 et H_2. Les propriétés des relations affines feront l'objet d'une étude ultérieure dans le modèle de la sémantique des traces en SIGNAL, conformément au plan que nous exposons dans la suite.

5.3 Organisation de la partie

Les relations affines possèdent des propriétés intéressantes qui peuvent être appliquées pour vérifier des contraintes d'horloges que l'on ne savait pas traiter avec le calcul d'horloges existant dans le compilateur. Nous proposons le modèle mathématique de la sémantique des traces du langage SIGNAL afin d'entreprendre une étude des relations affines et de définir le calcul formel des synchronisations des horloges en relations affines. Le chapitre suivant est dédié à la présentation du modèle; nous commençons par fixer les propriétés des traces et des flots et montrons comment chaque construction syntaxique de SIGNAL induit un *processus* défini comme un ensemble de flots qui satisfont un certain nombre de contraintes. Nous concluons avec les propriétés du monoïde des processus.

Le dernier chapitre est entièrement dédié aux relations affines et à leurs propriétés. Nous définissons un ensemble de propriétés liées à l'identité des relations affines et aux opérations d'union, intersection et différence de relations affines dans le contexte des flots que nous généralisons ensuite dans le contexte des processus. Suite à ce premier calcul, nous introduisons un ensemble de règles de synchronisabilité d'horloges qui permettrons d'étendre le calcul d'horloges existant. Nous concluons avec le modèle de représentation d'une relation affine et la définition des opérateurs affines en termes de flots.

Chapitre 6

Sémantique des traces de Signal (Modèle de suites)

Tout programme Signal est un *processus* définit comme un ensemble de *flots* dans lesquels des *contraintes* (relations) existent sur les variables impliquées. Comme on va le voir dans la suite, les *relations affines* ont des propriétés spécifiques - basées sur un calcul de nombres entiers positifs - qui permettent d'analyser et de traiter les synchronisations d'un programme ; nous envisageons ainsi une étude de ces propriétés dans le contexte des flots et des processus, qui font partie du *modèle sémantique des traces* de Signal. Dans ce chapitre, nous fixons les notions de base de ce modèle et introduisons les flots et les processus et leurs principales propriétés utilisées dans la suite ; nous définissons ainsi les grandes lignes du modèle que nous allons utiliser pour l'étude de propriétés sur des relations affines.

Le terme *signal* est utilisé pour dénoter toute variable du langage : cette dénomination est justifiée par le fait que chaque variable peut être assimilée à une suite (éventuellement infinie) de valeurs dont les instants logiques de présence constituent l'horloge du signal. Les valeurs d'un signal appartiennent toutes à un même sous-domaine d'un domaine de valeurs, désigné par leur type commun. L'horloge du signal permet de définir, relativement à un ensemble totalement ordonné contenant au moins autant d'éléments que la suite des valeurs de ce signal, le sous-ensemble des instants où le signal possède une valeur.

Une valuation associe, à un instant logique du programme, une valeur à une variable (signal). Un *événement* est un ensemble de valuations simultanées des variables d'un programme Signal. Dans un événement, une variable peut ne pas avoir de valeur associée : on dira alors que le signal est absent et on notera $\perp$ sa "valeur". Un événement comporte au moins une valuation.

Une *trace* représente une suite d'événements pour une exécution particulière d'un processus ; parmi les traces on identifie le sous-ensemble des *flots*. Un élément syntaxique de Signal définit du point de vue sémantique un *processus* constitué comme un ensemble de flots définis par des contraintes sur les variables impliquées (contraintes sur les valeurs ou les horloges des variables). Les opérateurs de base (noyau) de Signal ainsi que leur composition et la restriction, définissent un ensemble de programmes Signal qui génère à son tour un ensemble de processus parmi lesquels sont seuls considérés comme légaux les processus pour lesquels l'ordre de chaque ensemble d'instants est conforme à l'ordre induit par les dépendances de calcul (respect du principe de causalité), et qui sont sans définition implicite de signal non booléen.

Dans ce chapitre, nous reprenons les notions fixées ci-dessus, en proposant comme modèle d'étude de propriétés en Signal le *modèle de suites (traces)*. Après avoir fixé les notions de base, nous proposons une sémantique pour l'ensemble des opérateurs élémentaires de Signal ; cette sémantique constituera le support pour la vérification des contraintes exprimées sur les flots d'un processus afin de garantir que le processus est correct par rapport à sa spécification[1].

6.1 Syntaxe

On considère :

- $A = \{a, a_1, \dots, a_n, b, \dots\}$
  un ensemble dénombrable de variables (appelées *ports*) ;
- $F = \{f, f_1, \dots, g, \dots\}$
  un ensemble fini de symboles de fonctions typés ;
- $C = \{\texttt{true}, \texttt{false}, \dots\}$
  un ensemble de constantes (symboles de fonctions d'arité nulle) parmi lesquelles on distingue les constantes booléennes ;
- $T = \{\texttt{event}, \texttt{boolean}, \dots\}$
  un ensemble fini de types (ensembles de valeurs qui peuvent être associées aux variables de l'ensemble A) ;
- les symboles `default`, `when`, `$` .

On définit :

- $GD = \{t\ a\}$
  pour $t \in T$ et $a \in A$, l'ensemble des *déclarations* (association d'un type à une variable) ;
- $GSS = \{a \texttt{ := } g(x_1, \dots, x_n)\}$
  où $x_i \in A \cup C$ et $a \in A$, l'ensemble des générateurs *statiques synchrones* (processus élémentaires) ;
- $GDS = \{a_2 \texttt{ := } a_1 \texttt{ \$ init } x_0\}$
  où $x_0 \in C$ et $a_1 \in A$, $a_2 \in A$, l'ensemble des générateurs *dynamiques synchrones* (processus élémentaires) ;
- $GECB = \{a_3 \texttt{ := } x_1 \texttt{ when } a_2\}$
  avec $x_1 \in A \cup C$ et $a_2, a_3 \in A$, l'ensemble des générateurs d'*extraction sous condition booléenne* (processus élémentaires) ;
- $GM = \{a_3 \texttt{ := } a_1 \texttt{ default } x_2\}$
  avec $x_2 \in A \cup C$ et $a_1, a_3 \in A$ l'ensemble des générateurs de *mélange* (processus élémentaires) ;
- récursivement l'ensemble $PROC$ des processus comme le plus petit ensemble contenant :
  - $G = GD \cup GSS \cup GDS \cup GECB \cup GM$
    l'ensemble des générateurs ;

1. Nous fixons les principales notations utilisées dans la suite :
- On notera $\mathbb{N}$ l'ensemble des entiers positifs (0 inclus).
- Par convention, on notera $A \rightarrow B$ l'ensemble de fonctions f suivant : $\{f \mid f : A \rightarrow B\}$.

- $PC = \{P_1 \mid P_2 \quad$ où P_1 et P_2 appartiennent à $PROC\}$
  (processus de composition) ;
- $PR = \{P_1/a \quad$ où P_1 appartient à $PROC$ et a à $A\}$
  (processus de restriction) .

$(PROC, \ \mid \ , \ / \ )$ représente l'algèbre syntaxique des programmes SIGNAL (G contient les opérateurs de base – les opérateurs qui définissent complètement le langage – ils vont être identifiés comme *primitifs (élémentaires)* dans la suite). A chaque terme appartenant à $PROC$ peut être associée une interprétation sémantique qui sera définie en termes de *flots (traces spécifiques)* sur les variables impliquées – on obtient ainsi l'algèbre sémantique des *processus* SIGNAL. Afin de définir la notion de flot et les propriétés qui lui sont associées, nous fixons dans la suite quelques notions de base qui seront utilisées comme support: les notions d'*événement* et de *trace* et leurs propriétés essentielles.

6.2 Événements

Soit D_a l'ensemble des valeurs que peut prendre une variable $a \in A$. Un processus correspondant à un programme SIGNAL est une suite d'instants logiques et d'événements associés à chaque instant. Une valuation associe, à un instant logique du programme, une valeur à une variable. Un *événement* est l'occurrence de la valuation *simultanée* de variables distinctes; la simultanéité des valuations est intrinsèquement liée au caractère synchrone du langage SIGNAL : toutes les actions (communication et calculs) s'effectuent en temps logique nul.

Dans un événement, une variable peut ne pas avoir de valeur associée : on dira alors que le signal est absent et on notera $\perp$ sa "valeur". Introduisant le symbole $\perp$ ($\perp \notin D_a$) pour désigner l'absence de valuation d'une variable, on note :

$$D_a^{\perp} = D_a \cup \{\perp\}$$

$D^{\perp} = D_{a_1}^{\perp} \cup D_{a_2}^{\perp} \cup \ldots \cup D_{a_{n-1}}^{\perp} \cup D_{a_n}^{\perp}$, $\forall\ a_i \in A$, désigne l'ensemble total de valeurs que peut prendre une variable de A.

Dans $D^{\perp}$, on distingue B=$\{true,\ false\}$, l'ensemble des valeurs booléennes.

Pour A_1 un sous-ensemble non vide de A, on définit la restriction de $D^{\perp}$ à A_1 comme l'ensemble $D_{A_1}^{\perp} = D_{a_1}^{\perp} \cup D_{a_2}^{\perp} \cup \ldots \cup D_{a_{n-1}}^{\perp} \cup D_{a_n}^{\perp}$, $\forall\ a_i \in A_1$. Dans la suite, la notation $D^{\perp}$ sera utilisée pour identifier $D_{A_1}^{\perp}$ quand le domaine A_1 est implicite du contexte.

6.2.1 Définition

Formellement, un événement sur A_1 – l'ensemble des valuations simultanées des variables de l'ensemble A_1 – est une application :

$$e : A_1 \ \rightarrow \ D^{\perp}$$

telle que

- $e(a) = \perp$ indique que a n'a pas de valeur pour l'événement e (on dira aussi que a est absent dans l'événement e).
- $e(a) = v$ indique, pour $v \in D_a$, que a porte la valeur v pour l'événement e (a est présent pour l'événement e).

– $e(A_1)$ est l'image de A_1 par la fonction e.

L'ensemble des *événements* sur A_1 ($A_1 \rightarrow D^{\perp}$) est noté $\mathcal{E}_{A_1}$.

Par convention, on note $e_{\emptyset}$ l'unique événement sur l'ensemble vide de ports. En conséquence, $\mathcal{E}_{\emptyset}$ est un singleton, $\mathcal{E}_{\emptyset} = \{e_{\emptyset}\}$.

Il existe un unique événement e sur l'ensemble total de variables A tel que $\forall a \in A, e(a) = \perp$. On note cet événement $e_{\perp}$.

L'ensemble

$$\mathcal{E} = \bigcup_{A_1 \subseteq A} \mathcal{E}_{A_1}$$

constitue l'ensemble de tous les événements sur l'ensemble total de variables A.

La composition d'événements est en fait la juxtaposition des valuations correspondant aux ensembles de variables des événements respectifs, tandis que la restriction d'un événement à un sous-ensemble A_2 de son ensemble de variables A_1 implique la valuation des variables de l'ensemble A_2 uniquement. Nous définissons formellement la composition et la restriction d'événements, notions utilisées ensuite pour la définition de la composition et restriction de traces.

6.2.2 Composition d'événements

Afin de définir la composition d'événements, on introduit le symbole $\#$ (n'appartenant pas à $\mathcal{E}$) pour désigner l'événement "bloquant". Dans la suite, on note

$$\mathcal{E}^{\#}_{A_i} = \mathcal{E}_{A_i} \cup \{\#\}, \ \forall\ A_i \subseteq A$$

$$\mathcal{E}^{\#} = \mathcal{E} \cup \{\#\}$$

Pour deux événements e_1, e_2 différents de l'événement bloquant tels que $\forall a \in A_1 \cap A_2,\ e_1(a) = e_2(a),$ la composition de e_1 et e_2 est l'événement e_{12} défini comme suit :

$$e_{12}(a) =_{\Delta} \begin{cases} e_1(a), & \text{si } a \in A_1 \\ e_2(a), & \text{si } a \in A_2 \setminus A_1 \end{cases}$$

La composition de deux événements $e_1 : A_1 \rightarrow D^{\perp}$ et $e_2 : A_2 \rightarrow D^{\perp}$ généraux, notée $e_1 \cdot e_2$, est un événement $e \in \mathcal{E}^{\#}_{A_1 \cup A_2}$ tel que :

$$e =_{\Delta} \begin{cases} \#, & \text{si } e_1 = \# \text{ ou } e_2 = \# \text{ ou } \exists a \in A_1 \cap A_2 \text{ tel que } e_1(a) \neq e_2(a) \\ e_{12}, & \text{sinon} \end{cases}$$

On peut constater que l'événement bloquant $\#$ est généré par des valuations différentes pour une même variable, obtenues à partir des événements e_1 et e_2 à composer.

L'ensemble des événements $(\mathcal{E}^{\#}, \cdot, e_{\emptyset})$ est un monoïde commutatif (l'opérateur $\cdot$ de composition est associatif et commutatif et $e_{\emptyset}$ est élément neutre par rapport à la composition).

6.2.3 Restriction d'un événement

Soit e un événement défini sur un ensemble non vide A_1, $e : A_1 \rightarrow D^{\perp}$, et A_2 un sous-ensemble non vide de A_1.
On définit la *restriction* notée $(e)_{\| A_2}$ de e sur A_2 par :

$$\begin{array}{l} (e)_{\| A_2} \ : \ A_2 \ \rightarrow \ D^{\perp}, \\ ((e)_{\| A_2})(a) = e(a), \ (\forall) \ a \in A_2. \end{array}$$

Les restrictions vérifient en particulier les propriétés suivantes :

$$\begin{array}{rcll} (e)_{\| A_1} & = & e, & \forall e \in \mathcal{E}^{\#}_{A_1}, \ A_1 \neq \emptyset \\ (e)_{\| \emptyset} & = & e_{\emptyset} & \\ (\#)_{\| A_1} & = & \#, & \forall A_1 \subseteq A \\ (\#)_{\| \emptyset} & = & e_{\emptyset} & \end{array}$$

6.3 Ensemble des traces

Une *trace* est une suite d'événements pour une exécution particulière d'un processus. Dans cette section nous proposons une définition formelle d'une trace, ainsi que des opérateurs de composition et de restriction associés aux traces. Un sous-ensemble de l'ensemble des traces - *les flots sans interblocage* - constitueront ensuite le support pour la définition de la sémantique des processus élémentaires de SIGNAL. Des processus plus complexes peuvent être construits par composition et restriction de processus élémentaires pour lesquels ces opérations s'appliquent sur les flots des processus opérandes.

Pour tout sous-ensemble A_1 de A, on considère les définitions suivantes.

6.3.1 Définition d'une trace

L'ensemble des applications T, définies sur l'ensemble $\mathbb{N}$ des entiers naturels à valeur dans l'ensemble $\mathcal{E}^{\#}_{A_1}$ des événements sur l'ensemble A_1 non vide, est appelé ensemble des *traces* sur A_1. Il est noté $\mathcal{T}^{\#}_{A_1}$:

$$\mathcal{T}^{\#}_{A_1} = \mathbb{N} \ \rightarrow \ \mathcal{E}^{\#}_{A_1}$$

Ainsi, une trace représente une succession d'évaluations de variables de A_1 dans le temps; un instant de temps sera modélisé par un entier $t \in \mathbb{N}$.

L'ensemble des traces sur l'ensemble vide

$$\mathcal{T}^{\#}_{\emptyset} = \mathbb{N} \ \rightarrow \ \mathcal{E}^{\#}_{\emptyset},$$

est un singleton dont l'unique élément est noté $1_{\mathcal{T}}$.
L'ensemble de toutes les traces est

$$\mathcal{T}^{\#} = \bigcup_{A_1 \subseteq A} \mathcal{T}^{\#}_{A_1}$$

On dira qu'une trace $T : \mathbb{N} \rightarrow \mathcal{E}^{\#}_{A_1}$ est *sans interblocage* si elle ne contient pas d'événement bloquant, i.e. $T(\mathbb{N}) \subseteq \mathcal{E}_{A_1}$ (une trace sans interblocage a la propriété que $T(t) \neq \#$, $\forall t \in \mathbb{N}$).

Il existe une unique trace *sans interblocage* sur A qui, à tout instant, associe à chaque port la valeur $\perp$; elle est notée $0_{\mathcal{T}}$

$$0_{\mathcal{T}} \ : \ \mathbb{N} \ \rightarrow \ (A \ \rightarrow \ \{\perp\})$$

La composition de deux traces est équivalente à la composition, pour chaque instant logique, des événements correspondant à chaque trace. La restriction d'une trace à un sous-ensemble A_2 de son ensemble de variables A_1 produit une trace égale à la suite des événements de la trace d'origine, restreints sur l'ensemble A_2.

6.3.2 Composition de traces

En utilisant l'opérateur de composition d'événements introduit dans le paragraphe 6.2.2 on définit l'opérateur correspondant sur les traces.

La composition de deux traces $T_1 : \mathbb{N} \rightarrow \mathcal{E}^{\#}_{A_1}$, $T_2 : \mathbb{N} \rightarrow \mathcal{E}^{\#}_{A_2}$ est une trace $T \in \mathcal{T}^{\#}_{A_1 \cup A_2}$ telle que:

$$T(t) = T_1(t) \cdot T_2(t), \quad \forall t \in \mathbb{N}.$$

On note $T = T_1 \cdot T_2$.

La composition de deux traces est obtenue en composant à chaque instant les événements associés aux ensembles de variables de chaque trace.

La structure de monoïde commutatif sur l'ensemble des événements $\mathcal{E}^{\#}$ induit une structure similaire sur l'ensemble des traces $\mathcal{T}^{\#}$:

$$\begin{aligned} (T_1 \cdot T_2) \cdot T_3 &= T_1 \cdot (T_2 \cdot T_3) \\ T_1 \cdot T_2 &= T_2 \cdot T_1 \\ T \cdot 1_{\mathcal{T}} &= T \end{aligned}$$

6.3.3 Restriction d'une trace

Soit T une trace de $\mathcal{T}^{\#}_{A_1}$, où A_1 est non vide, et A_2 un sous-ensemble de A_1.
On définit la *restriction* notée $(T)_{\|A_2}$ de T sur A_2 par :

$$\begin{array}{l} (T)_{\|A_2} : \mathbb{N} \ \rightarrow \ \mathcal{E}_{A_2} \\ (\forall t) \quad (\ ((T)_{\|A_2})(t) = (T(t))_{\|A_2} \ ) \end{array}$$

La restriction de T sur A_2 est obtenue en associant à chaque instant logique la restriction sur A_2 de l'événement correspondant à l'instant considéré.

Les restrictions vérifient en particulier les propriétés suivantes ($\forall \ T \in \mathcal{T}^{\#}_{A_1}$) :

$$\begin{aligned} (T)_{\|A_1} &= T \\ (T)_{\|\emptyset} &= 1_{\mathcal{T}} \end{aligned}$$

6.4 Flots

On note $\mathcal{T}_{A_1}$ l'ensemble des traces *sans interblocage* définies sur un ensemble A_1. Dans la suite de cette présentation on ne va considérer que des traces de $\mathcal{T}_{A_1}$.

On introduit la notion de *flot* pour représenter la classe des traces de $\mathcal{T}_{A_1}$ équivalentes à une *densification* près. Pour cela, nous introduisons d'abord la notion de *fonction de densification* ou *densification* pour décrire une fonction strictement croissante qui permet de transformer une trace en modifiant les instants d'exécution de ses événements, tout en gardant leur ordre relatif.

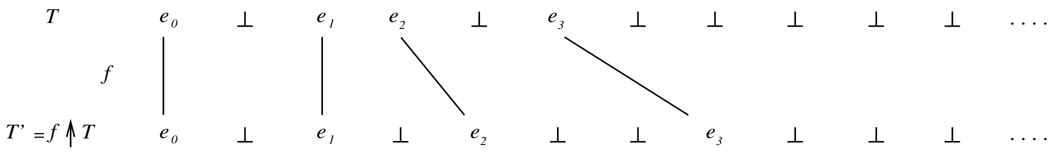

FIG. 6.1 – *Un exemple de densification d'une trace.*

6.4.1 Densification d'une trace

Soient $f : \mathbb{N} \to \mathbb{N}$ une application strictement croissante[2] et T une trace définie sur l'ensemble A_1 non vide. On appelle *observation densifiée* ou plus simplement *densification de* T selon f la trace $f \uparrow T$ définie par

$$(f \uparrow T) : \mathbb{N} \to \mathcal{E}_{A_1}$$
$$(f \uparrow T)(t) = \begin{cases} T(\check{f}^{-1}(t)), & \text{si } t \in f(\mathbb{N}) \\ (e_\perp)_{\|A_1}, & \text{si } t \in \mathbb{N} \setminus f(\mathbb{N}) \end{cases}$$

avec $\check{f}$ la fonction partout égale à f ayant pour co-domaine $f(\mathbb{N})$:

$$\check{f} : \mathbb{N} \to f(\mathbb{N}),$$
$$\check{f}(t) = f(t), \ \forall t \in \mathbb{N}$$

Cette fonction est inversible et strictement croissante (on note $\check{f}^{-1}$ son inverse).

La fonction f est appelée *fonction de densification* ou tout simplement *densification*. Un exemple de densification d'une trace T par une fonction de densification $f(0) = 0$, $f(1) = 1$, $f(2) = 2$, $f(3) = 4$, $f(4) = 5$, $f(5) = 7$, ..., est donné sur la Figure 6.1. La fonction f permet d'étendre la trace T en associant l'événement $(e_\perp)_{\|A_1}$ à chaque instant où l'événement de $f \uparrow T$ n'est pas défini (sur la Figure 6.1, on a préféré la notation abrégée $\perp$ pour $(e_\perp)_{\|A_1}$).

6.4.2 Propriétés des densifications

Par convention, pour toute densification f on a,

$$f \uparrow 1_{\mathcal{T}} = 1_{\mathcal{T}}$$

2. Pour que l'observation densifiée $f \uparrow T$ soit une trace, la fonction f doit être injective (pour ne pas superposer des événements dans $f \uparrow T$) et doit aussi garder l'ordre des instants de la trace T dans la trace $f \uparrow T$, donc f doit être croissante. Une fonction *strictement croissante* combine les deux propriétés.

La trace $0_{\mathcal{T}}$ est élément absorbant pour l'opérateur de densification; pour toute densification f on a alors,

$$f \uparrow 0_{\mathcal{T}} = 0_{\mathcal{T}}$$

Les densifications vérifient en particulier les propriétés suivantes :

Lemme de densification/inversion
Pour toute trace T et toute densification f les égalités suivantes sont vérifiées :

$$\begin{array}{l} ((f \uparrow T))_{\| f(\mathbb{N})} = T \circ \check{f}^{-1} \\ ((f \uparrow T))_{\| f(\mathbb{N})} \circ \check{f} = T \\ (f \uparrow T) \circ f = T \end{array}$$

Lemme de composition
Pour toute trace T et toute paire de densifications (f_1, f_2),

$$f_2 \uparrow (f_1 \uparrow T) = (f_2 \circ f_1) \uparrow T$$

Lemme de "distributivité"
Pour toute densification f et toute paire de traces T_1, T_2,

$$f \uparrow (T_1 \cdot T_2) = (f \uparrow T_1) \cdot (f \uparrow T_2)$$

Équivalence de traces
On dira que deux traces T_1, T_2 sont équivalentes à une densification près si et seulement si il existe une trace T et deux fonctions de densification f_1 et f_2 telles que

$$\begin{array}{l} T_1 = f_1 \uparrow T \\ T_2 = f_2 \uparrow T \end{array} \tag{6.1}$$

On note $\equiv_{\uparrow}$ la relation d'équivalence à une densification près. La Figure 6.2 contient deux traces T_1 et T_2 équivalentes à une densification près.

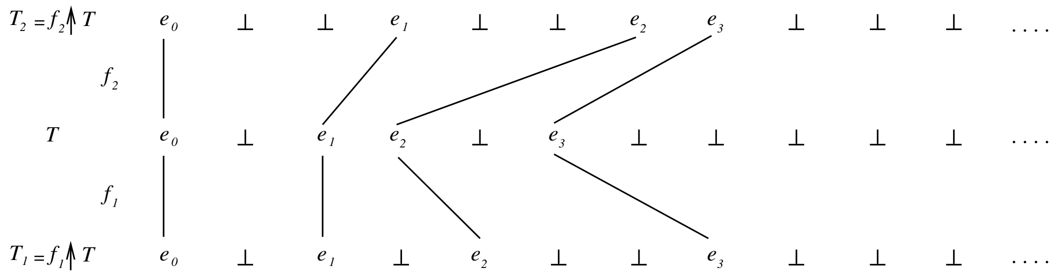

FIG. 6.2 – *Exemple de traces équivalentes à une densification près :* $T_1 \equiv_{\uparrow} T_2$.

6.4.3 Définition d'un flot

On appelle *flot* sur l'ensemble A_1 non vide toute trace T de $\mathcal{T}_{A_1}$ ne contenant pas d'occurrence intermédiaire d'événement vide :

$$(\forall t) \quad (\; (T(t)(A_1) = \{\perp\}) \quad \Rightarrow \quad (\; (\forall s) \quad (\; (s \geq t) \quad \Rightarrow \quad (T(t)(A_1) = \{\perp\}) \;) \;) \;)$$

$0_\mathcal{T}$ est un flot.
$1_\mathcal{T}$ est l'unique flot sur l'ensemble vide.
On note $\mathcal{F}$ (respectivement $\mathcal{F}_{A_1}$) l'ensemble des flots de $\mathcal{T}$ (respectivement $\mathcal{T}_{A_1}$).

Lemme d'égalité de flots
Pour toute paire de flots F_1, F_2 tels que $F_1 \equiv_\uparrow F_2$ il résulte $F_1 = F_2$ (deux flots équivalents coïncident).

Les flots seront considérés comme les représentants des classes des traces équivalentes à une densification près (relation d'équivalence $\equiv_\uparrow$). Pour toute trace T , l'unique flot équivalent à T sera noté $(T)_\downarrow$.

Équivalence de traces (conséquence)
Pour deux traces T_1, T_2 équivalentes à une densification près, il existe deux fonctions de densification f_1 et f_2 telles que

$$T_1 = f_1 \uparrow F$$
$$T_2 = f_2 \uparrow F$$

avec F le représentant de la classe d'équivalence de T_1 et T_2 $(F = (T_1)_\downarrow = (T_2)_\downarrow)$.

Les opérations de composition et restriction de traces seront étendues aux flots. Nous reprenons la restriction pour fixer une notation utilisée dans la suite.

6.4.4 Restriction d'un flot

Soit A_1 un ensemble non vide et A_2 un sous-ensemble de A_1.
Pour un flot $F \in \mathcal{F}_{A_1}$, on définit la *restriction* de F à A_2 notée $\Pi_{A_2}(F)$ par :

$$\Pi_{A_2}(F) : \mathbb{N} \rightarrow \mathcal{E}_{A_2}$$
$$\Pi_{A_2}(F) = ((F)_{\|A_2})_\downarrow$$

La restriction du flot F sur A_2 est en fait le flot équivalent à la restriction de la trace F sur A_2.
Les restrictions vérifient $(\forall \; F \in \mathcal{F}_{A_1})$:

$$\Pi_{A_1}(F) \;=\; F$$
$$\Pi_{\emptyset}(F) \;=\; 1_\mathcal{T}$$

Pour $A_1 = \{a_1, a_2, a_3, a_4\}$ et $A_2 = \{a_2, a_3\}$, on a représenté sur la figure 6.3 la restriction à A_2 d'un flot F appartenant à $\mathcal{F}_{A_1}$.

t		0	1	2	3	4	5	6	7	8	9	
F	a_1	$\perp$	v_1	v_2	$\perp$	v_3	v_4	v_5	v_6	$\perp$	$\perp$	....
	a_2	$\perp$	w_1	$\perp$	w_2	$\perp$	$\perp$	w_3	$\perp$	$\perp$	$\perp$	....
	a_3	x_1	x_2	$\perp$	$\perp$	x_3	$\perp$	x_4	$\perp$	$\perp$	$\perp$	....
	a_4	y_1	y_2	$\perp$	y_3	$\perp$	y_4	$\perp$	y_5	$\perp$	$\perp$	....
$\Pi'_{A_2}(F)$	a_2	$\perp$	w_1	w_2	$\perp$	w_3	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	....
	a_3	x_1	x_2	$\perp$	x_3	x_4	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	....

FIG. 6.3 – *Un exemple de restriction d'un flot.*

6.4.5 Composition de flots

Les flots représentant des traces particulières, la composition de flots est définie comme la composition de traces. Dans toute composition de flots ultérieure, on ne va considérer que des flots dont la composition produit un flot également.

6.4.6 Suites et cardinalité

Suite de valeurs La suite des valeurs portées par une variable x dans un flot F est notée $x_{(F)}$. Elle est définie par

$$(\forall t \in \mathbb{N}) \quad (\ x_{(F)}(t) = (\Pi_{\{x\}}(F))(t)(x)\ )$$

Longueur d'un flot Pour tout flot F de $\mathcal{F}_{A_1}$, soit $\mathbb{N}_{\perp}$ l'ensemble des instants $t \in \mathbb{N}$ tels que $F(t) \neq (e_{\perp})_{\|A_1}$. On définit la *longueur* de F, notée $\mid F \mid$, comme le cardinal $\mid \mathbb{N}_{\perp} \mid$ de $\mathbb{N}_{\perp}$, si $\mathbb{N}_{\perp}$ est fini, et comme la constante ∞, si $\mathbb{N}_{\perp}$ est infini.

La longueur d'un flot $F \in \mathcal{F}_{A_1}$ représente le nombre d'instants où au moins une variable de A_1 est présente (sa valeur est différente de $\perp$).

Par extension, la longueur de la suite de valeurs portées par la variable $x \in A_1$ dans le flot $F \in \mathcal{F}_{A_1}$ est égale à la longueur du flot obtenu par restriction de F à {x} :

$$\mid x_{(F)} \mid \ = \mid \Pi_{\{x\}}(F) \mid$$

6.4.7 Densification minimale

Sur l'ensemble des fonctions de densification on considère un ordre défini comme suit : pour toutes densifications $f_1, f_2 : \mathbb{N} \to \mathbb{N}$, on a $f_1 \leq f_2$ ssi $\forall t \in \mathbb{N},\ f_1(t) \leq f_2(t)$.

Le lemme suivant affirme un résultat très important et fortement utilisé dans la suite du document, spécialement dans le contexte des relations affines : il s'agit de l'unicité de la fonction de densification associée à un flot restreint $\Pi_{A_2}(F)$ dans le flot $F \in \mathcal{F}_{A_1}$, $A_2 \subseteq A_1$.

Pour tout flot $F \in \mathcal{F}_{A_1}$, si on considère A_2 un sous-ensemble de variables de A_1, il existe au moins une fonction de densification f qui permet d'étendre la restriction $\Pi_{A_2}(F)$ de F à A_2 afin d'obtenir le positionnement des variables de l'ensemble A_2 dans le flot F. En conclusion, la densification de $\Pi_{A_2}(F)$ selon f composée avec F produit comme résultat le flot F :

$$F = (f \uparrow \Pi_{A_2}(F)) \cdot F$$

En d'autres mots, l'identité ci-dessus décrit la manière dont le flot $\Pi_{A_2}(F)$ se retrouve dans le flot F englobant. Comme il résultera dans la suite, il existe plusieurs fonctions f qui différent par leurs expressions de définitions sur les instants où $\Pi_{A_2}(F) = (e_\perp)_{\| A_2}$. Parmi celles-ci, on peut identifier une plus petite par rapport à l'ordre $\leq$ défini sur l'ensemble des fonctions de densification.

Lemme de recomposition de flots Pour tout flot F de $\mathcal{F}_{A_1}$ et tout sous-ensemble A_2 de A_1 il existe une unique fonction de densification f minimale telle que:

$$F = (f \uparrow \Pi_{A_2}(F)) \cdot F$$

Preuve La définition de la restriction d'un flot $\Pi_{A_2}(F) = ((F)_{\| A_2})_\downarrow$ implique qu'il existe une densification $f : \mathbb{N} \rightarrow \mathbb{N}$ telle que

$$(F)_{\| A_2} = f \uparrow \Pi_{A_2}(F)$$

Comme $(F)_{\| A_2} \cdot F = F$, il existe une densification $f : \mathbb{N} \rightarrow \mathbb{N}$ telle que

$$F = (f \uparrow \Pi_{A_2}(F)) \cdot F$$

Nous construisons une fonction f qui vérifie l'identité antérieure. Pour cela, on se fixe les notations suivantes : $U = \{u \in \mathbb{N} \ / \ u < \mid \Pi_{A_2}(F) \mid \}$, $S = \{s \in \mathbb{N} \ / \ F(s)(A_2) \neq \{\perp\}\}$ et $V_s = \{v \in \mathbb{N} \ / \ F(v)(A_2) \neq \{\perp\}, v < s\}$. Soit $g : U \rightarrow S$ définie comme suit : $\forall u \in U, g(u) = s \in S$ tel que $\mid V_s \mid = u$. On considère f égale à g sur $U \subseteq \mathbb{N}$: $f : \mathbb{N} \rightarrow \mathbb{N}$, $f(u) = g(u), \forall u \in U$. Il reste à définir f sur $\mathbb{N} \setminus U = \{u \in \mathbb{N} \ / \ u \geq \mid \Pi_{A_2}(F) \mid \}$.

Si $\mid \Pi_{A_2}(F) \mid \neq \infty$, il existe une infinité de fonctions f telles que $F = (f \uparrow \Pi_{A_2}(F)) \cdot F$ puisqu'il existe une infinité de manières de définir f sur $\{u \in \mathbb{N} \ / \ u \geq \mid \Pi_{A_2}(F) \mid \} \subseteq \mathbb{N}$. Parmi celles-ci, la fonction

$$f : \mathbb{N} \rightarrow \mathbb{N}, f(t) = \begin{cases} g(t), & \text{si } t < \mid \Pi_{A_2}(F) \mid \\ g(\mid \Pi_{A_2}(F) \mid -1) + t - \mid \Pi_{A_2}(F) \mid +1, & \text{si } t \geq \mid \Pi_{A_2}(F) \mid \end{cases}$$

est minimale.

Si $\mid \Pi_{A_2}(F) \mid = \infty$, $U = \mathbb{N}$ et $f = g$ (uniquement définie). La fonction f est minimale. □

Conséquence En particulier, si $A_2 = A_1$, la fonction f est l'identité[3] sur $\mathbb{N}$.

$$A_2 = A_1 \Rightarrow \begin{cases} F = \Pi_{A_2}(F) \\ f = id_{\mathbb{N}} \end{cases}$$

Dans la figure 6.4, f_1 et f_2 sont deux exemples de fonctions de densification telles que $F = (f_1 \uparrow \Pi_{A_2}(F)) \cdot F = (f_2 \uparrow \Pi_{A_2}(F)) \cdot F$ $(f_1(t) = f_2(t), \forall\ t \in \{0,1,2,3,4,5\}, f_1(6) = 8, f_1(7) = 9, f_2(6) = 10, f_2(7) = 12)$; la densification f_1 est minimale.

Dans la preuve du lemme précédent, nous avons vu l'existence d'un ensemble infini de fonctions de densification f telles que

$$F = (f \uparrow \Pi_{A_2}(F)) \cdot F$$

et nous avons mis en évidence une fonction de densification minimale dans cet ensemble. Dans la suite, nous allons considérer que toute fonction de densification possède la propriété d'être minimale. En conséquence, afin de construire l'observation densifiée (densification) d'une trace T selon f, la fonction de densification f doit être strictement croissante et minimale avec les propriétés de la définition d'une densification.

---

3. $id_{\mathbb{N}} : \mathbb{N} \rightarrow \mathbb{N}, id_{\mathbb{N}}(t) = t, \forall t \in \mathbb{N}$

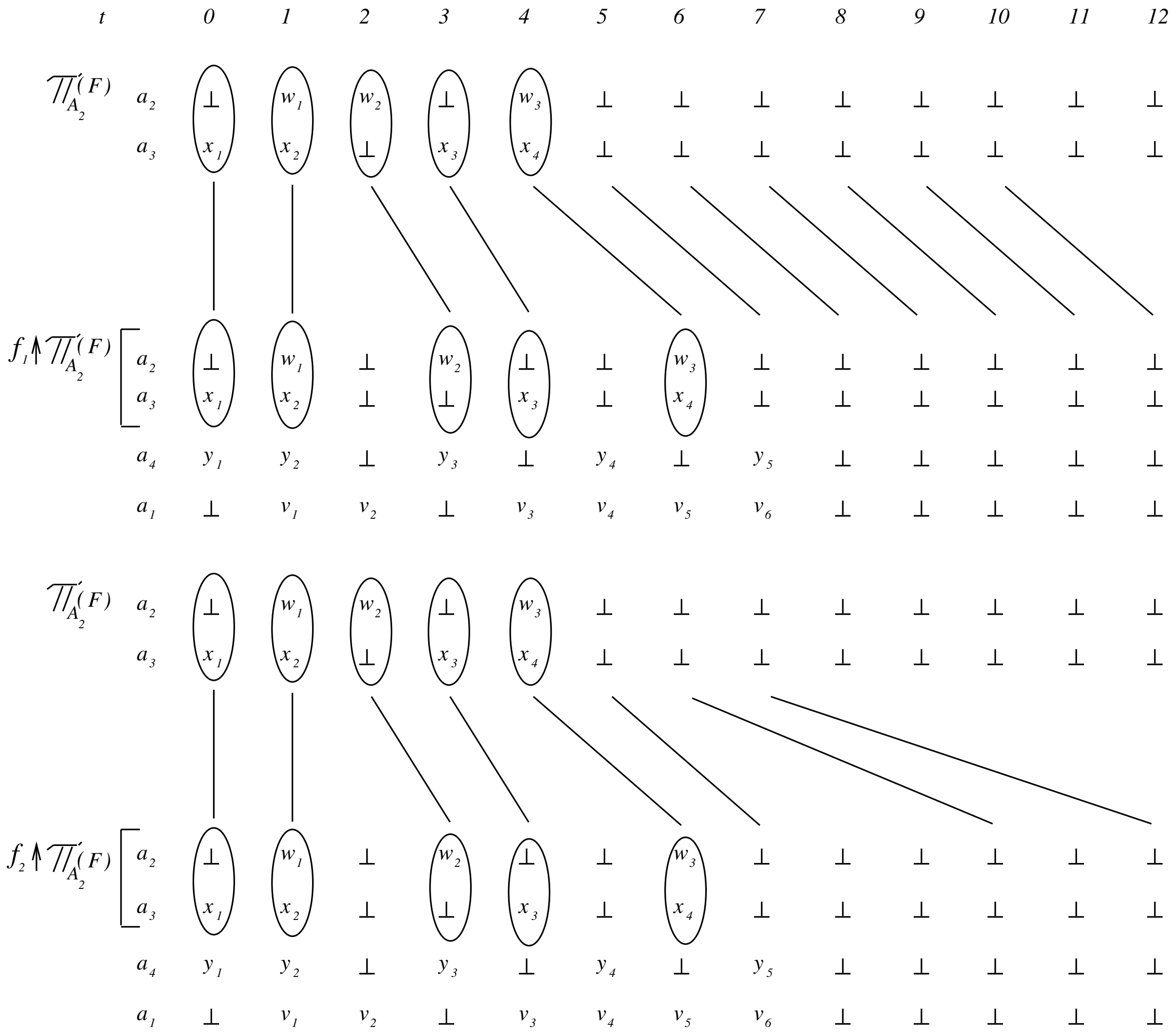

FIG. 6.4 – *Illustration du lemme de recomposition de flots.*

Le concept de flot sera utilisé afin de définir la sémantique des constructeurs SIGNAL dont la syntaxe a été présentée dans la section 6.1 : chaque construction syntaxique appartenant à l'ensemble $PROC$ qui utilise un ensemble de variables $A_1 \subseteq A$ induit du point de vue sémantique un *processus* défini par des *contraintes* (relations) sur les valeurs et les instants de présence des variables de l'ensemble A_1. Ces contraintes seront exprimées en utilisant les notions de variables "synchrones" et "identiques" dans un flot F - que nous introduisons dans la section suivante - étendues aux concepts de variables "synchronisables" et "identifiables", respectivement "synchrones" et "identiques" dans le contexte des processus vus comme ensembles de flots.

6.4.8 Propriétés sur les flots

6.4.8.1 Variables synchrones dans un flot

Notation : $x \overset{F}{\hat{=}} y$

Deux variables x et y sont dites *synchrones* dans $F \in \mathcal{F}_{A_1}$, $\{x,y\} \subseteq A_1$, ssi $\mid \Pi_{\{x\}}(F) \mid =$

$\mid \Pi_{\{y\}}(F) \mid$ et il existe une fonction de densification g telle que

$$F = (g \uparrow \Pi_{\{x\}}(F)) \cdot (g \uparrow \Pi_{\{y\}}(F)) \cdot F$$

En d'autres mots, les variables x et y sont synchrones dans F si et seulement si leurs instants de présence coïncident.

La propriété de distributivité suivante est vérifiée : $F = g \uparrow (\Pi_{\{x\}}(F) \cdot \Pi_{\{y\}}(F)) \cdot F$.

La relation $\stackrel{F}{\hat{=}}$ est une relation d'équivalence.

6.4.8.2 Variables identiques dans un flot

On note : $x \stackrel{F}{=}_{\downarrow} y$

Deux variables x et y sont dites *équivaluées* dans un flot F ssi $x_{(F)} = y_{(F)}$.

En d'autres mots, x et y sont équivaluées dans F ssi leurs suites de valeurs sont identiques.

La relation $\stackrel{F}{=}_{\downarrow}$ est une relation d'équivalence.

On note : $x \stackrel{F}{=} y$

Deux variables x et y sont *identiques* dans F ssi $x \stackrel{F}{\hat{=}} y$ et $x \stackrel{F}{=}_{\downarrow} y$.

Deux variables x et y sont identiques dans un flot F ssi elles sont synchrones et équivaluées, c'est-à-dire que leurs suites de valeurs, ainsi que leurs instants de présence, sont les mêmes.

La relation $\stackrel{F}{=}$ est une relation d'équivalence.

6.4.8.3 Horloge d'une variable

Conceptuellement, l'horloge d'un signal (variable) peut être identifiée à la suite d'instants où la variable est présente.

Pour toute variable $x \in A_1$, nous définissons l'*horloge de x* par la variable $h(x) \in A_1$ satisfaisant les conditions suivantes :

– $\forall\ F \in \mathcal{F}_{A_1}$, $h(x)$ et x sont synchrones dans F :

$$x \stackrel{F}{\hat{=}} h(x)$$

– Les valeurs de $h(x)$ différentes de $\perp$ sont toujours égales à *true* :

$$\forall\ t \in \mathbb{N},\ t \leq \mid \Pi_{\{h(x)\}}(F) \mid\ ,\ \ \Pi_{\{h(x)\}}(F)(t)(h(x)) = true$$

Soit $M \subseteq \mathbb{N}$ l'ensemble des instants $t \in \mathbb{N}$ tels que x et $h(x)$ sont absentes pour tout instant $t' \geq t$. Dans ce cas, $f(\mathbb{N}) \setminus M$ décrit l'ensemble des instants de présence des variables x et $h(x)$. Comme toute horloge $h(x)$ est indépendante des valeurs de la variable x et possède la valeur *true* à ses instants de présence, on peut définir un isomorphisme [Ama95] entre l'ensemble des horloges associées aux variables d'un programme SIGNAL et le treillis booléen des formules booléennes : on associe à chaque horloge $h(x)$ une variable booléenne h à valeur *true* quand h est présente et vraie et à valeur $false$ sinon.

Aussi, un isomorphisme entre l'espace des horloges d'un programme SIGNAL et le treillis booléen d'ordre $\leq$ et de borne supérieure $\mathbb{N}$ des sous-ensembles de $\mathbb{N}$ peut être construit en associant à chaque $h(x)$ l'ensemble $f(\mathbb{N}) \setminus M$ des instants de présence de x.

Le concept d'horloge permet d'introduire les propriétés suivantes :

$$\forall F \in \mathcal{F}_{A_1},\ \{x,y\} \subseteq A_1, \quad x \overset{F}{\hat{=}} y \ \Leftrightarrow\ h(x) \overset{F}{=} h(y)$$
$$\forall F \in \mathcal{F}_{A_1},\ \{x,y\} \subseteq A_1, \quad x \overset{F}{=} y \ \Rightarrow\ h(x) \overset{F}{=} h(y)$$

Le synchronisme des variables x et y est équivalent à la contrainte que leurs horloges soient identiques. Or, les contraintes sur les flots des processus SIGNAL seront, entre autres, des relations "identique" entre horloges. La vérification de ces contraintes nécessitera la résolution d'un système d'identités sur horloges, en utilisant un système de règles de synchronisabilité (identifiabilité) déduites de la structure de treillis booléen et d'autres propriétés sur horloges (propriétés des conditions booléennes, propriétés des horloges obtenues par transformation affine).

Nous définissons maintenant les opérations d'union, intersection et différence de variables générales et horloges en particulier; celles-ci vont intervenir dans les expressions sur horloges du système de contraintes associé à tout programme SIGNAL.

Nous présentons ensuite les processus SIGNAL et les propriétés "synchronisable", "synchrone", "identifiable" et "identique" dans ce contexte.

6.4.8.4 Opérations sur variables et horloges

Considérons A_1 un ensemble de variables et x, y deux variables arbitraires appartenant à A_1. Pour tout flot $F \in \mathcal{F}_{A_1}$, il existe f_x, f_y densifications telles que le flot F s'écrit :

$$F \ = \ (\ f_x \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ f_y \uparrow \Pi_{\{y\}}(F)\ ) \cdot F$$

Nous définissons les opérations d'*union avec priorité* $\vee_v$, *intersection avec priorité* $\wedge_v$ et *différence* $\setminus_v$ sur x et y telles que les résultats, respectivement $u_v, i_v, d_v \in A_1$, de ces opérations satisfont les conditions :

$\forall\ F \in \mathcal{F}_{A_1}$, F s'écrit :

$$F \ = \ (\ f_{u_v} \uparrow \Pi_{\{u_v\}}(F)\ ) \cdot (\ f_{i_v} \uparrow \Pi_{\{i_v\}}(F)\ ) \cdot (\ f_{d_v} \uparrow \Pi_{\{d_v\}}(F)\ ) \cdot F$$

où les fonctions de densification f_{u_v}, f_{i_v} et f_{d_v} et les flots de valeurs $\Pi_{\{u_v\}}(F)$, $\Pi_{\{i_v\}}(F)$ et $\Pi_{\{d_v\}}(F)$ vérifient les équations suivantes :

$$f_{u_v}(\mathbb{N}) = f_x(\mathbb{N}) \cup f_y(\mathbb{N})$$
$$\forall\ t \in \mathbb{N},\ \Pi_{\{u_v\}}(F)(t)(u_v) = \begin{cases} \Pi_{\{x\}}(F)(s_1)(x), & \text{si } \exists\ s_1 \in \mathbb{N},\ f_{u_v}(t) = f_x(s_1) \\ & \text{et } \Pi_{\{x\}}(F)(s_1)(x) \neq \perp \\ \Pi_{\{y\}}(F)(s_2)(y), & \text{avec } s_2 \in \mathbb{N},\ f_{u_v}(t) = f_y(s_2),\ \text{sinon} \end{cases}$$

$$f_{i_v}(\mathbb{N}) = f_x(\mathbb{N}) \cap f_y(\mathbb{N})$$
$$\forall\ t \in \mathbb{N},\ \Pi_{\{i_v\}}(F)(t)(i_v) = \Pi_{\{x\}}(F)(s)(x),\ \text{avec } s \in \mathbb{N},\ f_{i_v}(t) = f_x(s)$$

$$f_{d_v}(\mathbb{N}) = f_x(\mathbb{N}) \setminus f_y(\mathbb{N})$$
$$\forall\ t \in \mathbb{N},\ \Pi_{\{d_v\}}(F)(t)(d_v) = \Pi_{\{x\}}(F)(s)(x),\ \text{avec } s \in \mathbb{N},\ f_{d_v}(t) = f_x(s)$$

avec f_x, f_y les fonctions de densification et $\Pi_{\{x\}}(F)$, $\Pi_{\{y\}}(F)$ les flots de valeurs associés aux variables x et y dans[4] F.

Concernant les horloges, on obtient les relations suivantes :

$$h(u_v) = h(x) \vee_v h(y)$$
$$h(i_v) = h(x) \wedge_v h(y)$$
$$h(d_v) = h(x) \setminus_v h(y)$$

Dans la suite, les opérations d'union, intersection et différence d'horloges seront notées simplement $\vee, \wedge, \setminus$.

6.5 Processus

Pour un ensemble $A_1 \subseteq A$, on définit un *processus* comme un ensemble de flots sur A_1, c'est-à-dire un sous-ensemble de $\mathcal{F}_{A_1}$. $\mathcal{P}(\mathcal{F}_{A_1})$[5] représente ainsi l'ensemble des processus sur A_1 et sera noté dans la suite $\mathcal{P}_{A_1}$; tout élément $\mathbf{P} \in \mathcal{P}_{A_1}$ est un processus sur A_1.

Chaque construction syntaxique appartenant à l'ensemble $PROC$ (défini dans la section 6.1) et qui utilise un ensemble de variables $A_1 \subseteq A$ induit du point de vue sémantique un *processus* $\mathbf{P} \in \mathcal{P}_{A_1}$ (noté aussi $\mathbf{P} : \mathcal{P}_{A_1}$ pour mettre en évidence l'appartenance à $\mathcal{P}_{A_1}$) défini par des *contraintes* (relations) sur les flots de $\mathcal{F}_{A_1}$. Du point de vue sémantique, un *processus* SIGNAL général sur $A_1 \subseteq A$ est ainsi un ensemble de flots sur A_1 défini à partir des opérateurs primitifs, de la composition et de la restriction.

6.5.1 Définition des opérateurs primitifs en SIGNAL

Dans ce paragraphe, nous présentons chaque processus élémentaire de SIGNAL comme un ensemble de flots qui doivent satisfaire un certain nombre de contraintes. Plus loin dans notre exposé, nous allons voir que, à partir de ces contraintes, on peut associer à tout processus élémentaire un système d'équations sur horloges et un graphe de dépendances conditionnées entre valeurs qui constituent le support de vérification, par le compilateur de SIGNAL, de la cohérence de ces contraintes.

Pour un flot $F \in \mathcal{F}_{A_1}$ et tout signal $x \in A_1$, on note $F_t(x)$ la valeur de x à l'instant t.

6.5.1.1 Processus monochrones

Un processus $\mathbf{P} : \mathcal{P}_{A_1}$ défini sur A_1 non vide est appelé *monochrone* si, à tout instant $t \in \mathbb{N}$ durant lequel l'un des signaux est présent (respectivement, absent), tous le sont :

$$\forall\ F \in \mathbf{P} : \mathcal{P}_{A_1},\ \ \forall\ t \in \mathbb{N},\ \ (\ \perp\ \in F_t(A_1) \Rightarrow F_t(A_1) = \{\perp\}\ )$$

Processus monochrones statiques Soit g un symbole de fonction typée. Sous une interprétation I pour laquelle l'interprétation de g est notée $[|g|]_I$, le terme

$$x_{n+1} \texttt{ := } g(\ x_1, ..., x_n\ )$$

4. Si $f_{i_v}(\mathbb{N}) = \emptyset$ et $f_{d_v}(\mathbb{N}) = \emptyset$, on considère, par convention, que f_{i_v} et f_{d_v} sont égales à la constante ∞ sur tout leur domaine de définition : $f_{i_v}(t) = \infty$, $\forall t \in \mathbb{N}$ et $f_{d_v}(t) = \infty$, $\forall t \in \mathbb{N}$. Dans ce cas : ($\Pi_{\{i_v\}}(F) = 0_{\mathcal{T}} \Leftrightarrow \mid \Pi_{\{i_v\}}(F) \mid\ = 0$) et ($\Pi_{\{d_v\}}(F) = 0_{\mathcal{T}} \Leftrightarrow \mid \Pi_{\{d_v\}}(F) \mid\ = 0$).

5. Pour tout ensemble A non vide, $\mathcal{P}(A)$ représente l'ensemble de tous les sous-ensembles de A.

définit un processus $\mathbf{P} \in \mathcal{P}_{\{x_1,\ldots,x_{n+1}\}}$ par une relation liant la suite des valeurs du signal x_{n+1} à la suite obtenue par l'extension terme à terme de l'application de g, sous cette interprétation, à la suite des tuples de valeurs des signaux $x_1,\ldots,x_n$:

$$\begin{array}{llll}\mathbf{P} =_\Delta \{ \ F \in \mathcal{F}_{\{x_1,\ldots,x_{n+1}\}} \ / & (\forall t) & (& (F_t(x_1) \ = \ldots = F_t(x_{n+1}) \ = \{\perp\}) \\ & & & \bigvee \ \ (\ (\exists v_1,\ldots,v_n,v_{n+1}) \ \ (\\ & & & \quad (\ (\forall x_i) \ \ (\ (v_i \neq \perp) \ \bigwedge \ (F_t(x_i) = v_i) \ ) \ ) \\ & & & \quad\quad \bigwedge \ \ (v_{n+1} = [|g|]_I(v_1,\ldots,v_n)) \ ) \ ) \quad) \ \}\end{array}$$

Processus monochrones dynamiques : le retard Le terme

$$a_2 \ \texttt{:=} \ a_1 \ \texttt{\$ init} \ x_0$$

définit un processus $\mathbf{P} \in \mathcal{P}_{\{a_1,a_2\}}$ par la relation imposant l'égalité de la suite des valeurs du signal a_2 et de la suite des valeurs du signal a_1, décalée d'un instant ; x_0 est la valeur initiale de a_2.

$$\begin{array}{llll}\mathbf{P} =_\Delta \{ \ F \in \mathcal{F}_{\{a_1,a_2\}} \ / & (\forall t) & (& (F_t(a_1) \ = F_t(a_2) \ = \{\perp\}) \\ & & & \bigvee \ \ (((t \geq 1) \ \bigwedge \ (F_t(a_2) = F_{t-1}(a_1))) \\ & & & \bigvee \ \ ((F_0(a_1) \neq \perp) \ \bigwedge \ (F_0(a_2) = x_0))) \quad) \ \}\end{array}$$

6.5.1.2 Processus polychrones

Un processus $\mathbf{P} : \mathcal{P}_{A_1}$ défini sur A_1 non vide est appelé *polychrone* s'il contient un flot F pour lequel il existe un instant t durant lequel l'un des signaux est présent alors qu'un autre ne l'est pas. Par extension, on dira qu'un terme est polychrone s'il permet de définir des processus polychrones.

Extraction sous condition booléenne Le terme

$$a_3 \ \texttt{:=} \ x_1 \ \texttt{when} \ a_2$$

définit un processus $\mathbf{P} \in \mathcal{P}_{\{x_1,a_2,a_3\}}$ par la relation imposant l'égalité de la suite des valeurs du signal a_3 et de la suite des occurrences de valeurs du signal x_1 lorsque le signal booléen a_2 possède la valeur *true*. On dit que a_3 est une extraction (sous-échantillonnage) de x_1 sous la condition booléenne a_2.

$$\begin{array}{llll}\mathbf{P} =_\Delta \{ \ F \in \mathcal{F}_{\{x_1,a_2,a_3\}} \ / & (\forall t) & (& ((F_t(a_2) = \mathit{true}) \ \bigwedge \ (F_t(a_3) = F_t(x_1))) \\ & & & \bigvee \ \ ((F_t(a_2) \neq \mathit{true}) \ \bigwedge \ (F_t(a_3) = \perp)) \quad) \ \}\end{array}$$

Entrelacement de signaux Le terme

$$a_3 \ \texttt{:=} \ a_1 \ \texttt{default} \ x_2$$

définit un processus $\mathbf{P} \in \mathcal{P}_{\{a_1,x_2,a_3\}}$ par la relation imposant l'égalité de la suite des valeurs du signal a_3 et de la suite formée des occurrences de valeurs du signal a_1 ou par défaut des occurrences de valeurs du signal x_2.

$$\begin{array}{llll}\mathbf{P} =_\Delta \{ \ F \in \mathcal{F}_{\{a_1,x_2,a_3\}} \ / & (\forall t) & (& ((F_t(a_1) \neq \perp) \ \bigwedge \ (F_t(a_3) = F_t(a_1))) \\ & & & \bigvee \ \ ((F_t(a_1) = \perp) \ \bigwedge \ (F_t(a_3) = F_t(x_2))) \quad) \ \}\end{array}$$

6.5.2 Composition de processus

La définition d'un processus comme un ensemble de flots et l'opérateur de composition sur les flots induisent un opérateur de composition similaire sur les processus.

Considérant deux processus $\mathbf{P}_1$ et $\mathbf{P}_2$ sur les ensembles de variables A_1 et A_2, la composition de $\mathbf{P}_1$ et $\mathbf{P}_2$, notée $\mathbf{P}_1 \mid \mathbf{P}_2$, définit un processus $\mathbf{P} \in \mathcal{P}_{A_1 \cup A_2}$ de la manière suivante:

$$\mathbf{P} =_\Delta \{ \ F \in \mathcal{F}_{A_1 \cup A_2} \ / \ \Pi_{A_1}(F) \in \mathbf{P}_1 \text{ et } \Pi_{A_2}(F) \in \mathbf{P}_2 \ \}$$

Le processus $\mathbf{P}$ est ainsi l'ensemble de flots obtenus par composition non bloquante de traces, densifiées avec des fonctions f_1 et f_2, appartenant à $\mathbf{P}_1$ et $\mathbf{P}_2$.

La figure 6.5 contient un exemple de composition de processus.

6.5.3 Restriction d'un processus

Soit $\mathbf{P}_1$ un processus sur l'ensemble de variables A_1 et A_2 un sous-ensemble de A_1. On définit la restriction de $\mathbf{P}_1$ à $A_1 \setminus A_2$, notée $\mathbf{P}_1 \ / \ \mathbf{A}_2$, comme le processus $\mathbf{P} \in \mathcal{P}_{A_1 \setminus A_2}$ tel que

$$\mathbf{P} =_\Delta \{ \ F \in \mathcal{F}_{A_1 \setminus A_2} \ / \ \ (\exists F_1) \quad (\ (F_1 \in \mathbf{P}_1) \bigwedge (F = \Pi_{A_1 \setminus A_2}(F_1)) \ ) \ \ \}$$

Le processus $\mathbf{P}$ est le résultat de la restriction des flots de $\mathbf{P}_1$ à l'ensemble $A_1 \setminus A_2$.

Pour définir l'opérateur de restriction sur processus, nous avons utilisé la notation existante dans la syntaxe du langage SIGNAL. Pour maintenir un certain degré de cohérence avec les notations antérieures et pour plus de lisibilité, nous utiliserons une notation équivalente de la forme $\Pi_{A_1 \setminus A_2}(\mathbf{P}_1)$. En conséquence, on a les égalités suivantes :

$$\mathbf{P} = \mathbf{P}_1 / \mathbf{A}_2 = \Pi_{A_1 \setminus A_2}(\mathbf{P}_1)$$

6.5.4 Propriétés des processus

Un processus SIGNAL peut être un opérateur primitif ou peut être obtenu par composition ou restriction de processus à partir des opérateurs primitifs. De manière générale, un processus SIGNAL est un ensemble de flots *contraints* par les valeurs que les signaux impliqués peuvent prendre et par les relations entre leurs instants de présence/absence respectifs (on verra dans la suite que ces relations se traduisent dans un ensemble de contraintes sur les horloges des signaux). La résolution du système de contraintes sur horloges - nécessaire pour vérifier que le programme est correct du point de vue des contraintes exprimées - repose sur un système de règles de synchronisabilité déduites des propriétés de l'espace des horloges. Nous donnons la signification des termes "synchronisable" et "identifiable", qui représentent des extensions de "synchrone" et "identique" présents au niveau des flots.

6.5.4.1 Synchronisabilité

$$x \overset{\mathbf{P}}{\odot} y$$

Deux variables x et y sont *synchronisables* dans le processus $\mathbf{P}$ ssi *il existe* un flot $F \in \mathbf{P}$ tel que $x \overset{F}{\hat{=}} y$ (x et y sont synchrones dans F).

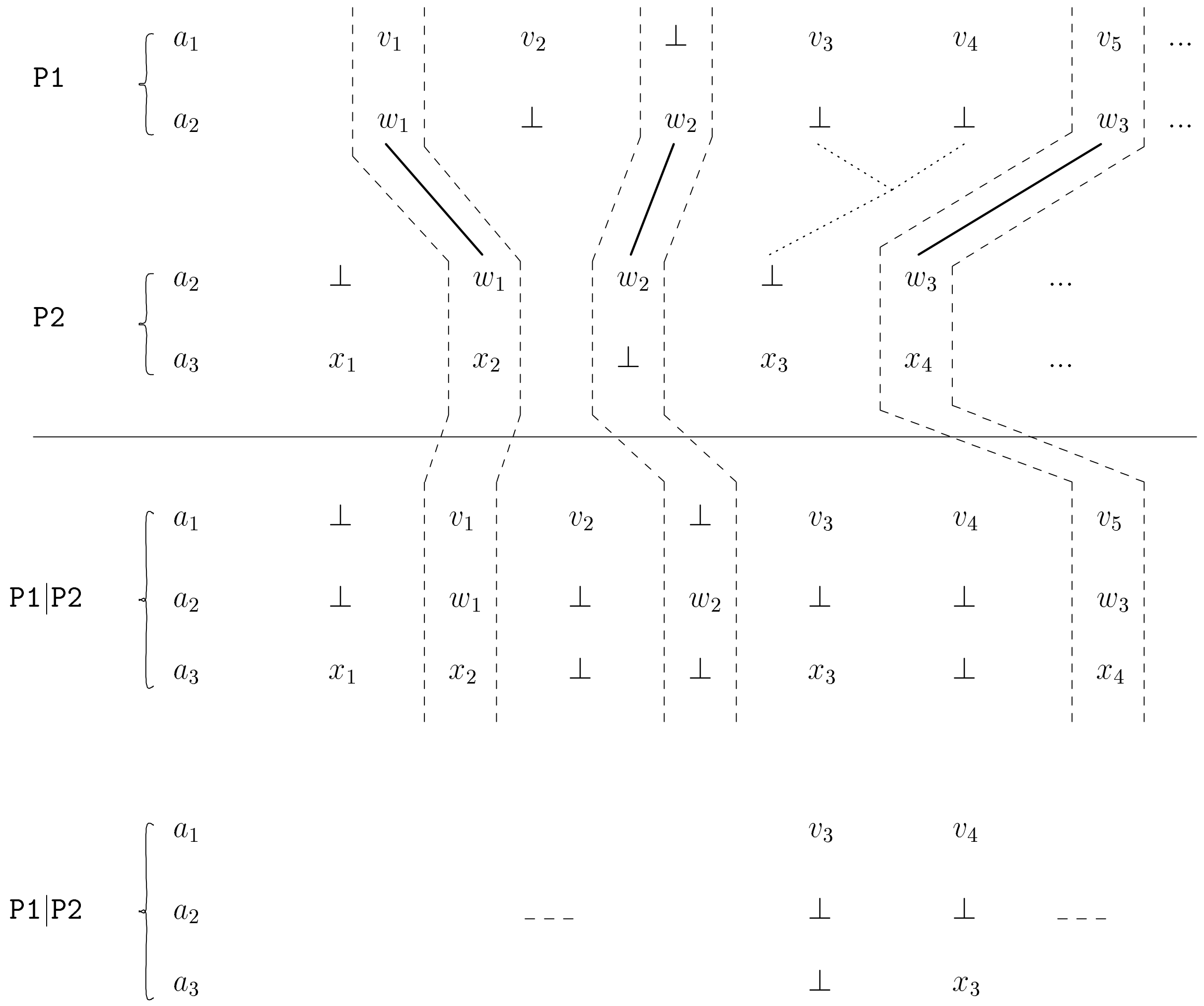

FIG. 6.5 – *Deux configurations possibles pour la composition de* $\mathtt{P_1}$ *et* $\mathtt{P_2}$.

La relation synchronisable n'est pas une équivalence, elle n'est pas transitive (elle est une relation reflexive et symétrique) : $x \overset{\mathbf{P}}{\odot} y$ et $y \overset{\mathbf{P}}{\odot} z$ n'implique pas toujours $x \overset{\mathbf{P}}{\odot} z$. Effectivement, si $x \overset{\mathbf{P}}{\odot} y$ et $y \overset{\mathbf{P}}{\odot} z$, le flot $F_1 \in \mathbf{P}$ dans lequel x et y sont synchrones ($x \overset{F_1}{\hat{=}} y$) n'est pas forcément le même que le flot $F_2 \in \mathbf{P}$ dans lequel y et z sont synchrones ($y \overset{F_2}{\hat{=}} z$), et l'existence de F_1 et F_2 ne garanti pas l'existence d'un flot $F_3 \in \mathbf{P}$ dans lequel x et z sont synchrones ($x \overset{F_3}{\hat{=}} z$).

$x \overset{\mathbf{P}}{\hat{=}} y$

Les variables x et y sont *synchrones* dans le processus $\mathbf{P}$ ssi *pour tout* flot $F \in \mathbf{P}$, $x \overset{F}{\hat{=}} y$.

La relation $\overset{\mathbf{P}}{\hat{=}}$ est une équivalence; la notation $\hat{=}$ sera utilisée quand aucune confusion n'est possible. Comme pour la relation synchronisable dans $\mathbf{P}$, il est immédiat que la relation synchrone dans **P** est reflexive et symétrique ; contrairement à la relation synchronisable, c'est une relation transitive puisque c'est une propriété valable dans *tous les flots* du processus **P** et $x \overset{\mathbf{P}}{\hat{=}} y$ et $y \overset{\mathbf{P}}{\hat{=}} z$ implique $x \overset{\mathbf{P}}{\hat{=}} z$.

6.5.4.2 Signaux identiques

$x \overset{\mathbf{P}}{\odot_\downarrow} y$

Les variables x et y sont *équivaluables* dans **P** ssi *il existe* un flot $F \in$ **P** tel que $x \overset{F}{=}_\downarrow y$ (x et y sont équivaluées dans F).

La relation équivaluable n'est pas une équivalence.

$x \overset{\mathbf{P}}{=}_\downarrow y$

Deux variables x et y sont *équivaluées* dans **P** ssi *pour tout* flot $F \in$ **P**, $x \overset{F}{=}_\downarrow y$.

La relation équivalué est une équivalence ; elle sera notée $=_\downarrow$ quand aucune confusion n'est possible.

$x \overset{\mathbf{P}}{\ominus} y$

Deux variables x et y sont *identifiables* dans **P** ssi *il existe* un flot $F \in$ **P**, $x \overset{F}{=} y$ (x et y sont identiques dans F).

La relation identifiable n'est pas une équivalence.

$x \overset{\mathbf{P}}{=} y$

x et y sont *identiques* dans **P** ssi *pour tout* flot $F \in$ **P** on a $x \overset{F}{=} y$.

Deux variables x et y sont identiques dans le processus **P** ssi elles sont synchrones et équivaluées dans **P**.

La relation identique est une équivalence. Quand le processus **P** est implicite, on notera $x = y$ cette propriété.

6.5.4.3 Horloges

Les propriétés *"x et y sont synchronisables"* et *"x et y sont synchrones"* dans un processus **P** peuvent s'exprimer en termes d'horloges de la manière suivante :

$$x \overset{\mathbf{P}}{\odot} y \iff h(x) \overset{\mathbf{P}}{\ominus} h(y)$$
$$x \overset{\mathbf{P}}{\hat{=}} y \iff h(x) \overset{\mathbf{P}}{=} h(y)$$

Aussi, les horloges de deux variables identifiables dans $\mathbf{P}$ sont identifiables dans $\mathbf{P}$; les horloges de deux variables identiques dans $\mathbf{P}$ sont identiques dans $\mathbf{P}$:

$$\begin{array}{lcl} x \overset{\mathbf{P}}{\ominus} y & \Rightarrow & h(x) \overset{\mathbf{P}}{\ominus} h(y) \\ x \overset{\mathbf{P}}{=} y & \Rightarrow & h(x) \overset{\mathbf{P}}{=} h(y) \end{array}$$

6.5.4.4 Le monoïde des processus

Le processus $\mathbf{1}_{\mathcal{P}}$ étant le processus ayant la trace $1_{\mathcal{T}}$ comme seul élément, l'ensemble des processus $\mathcal{P}_A$ est un *monoïde commutatif :* $\forall\ \mathbf{P}_1$, $\mathbf{P}_2$ et $\mathbf{P}_3 \in \mathcal{P}_A$,

$$\begin{array}{l} (\mathbf{P}_1 \mid \mathbf{P}_2) \mid \mathbf{P}_3 = \mathbf{P}_1 \mid (\mathbf{P}_2 \mid \mathbf{P}_3) \\ (\mathbf{P}_1 \mid \mathbf{P}_2) = (\mathbf{P}_2 \mid \mathbf{P}_1) \\ \mathbf{P} \mid \mathbf{1}_{\mathcal{P}} = \mathbf{P} \end{array}$$

De plus, le processus $\mathbf{\Phi} \in \mathcal{P}_A$ défini comme l'ensemble vide des flots sur A est absorbant :

$$\mathbf{P} \mid \mathbf{\Phi} = \mathbf{\Phi}$$

Enfin, l'opérateur $\mid$ est idempotent :

$$\mathbf{P} \mid \mathbf{P} = \mathbf{P}$$

6.5.4.5 Vérification de propriétés

Nous démontrons deux résultats que nous allons utiliser dans la suite, notamment dans la preuve de propriétés sur des relations affines. Étant donnés deux processus $\mathbf{P} \in \mathcal{P}_{A_1}$ et $\mathbf{Q} \in \mathcal{P}_{A_2}$, avec $A_2 \subseteq A_1$, le lemme suivant donne une condition nécessaire et suffisante pour que le processus $\mathbf{Q}$ exprime une propriété vraie dans tous les flots de $\mathbf{P}$ et en conséquence que toute propriété vraie dans $\mathbf{Q}$ soit valide dans $\mathbf{P}$.

Lemme 1 *Considérons les processus* $\mathbf{P} \in \mathcal{P}_{A_1}$ *et* $\mathbf{Q} \in \mathcal{P}_{A_2}$ *tel que* $A_2 \subseteq A_1$.
Alors :

$$\mathbf{P} \mid \mathbf{Q} = \mathbf{P} \quad \Leftrightarrow \quad \Pi_{A_2}(\mathbf{P}) \subseteq \mathbf{Q}$$

Preuve • On suppose $\mathbf{P} \mid \mathbf{Q} = \mathbf{P}$ et on démontre $\Pi_{A_2}(\mathbf{P}) \subseteq \mathbf{Q}$.

$$\begin{array}{lcl} \mathbf{P} \mid \mathbf{Q} = \mathbf{P} & \Rightarrow & \forall\ F \in \mathbf{P}, \exists\ f_1, f_2 \text{ densifications telles que } F \text{ s'écrit} \\ & & F = (\ f_1 \uparrow \Pi_{A_1}(F)\ ) \cdot (\ f_2 \uparrow \Pi_{A_2}(F)\ ) \\ & & \text{et } \Pi_{A_1}(F) \in \mathbf{P}, \Pi_{A_2}(F) \in \mathbf{Q} \\ & \Rightarrow & \forall\ F \in \mathbf{P}, \Pi_{A_2}(F) \in \mathbf{Q} \\ & \Rightarrow & \Pi_{A_2}(\mathbf{P}) \subseteq \mathbf{Q} \end{array}$$

• On suppose $\Pi_{A_2}(\mathbf{P}) \subseteq \mathbf{Q}$ et on démontre $\mathbf{P} \mid \mathbf{Q} = \mathbf{P}$.
$\forall\ F \in \mathbf{P} \mid \mathbf{Q}$, $\exists\ f_1, f_2$ densifications telles que F s'écrit

$$F = (\ f_1 \uparrow \Pi_{A_1}(F)\ ) \cdot (\ f_2 \uparrow \Pi_{A_2}(F)\ )$$

et $\Pi_{A_1}(F) \in \mathbf{P}, \Pi_{A_2}(F) \in \mathbf{Q}$.

En décomposant le premier terme de définition du flot F ci-dessus correspondant aux ensembles A_2 et $A_1 \setminus A_2$ et en utilisant le lemme de recomposition de flots, on obtient :

$$\begin{array}{rcl} F & = & (\ g_1 \uparrow \Pi_{A_1 \setminus A_2}(F)\ ) \cdot (\ g_2 \uparrow \Pi_{A_2}(F)\ ) \cdot (\ f_2 \uparrow \Pi_{A_2}(F)\ ) \\ & = & (\ g_1 \uparrow \Pi_{A_1 \setminus A_2}(F)\ ) \cdot (\ g_2 \uparrow \Pi_{A_2}(F)\ ) \\ & = & (\ f_1 \uparrow \Pi_{A_1}(F)\ ) = \Pi_{A_1}(F) \in \mathbf{P} \end{array}$$

On a obtenu :

$$(\ \forall F,\ F \in \mathbf{P} |\ \mathbf{Q} \ \Rightarrow\ F \in \mathbf{P}\ ) \quad \Leftrightarrow \quad (\ \mathbf{P} |\ \mathbf{Q} \subseteq \mathbf{P}\ )$$

Soit $F \in \mathbf{P}$ un flot arbitraire. Cf. au lemme de recomposition de flots, il existe f_1, f_2 densifications telles que F s'écrit :

$$F = (\ f_1 \uparrow \Pi_{A_1}(F)\ ) \cdot (\ f_2 \uparrow \Pi_{A_2}(F)\ )$$

et $\Pi_{A_1}(F) \in \mathbf{P}$, $\Pi_{A_2}(\mathbf{P}) \subseteq \mathbf{Q} \ \Rightarrow\ \Pi_{A_2}(F) \in \mathbf{Q}$. Il résulte $F \in \mathbf{P} |\ \mathbf{Q}$, i.e. $\mathbf{P} \subseteq \mathbf{P} |\ \mathbf{Q}$.

□

A partir de ce résultat nous pouvons démontrer le théorème suivant :

Théorème 1 *Considérons les processus $\mathbf{P} \in \mathcal{P}_{A_1}$ et $\mathbf{Q} \in \mathcal{P}_{A_2}$ tel que $A_2 \subseteq A_1$.*
Alors :

$$\Pi_{A_2}(\mathbf{P}) = \mathbf{Q} \ \Rightarrow\ \forall\ \mathbf{R} \in \mathcal{P}_{A_2},\ \left\{ \begin{array}{ll} a) & \mathbf{Q} |\ \mathbf{R} = \mathbf{Q} \ \Rightarrow\ \mathbf{P} |\ \mathbf{R} = \mathbf{P} \\ b) & \mathbf{P} |\ \mathbf{R} = \mathbf{P} \ \Rightarrow\ \mathbf{Q} |\ \mathbf{R} = \mathbf{Q} \end{array} \right.$$

Preuve Comme $\Pi_{A_2}(\mathbf{P}) = \mathbf{Q}$, il résulte $\mathbf{P} |\ \mathbf{Q} = \mathbf{P}$ et on obtient :

$$\begin{array}{l} a)\ \mathbf{Q} |\ \mathbf{R} = \mathbf{Q} \ \Rightarrow\ \mathbf{P} |\ \mathbf{R} = (\ \mathbf{P} |\ \mathbf{Q}\ ) |\ \mathbf{R} = \mathbf{P} |\ (\ \mathbf{Q} |\ \mathbf{R}\ ) = \mathbf{P} |\ \mathbf{Q} = \mathbf{P} \\ b)\ \mathbf{P} |\ \mathbf{R} = \mathbf{P} \ \Rightarrow\ \Pi_{A_2}(\mathbf{P}) \subseteq \mathbf{R} \ \Rightarrow\ \Pi_{A_2}(\mathbf{P}) |\ \mathbf{R} = \Pi_{A_2}(\mathbf{P}) \ \Rightarrow\ \mathbf{Q} |\ \mathbf{R} = \mathbf{Q} \end{array}$$

□

Étant donnés les processus $\mathbf{P} \in \mathcal{P}_{A_1}$, $\mathbf{Q} \in \mathcal{P}_{A_2}$ tels que $\Pi_{A_2}(\mathbf{P}) = \mathbf{Q}$, avec$A_2 \subseteq A_1$, le théorème précédent permet d'affirmer que toute propriété valide dans $\mathbf{Q}$ pour l'ensemble de variables A_2 reste vraie dans $\mathbf{P}$ et vice versa, toute propriété vraie dans $\mathbf{P}$ pour l'ensemble de variables A_2 reste vraie dans $\mathbf{Q}$. En effet, il suffit d'exprimer une propriété sous la forme d'un processus $\mathbf{R} \in \mathcal{P}_{A_2}$, et dans ce cas $\mathbf{Q} |\ \mathbf{R} = \mathbf{Q}$ et $\mathbf{P} |\ \mathbf{R} = \mathbf{P}$ exprime le fait que la propriété $\mathbf{R}$ est valide dans $\mathbf{P}$ et $\mathbf{Q}$.

6.6 Conclusion

Tout processus SIGNAL – que ce soit un opérateur primitif ou un processus plus complexe obtenu par composition ou restriction à partir des opérateurs primitifs – est un ensemble de flots dans lesquels les valeurs des signaux et leurs instants de présence/absence respectifs (horloges) satisfont un certain nombre de contraintes. Ces contraintes spécifient de façon implicite le comportement que l'utilisateur attend d'un programme (et du processus correspondant) en SIGNAL et se traduisent dans des propriétés que le programme doit vérifier. Afin de garantir des programmes corrects du point de vue des propriétés exprimées et pour réduire l'overhead d'une vérification à l'exécution, l'environnement actuel de SIGNAL propose une vérification statique des contraintes sur les flots d'un processus. Il s'agit en fait de vérifier qu'il existe des flots avec les contraintes spécifiées et de donner une définition explicite de ces flots en précisant, à chaque instant, la présence ou l'absence des signaux impliqués et leurs valeurs. Pour cela, les contraintes de tout processus SIGNAL ont été exprimées sous la forme d'un système d'équations sur les valeurs et sur les horloges des signaux du processus et une méthode de résolution, basée sur les propriétés de ce système, a été mise en place.

Nous allons nous intéresser plus particulièrement à la résolution du système d'équations sur horloges ou *calcul d'horloges*. Celui-ci est basé sur un ensemble de règles d'identifiabilité d'horloges qui précisent dans quelles conditions les horloges $h(x)$ et $h(y)$ de deux variables (signaux) x et y sont identifiables, c'est-à-dire

$$h(x) \overset{\mathbf{P}}{\ominus} h(y)$$

Comme

$$x \overset{\mathbf{P}}{\odot} y \quad \Leftrightarrow \quad h(x) \overset{\mathbf{P}}{\ominus} h(y)$$

les résultats sur l'identifiabilité des horloges peuvent être appliqués pour décider de la synchronisabilité des signaux x et y ; autrement dit, dans certaines conditions, on peut vérifier si les signaux x et y peuvent être synchronisés.

Le calcul d'horloges actuel – qui est un calcul formel sur les synchronisations de signaux – ne considère pas les horloges des signaux en relation affine ou les expressions construites avec ces horloges ; fournir le calcul formel correspondant constitue l'objet principal du chapitre suivant de ce document. Nous nous proposons d'accroître la puissance du calcul d'horloges existant en enrichissant le système de règles d'identifiabilité (synchronisabilité[6]) sur lequel celui-ci est basé. Dans ce but, nous étudions les propriétés des relations affines dans le modèle sémantique des traces et déduisons un système de règles d'identifiabilité de variables de même type qui sont appliquées directement aux variables d'horloges.

6. Deux horloges synchronisables sont identifiables et viceversa.

Chapitre 7

Les relations affines dans le modèle des suites

Nous avons introduit en début de cette partie le concept de transformation affine d'horloges en SIGNAL. Toute transformation affine de paramètres (n, φ, d) induit une relation affine de paramètres identiques entre les horloges opérandes. Dans ce chapitre, nous proposons une étude de propriétés dans l'espace des relations affines; ces propriétés seront utilisées pour décrire des nouvelles règles de synchronisabilité d'horloges qui vont accroître la puissance du système formel sur lequel est basé le calcul d'horloges existant.

Le calcul d'horloges peut être assimilé à un calcul formel sur les synchronisations des variables SIGNAL (signaux). En partant des propriétés de l'espace des horloges d'un programme SIGNAL, il s'agit d'étudier si deux variables x et y peuvent être synchronisées, c'est-à-dire si leur horloges peuvent être identifiées (cela en vertu de la propriété que deux variables x et y sont synchronisables si et seulement si leurs horloges sont identifiables). Comme l'identifiabilité d'horloges est équivalente à leur synchronisabilité, on peut parler indifféremment d'idenfiabilité ou de synchronisabilité d'horloges. Il n'existe pas actuellement un calcul formel sur les synchronisations des horloges en relation affine ou des expressions construites par intersection, union et/ou différence d'horloges en relation affine.

Dans ce chapitre, nous proposons un tel calcul, plus particulièrement nous étudions l'identifiabilité de variables de mêmes domaines de valeurs (mêmes types) entre lesquelles il existe des relations affines ou des relations qui peuvent être des expressions sur des relations affines. Les tests d'identifiabilité que nous proposons sont basés sur des tests d'équivalence de relations affines et des opérations de composition, union, intersection et différence de relations affines définies au niveau des flots et étendues ensuite au niveau des processus. Toutes les règles d'identifiabilité obtenues peuvent être appliquées ensuite pour les horloges, qui sont des variables de même domaine de valeurs. A plus long terme, notre étude peut constituer le point de départ d'une étude d'identifiabilité plus générale – pour des variables de types différents et pour des fonctions de densification plus générales – comme nous allons le préciser dans les conclusions de notre travail.

En conséquence, nous allons commencer notre présentation en introduisant, en section 7.1, l'espace des relations affines associées aux flots. Dans cet espace, nous étudions l'égalité de relations affines et nous définissons des opérations de composition, union, intersection et différence de relations affines. Ces propriétés sont reprises dans le contexte des relations affines associées aux processus dans la section 7.2. Ensuite, dans la section 7.3, nous introduisons de manière incrémentale notre nouveau système de règles d'identifiabilité de variables arbitraires qui deviennent des règles de synchronisabilité dans le cas de variables d'horloges ; dans certaines conditions, le test de synchronisabilité de deux expressions sur horloges en relation affine se fait par l'intermédiaire d'une *forme normale relative* unique. Le nouveau système de règles de synchronisabilité constituera le support pour la définition du calcul *affine* d'horloges d'après le modèle du calcul *booléen* existant. En section 7.4, nous montrons comment le concept de synchronisabilité peut se retrouver dans le calcul d'horloges booléen et proposons un calcul *affine* d'horloges d'après le modèle du calcul existant et en utilisant les nouvelles règles de synchronisabilité. Pour cela, nous définissons sur l'espace des horloges en relation affine un ordre $\preceq_{aff}$ et proposons de représenter toute relation affine sous la forme d'un processus **P** dont les variables forment un $\vee$–semi-treillis d'ordre $\preceq_{aff}$. Nous concluons ce chapitre avec une vue générale du calcul formel exposé et un prélude sur son intégration dans le calcul d'horloges de SIGNAL, suivis d'une discussion sur d'autres perspectives d'utilisation de nos résultats formels et sur leur impact sur le développement ultérieur du langage.

7.1 Relations affines & Flots

Dans cette section nous introduisons formellement la relation affine associée à un flot F et fixons quelques unes de ses propriétés.

Nous fixons d'abord l'ensemble des variables A_1 utilisé dans la suite; on considère A_1 un sous-ensemble non vide de A tel que les domaines des valeurs des variables de A_1 sont identiques: $\forall\ x, y \in A_1, D_x^{\perp} = D_y^{\perp}$.

Soit $F \in \mathcal{F}_{A_1}$ un flot arbitraire dans $\mathbf{P} \in \mathcal{P}_{A_1}$ et $x, y \in A_1$. Nous introduisons d'abord la relation de (f_1, f_2)-codensification dans F entre les variables x et y de même type.

- $x\ \mathcal{R}^F_{(f_1,f_2)}\ y$

  Les variables x et y sont (f_1, f_2)-*codensifiées* dans le flot F ssi $\exists f_1, f_2 : \mathbb{N} \to \mathbb{N}$ fonctions de densification et $\exists g : \mathbb{N} \to \mathbb{N}$, croissante sur $\mathbb{N}$ et strictement croissante sur $f_1(\mathbb{N}) \cup f_2(\mathbb{N})$, telles que

  $$F = (\ (g \circ f_1) \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ (g \circ f_2) \uparrow \Pi_{\{y\}}(F)\ ) \cdot F$$

  et

  $$\forall\ t, s \in \mathbb{N},\ f_1(t) = f_2(s)\ \Rightarrow\ x_{(F)}(t) = y_{(F)}(s)$$

  Dans ce cas, on dit aussi qu'il existe une relation de codensification de paramètres (f_1, f_2) entre x et y.

t		0	1	2	3	4	5	6	7	
$f_1(t)$			$f_1(0)$		$f_1(1)$			$f_1(2)$	$f_1(3)$	....
$f_2(t)$		$f_2(0)$	$f_2(1)$			$f_2(2)$		$f_2(3)$	$f_2(4)$	....
F	x	$\perp$	x_1	$\perp$	x_2	$\perp$	$\perp$	x_3	$\perp$	....
	y	y_1	y_2	$\perp$	$\perp$	y_3	$\perp$	y_4	$\perp$	....
	z	z_1	z_2	z_3	z_4	z_5	z_6	z_7	$\perp$	....

FIG. 7.1 – *Les variables x et y sont (f_1, f_2)-codensifiées dans F.*

t		0	1	2	3	4	5	6	7	8
$f_1'(t)$			$f_1'(0)$	$f_1'(1)$		$f_1'(2)$	$f_1'(3)$	$f_1'(4)$	$f_1'(5)$	....
$f_2'(t)$		$f_2'(0)$	$f_2'(1)$		$f_2'(2)$	$f_2'(3)$	$f_2'(4)$	$f_2'(5)$	$f_2'(6)$	....
F'	x	$\perp$	x_1	x_2	$\perp$	x_3	$\perp$	$\perp$	$\perp$	....
	y	y_1	y_2	$\perp$	y_3	y_4	$\perp$	$\perp$	$\perp$	....
	v	v_1	v_2	v_3	v_4	v_5	v_6	v_7	$\perp$	....

FIG. 7.2 – *Les variables x et y sont (f_1', f_2')-codensifiées dans F'.*

Commentaire 1) Nous avons représenté sur la Figure 7.1 un flot $F \in \mathcal{F}_{\{x,y,z\}}$. Si f_1, f_2 sont des fonctions de densification définies par :

$$\begin{array}{l} f_1, f_2 : \mathbb{N} \rightarrow \mathbb{N} \\ f_1(0) = 1, f_1(1) = 3, f_1(2) = 6, f_1(t) = t + 4, \forall t \in \mathbb{N}, t \geq 3 \\ f_2(0) = 0, f_2(1) = 1, f_2(2) = 4, f_2(3) = 6, f_2(t) = t + 3, \forall t \in \mathbb{N}, t \geq 4 \end{array}$$

et $x_1 = y_2$, $x_3 = y_4$, alors les variables x et y sont (f_1, f_2)-codensifiées dans le flot F, avec g la fonction identité sur $\mathbb{N}(g = id_{\mathbb{N}})$.

2) Afin de donner l'intuition qui se trouve derrière la fonction g de la définition formelle précédente, nous considérons le flot F' représenté sur la Figure 7.2. Les densifications f_1', f_2' sont définies comme suit :

$$\begin{array}{l} f_1', f_2' : \mathbb{N} \rightarrow \mathbb{N} \\ f_1'(0) = 1, f_1'(1) = 2, f_1'(2) = 4, f_1'(t) = t + 2, \forall t \in \mathbb{N}, t \geq 3 \\ f_2'(0) = 0, f_2'(1) = 1, f_2'(2) = 3, f_2'(3) = 4, f_2'(t) = t + 1, \forall t \in \mathbb{N}, t \geq 4 \end{array}$$

Les suites de valeurs des variables x et y vérifient $x_1 = y_2$, $x_3 = y_4$. Conformément à la définition, x et y sont (f_1', f_2')-codensifiées dans F'. Or, on remarque que le flot F' exhibe le même positionnement *relatif* des variables x et y que le flot F considéré précédemment. Cette propriété peut être exprimée en réécrivant le flot F' comme :

$$F' \quad = \quad (\ (g \circ f_1) \uparrow \Pi_{\{x\}}(F')\ ) \cdot (\ (g \circ f_2) \uparrow \Pi_{\{y\}}(F')\ ) \cdot F'$$

avec g définie par :

$$\begin{array}{l} g : \mathbb{N} \rightarrow \mathbb{N} \\ g(0) = 0, g(1) = 1, g(2) = g(3) = 2, g(4) = 3, g(5) = g(6) = 4, g(t) = t - 2, \forall t \in \mathbb{N}, t \geq 7 \end{array}$$

La fonction g est croissante sur $\mathbb{N}$ et strictement croissante sur $f_1(\mathbb{N}) \cup f_2(\mathbb{N})$. Aussi, les implications suivantes sont vérifiées :

$$\begin{array}{rcl} \forall\ t, s \in \mathbb{N},\ f_1(t) = f_2(s) & \Rightarrow & (g \circ f_1)(t) = (g \circ f_2)(s) \\ & \Rightarrow & f_1'(t) = f_2'(s) \\ & \Rightarrow & x_{(F')}(t) = y_{(F')}(s) \end{array}$$

La motivation de

$$(g \circ f_1)(t) = (g \circ f_2)(s) \quad \Rightarrow \quad f_1'(t) = f_2'(s)$$

vient du fait que x et y sont (f'_1, f'_2)-codensifiées dans F', donc F' s'écrit

$$F' \;=\; (\; f'_1 \uparrow \Pi_{\{x\}}(F')\;) \cdot (\; f'_2 \uparrow \Pi_{\{y\}}(F')\;) \cdot F'$$

et les fonctions $g \circ f_1$, f'_1 et $g \circ f_2$, f'_2 sont égales par le lemme de recomposition de flots.

Finalement

$$\forall\; t, s \in \mathbb{N},\; f'_1(t) = f'_2(s) \quad \Rightarrow \quad x_{(F')}(t) = y_{(F')}(s)$$

puisque x, y sont (f'_1, f'_2)-codensifiées dans F'.

En conclusion, x et y sont aussi (f_1, f_2)-codensifiées dans F' : l'usage de la fonction g a permis d'exprimer dans F' le même positionnement relatif des variables x et y que celui exhibé par le flot F. Exprimer la relation de (f'_1, f'_2)-codensification entre x et y dans F' aussi comme une relation de (f_1, f_2)-codensification montre que les deux relations captent en fait la même position *relative* entre x et y indifféremment des positions absolues de x et y dans les deux flots.

3) Évidemment, si F' exhibe le positionnement relatif des variables x et y dans F, la réciproque est valable, donc x et y sont (f'_1, f'_2)-codensifiées dans F. Cette propriété peut être exprimée en réécrivant le flot F comme :

$$F \;=\; (\; (g' \circ f'_1) \uparrow \Pi_{\{x\}}(F)\;) \cdot (\; (g' \circ f'_2) \uparrow \Pi_{\{y\}}(F)\;) \cdot F$$

avec g' définie par :

$$\begin{array}{l} g' : \mathbb{N} \;\rightarrow\; \mathbb{N} \\ g'(0) = 0, g'(1) = 1, g'(2) = 3, g'(3) = 4, g'(4) = 6, g'(t) = t + 2, \forall t \in \mathbb{N}, t \geq 5 \end{array}$$

La fonction g' est strictement croissante sur $f'_1(\mathbb{N}) \cup f'_2(\mathbb{N})$ (puisque strictement croissante sur $\mathbb{N}$ par définition). De manière similaire au cas 2) précédent, on peut démontrer les implications suivantes :

$$\begin{array}{rcl} \forall\; t, s \in \mathbb{N},\; f'_1(t) = f'_2(s) & \Rightarrow & (g' \circ f'_1)(t) = (g' \circ f'_2)(s) \\ & \Rightarrow & f_1(t) = f_2(s) \\ & \Rightarrow & x_{(F)}(t) = y_{(F)}(s) \end{array}$$

et, en conclusion, x et y sont (f'_1, f'_2)-codensifiées dans F. Les Figures 7.3 et 7.4 illustrent le rôle des fonctions g et g' comme expliqué ci-dessus. □

t		0	1	2	3	4	5	6	7	
$f_1(t)$			$f_1(0)$		$f_1(1)$			$f_1(2)$	$f_1(3)$	....
$f_2(t)$		$f_2(0)$	$f_2(1)$			$f_2(2)$		$f_2(3)$	$f_2(4)$	....
$(g' o f_1')(t)$			$g'(f_1'(0))$		$g'(f_1'(1))$			$g'(f_1'(2))$	$g'(f_1'(3))$	....
$(g' o f_2')(t)$		$g'(f_2'(0))$	$g'(f_2'(1))$			$g'(f_2'(2))$		$g'(f_2'(3))$	$g'(f_2'(4))$	....
F	x	$\perp$	x_1	$\perp$	x_2	$\perp$	$\perp$	x_3	$\perp$	....
	y	y_1	y_2	$\perp$	$\perp$	y_3	$\perp$	y_4	$\perp$	....
	z	z_1	z_2	z_3	z_4	z_5	z_6	z_7	$\perp$	....

FIG. 7.3 – *x et y sont (f_1, f_2)- et (f'_1, f'_2)-codensifiées dans F.*

t		0	1	2	3	4	5	6	7	8
$f_1'(t)$			$f_1'(0)$	$f_1'(1)$		$f_1'(2)$	$f_1'(3)$	$f_1'(4)$	$f_1'(5)$	$\dots$
$f_2'(t)$		$f_2'(0)$	$f_2'(1)$		$f_2'(2)$	$f_2'(3)$	$f_2'(4)$	$f_2'(5)$	$f_2'(6)$	$\dots$
$(g \circ f_1)(t)$			$g(f_1(0))$	$g(f_1(1))$		$g(f_1(2))$	$g(f_1(3))$	$g(f_1(4))$	$g(f_1(5))$	$\dots$
$(g \circ f_2)(t)$		$g(f_2(0))$	$g(f_2(1))$		$g(f_2(2))$	$g(f_2(3))$	$g(f_2(4))$	$g(f_2(5))$	$g(f_2(6))$	$\dots$
F'	x	$\perp$	x_1	x_2	$\perp$	x_3	$\perp$	$\perp$	$\perp$	$\dots$
	y	y_1	y_2	$\perp$	y_3	y_4	$\perp$	$\perp$	$\perp$	$\dots$
	v	v_1	v_2	v_3	v_4	v_5	v_6	v_7	$\perp$	$\dots$

FIG. 7.4 – *x et y sont (f_1, f_2)- et (f_1', f_2')-codensifiées dans F'.*

- Pour tout flot $F \in \mathcal{F}_{A_1}$, *la relation de (f_1, f_2)-codensification* $\mathcal{R}^F_{(f_1,f_2)}$ représente l'ensemble des couples $(x, y) \in A_1 \times A_1$ tels que $x \ \mathcal{R}^F_{(f_1,f_2)} \ y$ [San80]) :

$$\mathcal{R}^F_{(f_1,f_2)} = \{(x, y) \in A_1 \times A_1 \ / \ \ x \ \mathcal{R}^F_{(f_1,f_2)} \ y \ \}$$

Remarques :

1) Il existe toujours $F \in \mathcal{F}_{\{x,y\}}$ tel que $x \ \mathcal{R}^F_{(f_1,f_2)} \ y$, pour toute paire de densifications (f_1, f_2), i.e. le processus défini comme l'ensemble des flots sur $\{x, y\}$ dans lesquels x et y sont (f_1, f_2)-codensifiées est non vide :

$$\mathbf{P} = \{F \in \mathcal{F}_{\{x,y\}} \ | \ \ x \ \mathcal{R}^F_{(f_1,f_2)} \ y \ \} \neq \emptyset$$

Cette propriété, facile à vérifier par des exemples, permet d'affirmer qu'il existe des flots F tels que $\mathcal{R}^F_{(f_1,f_2)} \neq \emptyset$ et donc il est intéressant d'étudier les propriétés de la relation *non vide* $\mathcal{R}^F_{(f_1,f_2)}$.

2) La notation $x \ \mathcal{R}^F_{(f_1,f_2)} \ y$ suppose une orientation de la relation $\mathcal{R}^F_{(f_1,f_2)}$ afin que f_1 soit associée à x et f_2 à y ; en ce sens, on a généralement $\mathcal{R}^F_{(f_1,f_2)} \neq \mathcal{R}^F_{(f_2,f_1)}$. $\square$

Pour des fonctions de densification f_1, f_2 affines, la relation de (f_1, f_2)-codensification devient une *relation affine* ; nous présentons la définition en section 7.1.1. Parmi les propriétés générales des relations affines présentées en section 7.1.2, nous retrouvons une *forme normale* pour toute relation affine et la définition des fonctions d'occurrence relatives de deux variables en relation affine. Tout calcul ultérieur sera réalisé sur des relations affines en forme normale. Nous définissons l'équivalence de relations affines en section 7.1.3 – en utilisant les expressions des fonctions d'occurrence déduites au préalable – et proposons une forme canonique de toute relation affine par rapport à la relation d'équivalence (section 7.1.4). Finalement, en section 7.1.5, nous proposons un ensemble d'opérations sur relations affines – inverse, composition, union, intersection, différence – et présentons une *forme normale relative* d'expressions sur relations affines construites avec les opérateurs d'union, intersection et différence. Toutes ces propriétés, étendues aux relations affines associées aux processus, seront utilisées pour élargir le système de règles de synchronisabilité sur lequel repose le calcul d'horloges actuel.

7.1.1 La relation affine - définition

Si les fonctions de densification f_1 et f_2 sont affines, la relation de (f_1, f_2)-codensification est une relation affine.

- $x \; \mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))} \; y$

  Les variables x et y sont *affine-codensifiées* dans le flot F ssi elles sont (f_1, f_2)-codensifiées dans F, avec $f_1 = \lambda \; t.(nt + \varphi_1)$, $f_2 = \lambda \; t.(dt + \varphi_2)$ et[1] $n, d, \in \mathbb{N}^*$, $\varphi_1, \varphi_2 \in \mathbb{N}$.

  Dans ce cas, on dit aussi qu'il existe entre x et y une relation (de codensification) affine de paramètres $((n, \varphi_1), (d, \varphi_2))$.

		0	1	2	3	4	5	6	7	8	9	10
t		0	1	2	3	4	5	6	7	8	9	10
$f_1(t)$		0		2		4		6		8		10
$f_2(t)$			1				5				9	
$(g' \circ f_1')(t)$		$g'(0)$		$g'(3)$		$g'(6)$		$g'(9)$		$g'(12)$		$g'(15)$
$(g' \circ f_2')(t)$			$g'(2)$				$g'(8)$				$g'(14)$	
F	x	x_1	$\perp$	x_2	$\perp$	x_3	$\perp$	x_4	$\perp$	x_5	$\perp$	x_6
	y	$\perp$	y_1	$\perp$	$\perp$	$\perp$	y_2	$\perp$	$\perp$	$\perp$	y_3	$\perp$
	z	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8	z_9	z_{10}	z_{11}

FIG. 7.5 – *Illustration de la relation de affine-codensification (1).*

t		0	1	2	3	4	5	6	7	8	9	10	11	
$f_1'(t)$		0			3			6			9			
$f_2'(t)$				2						8				
$(g \circ f_1)(t)$		$g(0)$			$g(2)$			$g(4)$			$g(6)$			
$(g \circ f_2)(t)$				$g(1)$						$g(5)$				
F'	x	x_1	$\perp$	$\perp$	x_2	$\perp$	$\perp$	x_3	$\perp$	$\perp$	x_4	$\perp$	$\perp$	....
	y	$\perp$	$\perp$	y_1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	y_2	$\perp$	$\perp$	$\perp$	....
	v	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	....

FIG. 7.6 – *Illustration de la relation de affine-codensification (2).*

Commentaire D'après la définition de la relation de (f_1, f_2)-codensification, si x et y sont affine-codensifiées dans F, il existe $g : \mathbb{N} \rightarrow \mathbb{N}$, croissante sur $\mathbb{N}$ et strictement croissante sur $f_1(\mathbb{N}) \cup f_2(\mathbb{N})$,

1. $\mathbb{N}^* = \mathbb{N} \setminus \{0\}$

telle que F s'écrit

$$F = (\ (g \circ f_1) \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ (g \circ f_2) \uparrow \Pi_{\{y\}}(F)\ ) \cdot F$$

et

$$\forall\ t, s \in \mathbb{N},\ f_1(t) = f_2(s)\ \Rightarrow\ x_{(F)}(t) = y_{(F)}(s)$$

avec $f_1, f_2 : \mathbb{N} \ \rightarrow\ \mathbb{N}$, $f_1 = \lambda\ t.(nt + \varphi_1)$ et $f_2 = \lambda\ t.(dt + \varphi_2)$.

En section 5.1 nous avons présenté les motivations principales de l'existence de la fonction g ci-dessus. Intuitivement, g permet de décrire le positionnement *relatif* des variables x et y dans le flot F de la manière dont il se retrouve dans d'autres flots et afin de souligner le fait qu'il s'agit du même type de positionnement. C'est le cas pour les flots F et F' représentés sur les Figures 7.5 et 7.6 qui peuvent s'écrire (les contraintes sur les suites des valeurs de $x_{(F)}$ et $y_{(F)}$ sont implicites)

$$F = (\ f_1 \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ f_2 \uparrow \Pi_{\{y\}}(F)\ ) \cdot F$$
$$F = (\ (g' \circ f'_1) \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ (g' \circ f'_2) \uparrow \Pi_{\{y\}}(F)\ ) \cdot F$$

et respectivement

$$F' = (\ f'_1 \uparrow \Pi_{\{x\}}(F')\ ) \cdot (\ f'_2 \uparrow \Pi_{\{y\}}(F')\ ) \cdot F'$$
$$F' = (\ (g \circ f_1) \uparrow \Pi_{\{x\}}(F')\ ) \cdot (\ (g \circ f_2) \uparrow \Pi_{\{y\}}(F')\ ) \cdot F'$$

où les fonctions $f_1, f_2, f'_1, f'_2, g, g'$ sont définies comme suit :

$$\begin{array}{l}
f_1, f_2, f'_1, f'_2, g, g' : \mathbb{N} \ \rightarrow\ \mathbb{N} \\
f_1 = \lambda\ t.(2t) \quad\ f_2 = \lambda\ t.(4t+1), \quad\ f'_1 = \lambda\ t.(3t) \quad\ f'_2 = \lambda\ t.(6t+2) \\
g' \text{ croissante sur } \mathbb{N},\ \ g'(3t) = 2t,\ g'(6t+2) = 4t+1,\ \ \forall t \in \mathbb{N} \\
g \text{ croissante sur } \mathbb{N},\ \ g(2t) = 3t,\ g(4t+1) = 6t+2,\ \ \forall t \in \mathbb{N}
\end{array}$$

□

- Pour tout flot $F \in \mathcal{F}_{A_1}$, *la relation affine* $\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))}$ représente l'ensemble des couples $(x, y) \in A_1 \times A_1$, $x\ \mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))}\ y$:

$$\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))} = \{(x,y) \in A_1 \times A_1\ /\ \ x\ \mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))}\ y\ \ \}$$

Remarques :

1) Pour tout ensemble de valeurs pour les paramètres $n, d, \in \mathbb{N}^*$, $\varphi_1, \varphi_2 \in \mathbb{N}$, il existe toujours $F \in \mathcal{F}_{\{x,y\}}$ tel que $x\ \mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))}\ y$, c'est-à-dire :

$$\mathbf{P} = \{F \in \mathcal{F}_{\{x,y\}}\ |\ \ x\ \mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))}\ y\ \} \neq \emptyset$$

En effet, il existe des flots tels que $\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))}$ soit non vide et notre étude de propriétés sur les relations affines se déroule dans ce contexte.

2) En général $\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))} \neq \mathcal{R}^F_{((d,\varphi_2),(n,\varphi_1))}$. □

7.1.2 Propriétés générales des relations affines

Nous considérons $F \in \mathcal{F}_{A_1}$ arbitraire tel que toute relation affine $\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))}$ définie par la suite soit non vide, avec n, d, φ_1, φ_2 satisfaisant les conditions : $n, d \in \mathbb{N}^*, \varphi_1, \varphi_2 \in \mathbb{N}$.

7.1.2.1 Normalisation

Les propriétés de $\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))}$ dépendent des valeurs de ses paramètres $((n,\varphi_1),(d,\varphi_2))$. Nous proposons une notation normalisée de toute relation affine sous la forme $\mathcal{R}^F_{(n,\varphi,d)}$, avec $\varphi = \varphi_2 - \varphi_1$. Les propriétés déduites ensuite pour une relation affine $\mathcal{R}^F_{(n,\varphi,d)}$ seront valides pour toute relation $\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))}$, avec $\varphi_1, \varphi_2 \in \mathbb{N}, \varphi_2 - \varphi_1 = \varphi$.

Nous énonçons d'abord quelques résultats intermédiaires qui induisent directement le corollaire de normalisation.

Lemme 2 *(Lemme de translation)*

$$\forall\ c \in \mathbb{Z}\ \ \textit{tel que}\ \ \varphi_1 + c \geq 0\ \ \textit{et}\ \ \varphi_2 + c \geq 0,\ \ \textit{on a}$$

$$\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))} = \mathcal{R}^F_{((n,\varphi_1+c),(d,\varphi_2+c))}$$

Preuve • On prouve d'abord l'inclusion directe

$$\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))} \subseteq \mathcal{R}^F_{((n,\varphi_1+c),(d,\varphi_2+c))}$$

Soit $(x,y) \in \mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))}$, i.e. $(x,y) \in A_1 \times A_1$ et $x\ \mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))}\ y$.

Conformément à la définition d'une relation affine (section 7.1.1), $\exists\ g : \mathbb{N} \to \mathbb{N}$, croissante sur $\mathbb{N}$ et strictement croissante sur $f_1(\mathbb{N}) \cup f_2(\mathbb{N})$, telle que

$$F = (\ (g \circ f_1) \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ (g \circ f_2) \uparrow \Pi_{\{y\}}(F)\ ) \cdot F \tag{7.1}$$

et

$$\forall\ t,s \in \mathbb{N},\ f_1(t) = f_2(s)\ \Rightarrow\ x_{(F)}(t) = y_{(F)}(s) \tag{7.2}$$

avec f_1, f_2 densifications de la forme $f_1 = \lambda\ t.(nt + \varphi_1)$ et $f_2 = \lambda\ t.(dt + \varphi_2)$.

Soit la fonction g' suivante:

$$g' : \mathbb{N} \to \mathbb{N},$$
$$g'(t) = \begin{cases} g(0), & \forall t < c \\ g(t-c), & \forall t \geq c \end{cases}$$

On va démontrer que g' satisfait les conditions 7.1 et 7.2 ci-dessus pour les fonctions de densification $f_1' = \lambda\ t.(nt + \varphi_1 + c)$ et $f_2' = \lambda\ t.(dt + \varphi_2 + c)$.

1) On prouve d'abord que g' est croissante sur $\mathbb{N}$.

Soit $t_1, t_2 \in \mathbb{N}, t_1 < t_2$.

- Cas 1 : $t_2 < c \Rightarrow t_1 < c \Rightarrow g'(t_1) = g'(t_2) = g(0)$.
- Cas 2 : $t_1 < c \leq t_2 \Rightarrow g'(t_1) = g(0)$ et $g'(t_2) = g(t_2 - c)$.

$$\left.\begin{array}{l} t_2 \geq c \Leftrightarrow t_2 - c \geq 0 \\ g \text{ croissante sur } \mathbb{N} \end{array}\right\} \Rightarrow g(t_2 - c) \geq g(0) \Rightarrow g'(t_2) \geq g'(t_1)$$

- Cas 3 : $c \leq t_1 < t_2 \Rightarrow g'(t_1) = g(t_1 - c)$ et $g'(t_2) = g(t_2 - c)$.

$$\left.\begin{array}{l} c \leq t_1 < t_2 \Leftrightarrow 0 \leq t_1 - c < t_2 - c \\ g \text{ croissante sur } \mathbb{N} \end{array}\right\} \begin{array}{l} \Rightarrow\ \ g(t_2 - c) \geq g(t_1 - c) \\ \Rightarrow\ \ g'(t_2) \geq g'(t_1) \end{array}$$

En conclusion : $\forall t_1, t_2 \in \mathbb{N}, t_1 < t_2 \Rightarrow g'(t_1) \leq g'(t_2)$. Il résulte que g' est croissante sur $\mathbb{N}$.

2) On prouve maintenant que g' est strictement croissante sur $f'_1(\mathbb{N}) \cup f'_2(\mathbb{N})$.

Tout d'abord, on a pour tout $i \in \{1,2\}$ et pour tout $t \in \mathbb{N}$:

$$\begin{aligned} f'_i(t) = f_i(t) + c \quad &\Rightarrow \quad f'_i(t) \geq c \\ &\Rightarrow \quad g'(f'_i(t)) = g'(f_i(t) + c) = g(f_i(t) + c - c) = g(f_i(t)) \end{aligned}$$

Il résulte $g' \circ f'_i = g \circ f_i, \forall i \in \{1,2\}$.

De plus, $f_i : \mathbb{N} \to \mathbb{N}$ strictement croissante sur $\mathbb{N}$, pour tout $i \in \{1,2\}$, et g strictement croissante sur $f_1(\mathbb{N}) \cup f_2(\mathbb{N})$ impliquent $g \circ f_i : \mathbb{N} \to \mathbb{N}$ strictement croissantes pour tout $i \in \{1,2\}$.

En conclusion, $g' \circ f'_i$ est strictement croissante pour tout $i \in \{1,2\}$, ce qui implique g' strictement croissante sur $f'_1(\mathbb{N})$ et strictement croissante sur $f'_2(\mathbb{N})$.

Il est nécessaire de prouver g' strictement croissante sur $f'_1(\mathbb{N}) \cup f'_2(\mathbb{N})$.

Soit $t_1 \in f'_1(\mathbb{N}), t_2 \in f'_2(\mathbb{N}), t_1 < t_2$. Il existe $s_1, s_2 \in \mathbb{N}$ tels que $t_1 = f'_1(s_1), t_2 = f'_2(s_2)$ et $f'_1(s_1) < f'_2(s_2)$.

$$\begin{aligned} f'_1(s_1) < f'_2(s_2) \quad &\Rightarrow \quad f_1(s_1) + c < f_2(s_2) + c \quad \Rightarrow \quad f_1(s_1) < f_2(s_2) \\ &\overset{\substack{g \text{ str. cr. sur} \\ f_1(\mathbb{N}) \cup f_2(\mathbb{N})}}{\Rightarrow} \quad (g \circ f_1)(s_1) < (g \circ f_2)(s_2) \quad \Rightarrow \quad (g' \circ f'_1)(s_1) < (g' \circ f'_2)(s_2) \\ &\Rightarrow \quad g'(t_1) < g'(t_2) \end{aligned}$$

Pour le cas $t_1 < t_2, t_1 \in f'_2(\mathbb{N}), t_2 \in f'_1(\mathbb{N})$, il résulte comme précédemment que $g'(t_1) < g'(t_2)$.

En conclusion, g' est strictement croissante sur $f'_1(\mathbb{N}) \cup f'_2(\mathbb{N})$.

3) Par définition, F s'écrit :

$$F \quad = \quad (\ (g \circ f_1) \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ (g \circ f_2) \uparrow \Pi_{\{y\}}(F)\ ) \cdot F$$

En utilisant les égalités $g \circ f_1 = g' \circ f'_1$ et $g \circ f_2 = g' \circ f'_2$, F devient :

$$F \quad = \quad (\ (g' \circ f'_1) \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ (g' \circ f'_2) \uparrow \Pi_{\{y\}}(F)\ ) \cdot F$$

En utilisant 7.2, on obtient pour f'_1, f'_2 la suite d'implications suivante :

$$\forall\ t, s \in \mathbb{N},\ f'_1(t) = f'_2(s) \ \Rightarrow\ f_1(t) + c = f_2(s) + c \Rightarrow\ f_1(t) = f_2(s) \Rightarrow x_{(F)}(t) = y_{(F)}(s)$$

On a construit une fonction $g' : \mathbb{N} \to \mathbb{N}$, croissante sur $\mathbb{N}$ et strictement croissante sur $f'_1(\mathbb{N}) \cup f'_2(\mathbb{N})$, telle que

$$F = (\ (g' \circ f'_1) \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ (g' \circ f'_2) \uparrow \Pi_{\{y\}}(F)\ ) \cdot F$$

et

$$\forall\ t, s \in \mathbb{N},\ f'_1(t) = f'_2(s) \ \Rightarrow\ x_{(F)}(t) = y_{(F)}(s)$$

avec f'_1, f'_2 densifications de la forme $f'_1 = \lambda\ t.(nt + \varphi_1 + c)$ et $f'_2 = \lambda\ t.(dt + \varphi_2 + c)$.

Il résulte $x\ \mathcal{R}^F_{((n,\varphi_1+c),(d,\varphi_2+c))}\ y$, i.e. $\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))} \subseteq \mathcal{R}^F_{((n,\varphi_1+c),(d,\varphi_2+c))}$.

• Pour démontrer que $\mathcal{R}^F_{((n,\varphi_1+c),(d,\varphi_2+c))} \subseteq \mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))}$, on applique le cas antérieur pour $c' \in \mathbb{Z}, c' = -c$ et les fonctions $f_i, i \in \{1,2\}$, telles que

$$\begin{aligned} f_1 &= \lambda\ t.(nt + \varphi'_1), \quad \varphi'_1 = \varphi_1 + c \\ f_2 &= \lambda\ t.(dt + \varphi'_2), \quad \varphi'_2 = \varphi_2 + c \end{aligned}$$

(puisque $\varphi'_i + c' = (\varphi_i + c) - c = \varphi_i \geq 0$).

- Finalement, les deux inclusions précédentes conduisent à :

$$\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))} = \mathcal{R}^F_{((n,\varphi_1+c),(d,\varphi_2+c))}$$

□

Étant donné une relation affine $\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))}$, on note $\varphi = \varphi_2 - \varphi_1$. Cf. au lemme précédent, la relation affine $\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))}$ coïncide avec $\mathcal{R}^F_{((n,\varphi_1+c),(d,\varphi_2+c))}$, pour tout c entier tel que les phases $\varphi_1 + c$ et $\varphi_2 + c$ restent positives. En considérant $c = max(0,\varphi) - \varphi_2$, on se place dans les hypothèses du lemme antérieur et on obtient l'identité suivante :

Corollaire 1

$$\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))} = \mathcal{R}^F_{((\ n,max(0,-\varphi) \ ), \ (\ d,max(0,\varphi) \ ))}$$

avec $\varphi = \varphi_2 - \varphi_1$. □

La relation $\mathcal{R}^F_{((\ n,max(0,-\varphi) \ ), \ (\ d,max(0,\varphi) \ ))}$ représente *la forme normale* de la relation $\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))}$ et sera notée $\mathcal{R}^F_{(n,\varphi,d)}$ dans la suite.

Pour toutes les paires $(\varphi_1,\varphi_2),(\varphi'_1,\varphi'_2)$ satisfaisant $\varphi_2 - \varphi_1 = \varphi'_2 - \varphi'_1$, les relations affines $\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))}$ et $\mathcal{R}^F_{((n,\varphi'_1),(d,\varphi'_2))}$ sont identiques. Pour exemplifier, nous considérons le flot $F \in \mathcal{F}_{\{x,y,z\}}$ représenté sur la Figure 7.7 ; F peut s'écrire

$$F = (\ f_1 \uparrow \Pi_{\{x\}}(F) \ ) \cdot (\ f_2 \uparrow \Pi_{\{y\}}(F) \ ) \cdot F$$
$$F = (\ (g' \circ f'_1) \uparrow \Pi_{\{x\}}(F) \ ) \cdot (\ (g' \circ f'_2) \uparrow \Pi_{\{y\}}(F) \ ) \cdot F$$

où les fonctions f_1, f_2, f'_1, f'_2, g' sont définies comme suit :

$$\begin{array}{l} f_1, f_2, f'_1, f'_2, g' : \mathbb{N} \rightarrow \mathbb{N} \\ f_1 = \lambda \ t.(2t) \quad f_2 = \lambda \ t.(4t+1) \quad f'_1 = \lambda \ t.(2t+3) \quad f'_2 = \lambda \ t.(4t+4) \\ g' \text{ croissante sur } \mathbb{N}, \ \ g'(2t+3) = 2t, \ g'(4t+4) = 4t+1, \ \ \forall t \in \mathbb{N} \end{array}$$

et on déduit $\mathcal{R}^F_{((2,0),(4,1))} = \mathcal{R}^F_{((2,3),(4,4))} = \{(x,y)\}$.

Les relations affines $\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))}$ et $\mathcal{R}^F_{((n,\varphi'_1),(d,\varphi'_2))}$ peuvent être représentées par la relation unique $\mathcal{R}^F_{(n,\varphi,d)}$, avec $\varphi = \varphi_2 - \varphi_1 = \varphi'_2 - \varphi'_1$. Toute propriété induite ultérieurement pour $\mathcal{R}^F_{(n,\varphi,d)}$ à partir des valeurs des paramètres (n,φ,d) sera valable pour toute relation $\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))}$, avec $\varphi_2 - \varphi_1 = \varphi$.

Nous concluons en reprenant le corollaire 1 avec la notation normalisée.

Corollaire 2 *(Normalisation)* $\forall \ \varphi_1, \varphi_2 \ \in \mathbb{N}$

$$\mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))} = \mathcal{R}^F_{(n,\varphi,d)}, \qquad \textit{avec } \varphi = \varphi_2 - \varphi_1$$

□

t		0	1	2	3	4	5	6	7	8	9	10	11	
$f_1(t)$		0		2		4		6		8		10		
$f_2(t)$			1				5				9			
$(g' \, o \, f_1')(t)$		$g'(3)$		$g'(5)$		$g'(7)$		$g'(9)$		$g'(11)$		$g'(13)$		
$(g' \, o \, f_2')(t)$			$g'(4)$				$g'(8)$				$g'(12)$			
F	x	x_1	$\perp$	x_2	$\perp$	x_3	$\perp$	x_4	$\perp$	x_5	$\perp$	x_6	$\perp$	....
	y	$\perp$	y_1	$\perp$	$\perp$	$\perp$	y_2	$\perp$	$\perp$	$\perp$	y_3	$\perp$	$\perp$	....
	z	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8	z_9	z_{10}	z_{11}	$\perp$	....

FIG. 7.7 – $\mathcal{R}^F_{((2,0),(4,1))} = \mathcal{R}^F_{((2,3),(4,4))} = \{x, y\}$.

Commentaire

- On retrouve dans (n, φ, d) les paramètres de la transformation affine appliquée à l'horloge H_x de la variable x afin d'obtenir l'horloge H_y de y, pour x et y satisfaisant $x \; \mathcal{R}^F_{(n,\varphi,d)} \; y$ (conformément à la définition d'une transformation affine d'horloges introduite en section 5.2).
- Nous reprenons la définition formelle d'une relation affine en fonction des paramètres (n, φ, d) :
  - $x \; \mathcal{R}^F_{(n,\varphi,d)} \; y$
    Les variables x et y sont en relation affine de paramètres (n, φ, d) dans F ssi elles sont (f_1, f_2)-codensifiées dans F, avec $f_1 = \lambda\, t.(nt + max(0, -\varphi))$, $f_2 = \lambda\, t.(dt + max(0, \varphi))$.
  - $\mathcal{R}^F_{(n,\varphi,d)} = \{(x, y) \in A_1 \times A_1 \; / \; x \; \mathcal{R}^F_{(n,\varphi,d)} \; y \;\}$

□

7.1.2.2 Fonction d'occurrence

La fonction d'occurrence d'une variable par rapport à une autre est une expression de leur positionnement relatif ; dans le cas des variables en relation affine, elle dépend uniquement des valeurs (n, φ, d) des paramètres de la relation affine et met en évidence le caractère régulier de leur positionnement relatif.

Soit $F \in \mathcal{F}_{A_1}$ et x, y un couple de variables telles que $\{x, y\} \subseteq A_1$. D'après le lemme de recomposition de flots, il existe $f_x, f_y : \mathbb{N} \rightarrow \mathbb{N}$ fonctions de densification telles que

$$F = (f_x \uparrow \Pi_{\{x\}}(F)) \cdot (f_y \uparrow \Pi_{\{y\}}(F)) \cdot F \tag{7.3}$$

La fonction d'occurrence de y par rapport à x dans le flot F est une application $\mathcal{A}^F_x(y)$ définie comme suit :

$$\mathcal{A}^F_x(y) \; : \; \mathbb{N}^* \rightarrow \mathbb{N},$$
$$\mathcal{A}^F_x(y)(t) = \begin{cases} s+1, & \text{si } \exists s \in \mathbb{N} \text{ maximal tel que } f_y(s) \leq f_x(t-1) \\ 0, & \text{sinon} \end{cases}$$

("$s \in \mathbb{N}$ maximal tel que $f_y(s) \leq f_x(t)$ " représente le plus grand $s \in \mathbb{N}$ avec la propriété considérée).

Pour tout $t \in \mathbb{N}^*$, $\mathcal{A}_x^F(y)(t)$ désigne le nombre d'occurrences de y qui ne suivent pas strictement la t^{eme} occurrence de x dans F ou le nombre d'occurrences de y qui précèdent ou sont égales à la t^{eme} occurrence de x dans F (voir exemple sur la Figure 7.8).

t		0	1	2	3	4	5	6	7	8	9	10	11	
F	x	x_1	$\perp$	$\perp$	x_2	x_3	$\perp$	x_4	$\perp$	$\perp$	x_5	x_6	x_7	....
	y	$\perp$	y_1	y_2	$\perp$	y_3	y_4	$\perp$	y_5	$\perp$	$\perp$	$\perp$	y_6	....

$\mathcal{A}_x^F(y)\ (7) = 6$

FIG. 7.8 – *Illustration de la fonction d'occurrence.*

De manière symétrique peut être définie la fonction d'occurrence de x par rapport à y : $\mathcal{A}_y^F(x) : \mathbb{N}^* \rightarrow \mathbb{N}$.

Les applications $\mathcal{A}_x^F(y)$ et $\mathcal{A}_y^F(x)$ dépendent des fonctions de densification f_x et f_y qui définissent l'alternance des occurrences de x et y dans le flot F. Afin de prendre en compte cette propriété, nous préférons les notations $\mathcal{A}_x^F(y)_{(f_x, f_y)}$ et respectivement $\mathcal{A}_y^F(x)_{(f_x, f_y)}$ qui permettront, dans le contexte des relations affines, de définir des fonctions d'occurrence différentes pour des relations affines de paramètres différents.

Dans le cas où $(x, y) \in \mathcal{R}^F_{(n,\varphi,d)}$, $\exists\ g : \mathbb{N} \rightarrow \mathbb{N}$, croissante sur $\mathbb{N}$ et strictement croissante sur $f_1(\mathbb{N}) \cup f_2(\mathbb{N})$, telles que

$$F = (\ (g \circ f_1) \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ (g \circ f_2) \uparrow \Pi_{\{y\}}(F)\ ) \cdot F$$

et

$$\forall\ t, s \in \mathbb{N},\ f_1(t) = f_2(s)\ \Rightarrow\ x_{(F)}(t) = y_{(F)}(s)$$

avec $f_1 = \lambda\ t.(nt + max(0, -\varphi))$, $f_2 = \lambda\ t.(dt + max(0, \varphi))$.

On notera

$$\mathcal{A}_x^F(y)_{(n,\varphi,d)} \stackrel{not}{=} \mathcal{A}_x^F(y)_{(g\ \circ\ f_1, g\ \circ\ f_2)}$$
$$\mathcal{A}_y^F(x)_{(n,\varphi,d)} \stackrel{not}{=} \mathcal{A}_y^F(x)_{(g\ \circ\ f_1, g\ \circ\ f_2)}$$

les fonctions d'occurrence relatives de x et y afin de souligner l'influence de la relation affine de paramètres (n, φ, d) sur le positionnement relatif de x et y.

En effet, nous allons voir que les applications $\mathcal{A}_x^F(y)_{(n,\varphi,d)}$ et $\mathcal{A}_y^F(x)_{(n,\varphi,d)}$ dépendent uniquement des valeurs du triplet (n, φ, d).

Lemme 3 $\forall\ (x, y) \in \mathcal{R}^F_{(n,\varphi,d)}$ *et* $\forall\ t \in \mathbb{N}^*$,

$$\mathcal{A}_y^F(x)_{(n,\varphi,d)}(t) \quad = \quad \begin{cases} 0, & si\ d(t-1) + \varphi < 0 \\ \left[\frac{d(t-1)+\varphi}{n}\right] + 1, & sinon \end{cases} \tag{7.4}$$

$$\mathcal{A}_x^F(y)_{(n,\varphi,d)}(t) \quad = \quad \begin{cases} 0, & si\ n(t-1) - \varphi < 0 \\ \left[\frac{n(t-1)-\varphi}{d}\right] + 1, & sinon \end{cases} \tag{7.5}$$

avec $[n]$ la partie entière de n ($[n]\ \in \mathbb{Z}$).

Preuve On va démontrer (7.4), la preuve pour la deuxième égalité étant similaire.

Comme $(x, y) \in \mathcal{R}^F_{(n,\varphi,d)}$, $\exists\, g : \mathbb{N} \rightarrow \mathbb{N}$, croissante sur $\mathbb{N}$ et strictement croissante sur $f_1(\mathbb{N}) \cup f_2(\mathbb{N})$, telle que

$$F = (\ (g \circ f_1) \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ (g \circ f_2) \uparrow \Pi_{\{y\}}(F)\ ) \cdot F$$

et

$$\forall\, t, s \in \mathbb{N},\ f_1(t) = f_2(s) \ \Rightarrow\ x_{(F)}(t) = y_{(F)}(s)$$

avec $f_1 = \lambda\, t.(nt + max(0, -\varphi))$, $f_2 = \lambda\, t.(dt + max(0, \varphi))$.

La définition de $\mathcal{A}^F_y(x)_{(g \circ f_1, g \circ f_2)}$ et g strictement croissante sur $f_1(\mathbb{N}) \cup f_2(\mathbb{N})$ impliquent ($\forall\, t \in \mathbb{N}^*$) :

$$\begin{aligned}
\mathcal{A}^F_y(x)_{(n,\varphi,d)}(t) &= \mathcal{A}^F_y(x)_{(g \circ f_1, g \circ f_2)} \\
&= \begin{cases} s+1, & \text{si } \exists s \in \mathbb{N} \text{ maximal, } (g \circ f_1)(s) \leq (g \circ f_2)(t-1) \\ 0, & \text{sinon} \end{cases} \\
&= \begin{cases} s+1, & \text{si } \exists s \in \mathbb{N} \text{ maximal, } f_1(s) \leq f_2(t-1) \\ 0, & \text{sinon} \end{cases} \\
&= \begin{cases} s+1, & \text{si } \exists s \in \mathbb{N} \text{ maximal,} \\ & \quad ns + max(0, -\varphi) \leq d(t-1) + max(0, \varphi) \\ 0, & \text{sinon} \end{cases} \\
&= \begin{cases} s+1, & \text{si } \exists s \in \mathbb{N} \text{ maximal, } s \leq \frac{d(t-1)+\varphi}{n} \\ 0, & \text{sinon} \end{cases} \\
&= \begin{cases} 0, & \text{si } (d(t-1) + \varphi) < 0 \\ \left\lfloor \frac{d(t-1)+\varphi}{n} \right\rfloor + 1, & \text{sinon} \end{cases}
\end{aligned}$$

$\square$

Les fonctions d'occurrence associées à deux variables x et y en relation affine possèdent des expressions de calcul simples et qui dépendent uniquement des paramètres (n, φ, d) de la relation affine. Les fonctions d'occurrence définissent la position relative de deux variables dans un flot et par ceci sont intrinsèquement liées à la relation, éventuellement affine, existant entre les variables. Par exemple, une condition nécessaire et suffisante pour l'égalité de deux relations affines

$$\mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')}$$

est que les fonctions d'occurrence relatives de x et y, pour toute paire $(x, y) \in \mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')}$, soient égales :

$$a \quad \begin{cases} \mathcal{A}^F_x(y)_{(n,\varphi,d)} = \mathcal{A}^F_x(y)_{(n',\varphi',d')} \\ \mathcal{A}^F_y(x)_{(n,\varphi,d)} = \mathcal{A}^F_y(x)_{(n',\varphi',d')} \end{cases}$$

Le système a impose des contraintes sur les valeurs des triplets (n, φ, d) et (n', φ', d') qui représenteront des conditions nécessaires et suffisantes pour l'égalité (équivalence) des relations affines correspondantes. Or, comme nous allons le voir dans la suite, une seule équation du système a est suffisante pour caractériser l'ensemble de ces conditions, i.e.

$$\mathcal{A}^F_x(y)_{(n,\varphi,d)} = \mathcal{A}^F_x(y)_{(n',\varphi',d')} \ \Leftrightarrow\ \mathcal{A}^F_y(x)_{(n,\varphi,d)} = \mathcal{A}^F_y(x)_{(n',\varphi',d')}$$

Nous énonçons et démontrons maintenant un lemme qui illustre cette idée ; il constituera le support pour toute preuve d'équivalence de relations affines.

Lemme 4 *Pour tout flot* $F \in \mathcal{F}_{A_1}$ *et tout doublet* $(x, y) \in \mathcal{R}^F_{(n,\varphi,d)} \cap \mathcal{R}^F_{(n',\varphi',d')}$*, on considère les systèmes d'égalités suivants :*

$$a \quad \begin{cases} \mathcal{A}^F_x(y)_{(n,\varphi,d)} & = \mathcal{A}^F_x(y)_{(n',\varphi',d')} \\ \mathcal{A}^F_y(x)_{(n,\varphi,d)} & = \mathcal{A}^F_y(x)_{(n',\varphi',d')} \end{cases}$$

$$b \quad \left\{ \mathcal{A}^F_x(y)_{(n,\varphi,d)} = \mathcal{A}^F_x(y)_{(n',\varphi',d')} \right.$$

$$c \quad \left\{ \mathcal{A}^F_y(x)_{(n,\varphi,d)} = \mathcal{A}^F_y(x)_{(n',\varphi',d')} \right.$$

Les équivalences suivantes sont vérifiées :

1. $b \Leftrightarrow c$ **2.** $a \Leftrightarrow b$ **3.** $a \Leftrightarrow c$

Preuve Pour la preuve complète du lemme, nous renvoyons le lecteur à l'annexe A.

L'idée est qu'il suffit d'une seule fonction d'occurrence pour spécifier complètement la position relative des variables x et y. Effectivement, $\mathcal{A}^F_y(x)_{(n,\varphi,d)} = \mathcal{A}^F_y(x)_{(n',\varphi',d')}$ impose la même position relative de x et y, définie en même temps par les relations affines $\mathcal{R}^F_{(n,\varphi,d)}$ et $\mathcal{R}^F_{(n',\varphi',d')}$: $x \; \mathcal{R}^F_{(n,\varphi,d)} \; y$ et $x \; \mathcal{R}^F_{(n',\varphi',d')} \; y$. Il résulte aussi $\mathcal{A}^F_x(y)_{(n,\varphi,d)} = \mathcal{A}^F_x(y)_{(n',\varphi',d')}$. □

7.1.3 Équivalence de relations affines

Soit un processus $\mathbf{P}$ tel que $(x, y_1) \in \mathcal{R}^F_{(n,\varphi,d)}$ et $(x, y_2) \in \mathcal{R}^F_{(n',\varphi',d')}$, pour tout flot F appartenant à $\mathbf{P}$ (comme on le définira plus loin, cela veut dire que les variables x, y_1 et x, y_2 sont respectivement en relation affine de paramètres (n, φ, d) et (n', φ', d') *dans le processus* $\mathbf{P}$). Les variables y_1 et y_2 sont identifiables dans $\mathbf{P}$ si et seulement si il existe un flot de $\mathbf{P}$ dans lequel elles sont identiques. Supposons $F_s \in \mathbf{P}$ tel que y_1 et y_2 soient identiques dans F_s ; cela veut dire que y_1 et y_2 peuvent être remplacés dans F_s et il résulte $(x, y_2) \in \mathcal{R}^{F_s}_{(n,\varphi,d)}$ et $(x, y_1) \in \mathcal{R}^{F_s}_{(n',\varphi',d')}$. Dans le cas particulier $\mathbf{P} \in \mathcal{P}_{\{x,y_1,y_2\}}$, on obtient $\mathcal{R}^{F_s}_{(n,\varphi,d)} = \{(x, y_1), (x, y_2)\}$ et $\mathcal{R}^{F_s}_{(n',\varphi',d')} = \{(x, y_1), (x, y_2)\}$. En conclusion, une condition nécessaire pour que y_1 et y_2 soient identifiables dans $\mathbf{P} \in \mathcal{P}_{\{x,y_1,y_2\}}$ est qu'il existe un flot $F_s \in \mathbf{P}$ ($F_s \in \mathcal{F}_{\{x,y_1,y_2\}}$) tel que $\mathcal{R}^{F_s}_{(n,\varphi,d)} = \mathcal{R}^{F_s}_{(n',\varphi',d')}$. Cette condition est suffisante s'il n'existe pas d'autres contraintes sur x, y_1, y_2 à part les contraintes imposées par les relations affines $(x, y_1) \in \mathcal{R}^F_{(n,\varphi,d)}$ et $(x, y_2) \in \mathcal{R}^F_{(n',\varphi',d')}$, pour tout flot $F \in \mathbf{P}$.

Dans l'exemple antérieur, $\mathcal{R}^{F_s}_{(n,\varphi,d)} = \mathcal{R}^{F_s}_{(n',\varphi',d')}$ apparaît comme une condition nécessaire pour que y_1 et y_2 soient identifiables dans $\mathbf{P}$. On se propose en conséquence d'étudier dans quelles conditions les relations non vides $\mathcal{R}^F_{(n,\varphi,d)}$ et $\mathcal{R}^F_{(n',\varphi',d')}$ sont égales, pour $F \in \mathcal{F}_{A_1}$ arbitraire. L'égalité de deux relations affines associées à un flot est une équivalence qui induit une relation d'équivalence sur les paramètres (n, φ, d) et (n', φ', d') associés à chaque relation. Pour toute relation affine $\mathcal{R}^F_{(n,\varphi,d)}$, nous mettrons en évidence une forme canonique définie comme le représentant de la classe d'équivalence associée à $\mathcal{R}^F_{(n,\varphi,d)}$.

Dans le cas général, les relations arbitraires $\mathcal{R}_1$ et $\mathcal{R}_2$ sont égales si et seulement si $x \; \mathcal{R}_1 \; y$ et $x \; \mathcal{R}_2 \; y$ sont équivalentes [San80] :

$$\mathcal{R}_1 = \mathcal{R}_2 \;\Leftrightarrow\; (x \; \mathcal{R}_1 \; y \;\Leftrightarrow\; x \; \mathcal{R}_2 \; y, \; \forall \, x, y)$$

Dans le cas des relations affines on obtient :

Définition 4 *Pour tout flot $F \in \mathcal{F}_{A_1}$, les relations $\mathcal{R}^F_{(n,\varphi,d)}$ et $\mathcal{R}^F_{(n',\varphi',d')}$ sont égales ssi la relation suivante est vérifiée :*

$$x \ \mathcal{R}^F_{(n,\varphi,d)} \ y \quad \Leftrightarrow \quad x \ \mathcal{R}^F_{(n',\varphi',d')} \ y \ , \ \ \forall \ x, y \in A_1$$

□

Les propriétés des relations affines dont on dispose à présent nous permettent de démontrer la proposition suivante[2] :

Proposition 1 *Pour tout flot $F \in \mathcal{F}_{A_1}$,*

$$\mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')}$$

ssi les contraintes (a) et (b) ci-dessous sont valides en exclusion mutuelle :

$$(a) \quad \begin{array}{l} (\ pgcd(n,d) \mid \varphi) \\ (\ pgcd(n',d') \mid \varphi') \\ nd' = n'd \\ n\varphi' = n'\varphi \end{array} \qquad (b) \quad \begin{array}{l} (\ pgcd(n,d) \not| \ \varphi) \\ (\ pgcd(n',d') \not| \ \varphi') \\ nd' = n'd \\ \left[\frac{dt+\varphi}{n}\right] \ = \ \left[\frac{d't+\varphi'}{n'}\right] \ , \forall t \in \mathbb{N}, dt+\varphi \geq 0 \end{array}$$

avec $pgcd(n,d)$ le plus grand diviseur commun de n et d.

Preuve Pour la preuve complète de la Proposition, nous renvoyons le lecteur à l'annexe B. Nous esquissons ici les idées principales de la preuve :

1) Si $\mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')}$, il résulte

$$\begin{array}{l} \mathcal{A}^F_x(y)_{(n,\varphi,d)} = \mathcal{A}^F_x(y)_{(n',\varphi',d')} \\ \mathcal{A}^F_y(x)_{(n,\varphi,d)} = \mathcal{A}^F_y(x)_{(n',\varphi',d')} \end{array}$$

pour toute paire (x, y) appartenant également à $\mathcal{R}^F_{(n,\varphi,d)}$ et $\mathcal{R}^F_{(n',\varphi',d')}$.

On obtient

$$\left[\frac{dt+\varphi}{n}\right] \ = \ \left[\frac{d't+\varphi'}{n'}\right] \ , \forall t \in \mathbb{N}, dt+\varphi \geq 0$$

qui induit les conditions (a) et (b) en fonction des cas $pgcd(n,d) \mid \varphi$, $pgcd(n',d') \mid \varphi'$ et respectivement $pgcd(n,d) \not| \ \varphi$, $pgcd(n',d') \not| \ \varphi'$.

2) Plus généralement, on a

$$\left[\frac{dt+\varphi}{n}\right] \ = \ \left[\frac{d't+\varphi'}{n'}\right] \ , \forall t \in \mathbb{N}, dt+\varphi \geq 0 \quad \Leftrightarrow \quad (a) \text{ ou } (b)$$

A partir de la partie droite de l'équivalence ci-dessus, on prouve que toute paire (x, y) de $\mathcal{R}^F_{(n,\varphi,d)}$ est également membre de $\mathcal{R}^F_{(n',\varphi',d')}$ et réciproquement, ce qui implique $\mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')}$.

2. Pour $a, b \in \mathbb{Z}$, on note $a \mid b$ si a est un diviseur de b et $a \not| \ b$ sinon.

□

A titre d'exemple, on a

$$(a) \quad \mathcal{R}^F_{(2,2,4)} = \mathcal{R}^F_{(3,3,6)} \qquad (b) \quad \mathcal{R}^F_{(2,1,4)} = \mathcal{R}^F_{(3,2,6)}$$

pour tout flot $F \in \mathcal{F}_{A_1}$; pour chaque cas (a), (b) ci-dessus, les valeurs des paramètres (n, φ, d) et (n', φ', d') vérifient les conditions (a), respectivement (b) de la proposition antérieure. Nous avons représenté sur les Figures 7.9 et 7.10 deux flots $F_1, F_2 \in \mathcal{F}_{\{x,y,z\}}$ tels que $\mathcal{R}^{F_1}_{(2,2,4)} = \mathcal{R}^{F_1}_{(3,3,6)} = \{(x,y)\}$ et $\mathcal{R}^{F_2}_{(2,1,4)} = \mathcal{R}^{F_2}_{(3,2,6)} = \{(x,y)\}$ (sur la Figure 7.9, les suites de valeurs de x et y vérifient $x_{2i} = y_i, \forall i \in \mathbb{N}^*$). En prenant, par exemple, le flot F_1, on remarque que on peut l'écrire sous la forme

$$F_1 = (\ f_1 \uparrow \Pi_{\{x\}}(F_1)\ ) \cdot (\ f_2 \uparrow \Pi_{\{y\}}(F_1)\ ) \cdot F_1$$
$$F_1 = (\ (g' \circ f'_1) \uparrow \Pi_{\{x\}}(F_1)\ ) \cdot (\ (g' \circ f'_2) \uparrow \Pi_{\{y\}}(F_1)\ ) \cdot F_1$$

et

$$\forall\ t, s \in \mathbb{N},\ f_1(t) = f_2(s) \ \Rightarrow\ x_{(F_1)}(t) = y_{(F_1)}(s)$$
$$\forall\ t, s \in \mathbb{N},\ f'_1(t) = f'_2(s) \ \Rightarrow\ x_{(F_1)}(t) = y_{(F_1)}(s)$$

où les fonctions f_1, f_2, f'_1, f'_2, g' sont définies comme suit :

$$\begin{array}{l} f_1, f_2, f'_1, f'_2, g' : \mathbb{N} \ \rightarrow\ \mathbb{N} \\ f_1 = \lambda\ t.(2t) \quad f_2 = \lambda\ t.(4t+2), \quad f'_1 = \lambda\ t.(3t) \quad f'_2 = \lambda\ t.(6t+3) \\ g' \text{ croissante sur } \mathbb{N},\ \ g'(3t) = 2t,\ \ g'(6t+3) = 4t+2, \ \ \forall t \in \mathbb{N} \end{array}$$

et on déduit que $\mathcal{R}^{F_1}_{(2,2,4)} = \mathcal{R}^{F_1}_{(3,3,6)} = \{(x,y)\}$.

t	0	1	2	3	4	5	6	7	8	9	10	11	
$f_1(t)$	0		2		4		6		8		10	....	
$f_2(t)$			2				6				10	....	
$(g' o f'_1)(t)$	$g'(0)$		$g'(3)$		$g'(6)$		$g'(9)$		$g'(12)$		$g'(15)$	....	
$(g' o f'_2)(t)$			$g'(3)$				$g'(9)$				$g'(15)$	....	

F_1 (a)													
x	x_1	$\perp$	x_2	$\perp$	x_3	$\perp$	x_4	$\perp$	x_5	$\perp$	x_6	$\perp$	....
y	$\perp$	$\perp$	y_1	$\perp$	$\perp$	$\perp$	y_2	$\perp$	$\perp$	$\perp$	y_3	$\perp$	....
z	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8	z_9	z_{10}	z_{11}	$\perp$	....

FIG. 7.9 – *Illustration de l'égalité de relations affines – conditions (a).*

De manière similaire on obtient pour F_2 que $\mathcal{R}^{F_2}_{(2,1,4)} = \mathcal{R}^{F_2}_{(3,2,6)} = \{(x,y)\}$.

Les conditions (a) et (b) de la Proposition 1 sont en fait équivalentes à la condition

$$\left[\frac{dt+\varphi}{n}\right] \ = \ \left[\frac{d't+\varphi'}{n'}\right], \forall t \in \mathbb{N}, dt + \varphi \geq 0 \tag{7.6}$$

t	0	1	2	3	4	5	6	7	8	9	10	11
$f_3(t)$	0		2		4		6		8		10	....
$f_4(t)$		1				5				9		....
$(g'' \circ f'_3)(t)$	$g''(0)$		$g''(3)$		$g''(6)$		$g''(9)$		$g''(12)$		$g''(15)$	....
$(g'' \circ f'_4)(t)$		$g''(2)$				$g''(8)$				$g''(14)$		....

F_2	x	x_1	$\perp$	x_2	$\perp$	x_3	$\perp$	x_4	$\perp$	x_5	$\perp$	x_6	$\perp$	....
(b)	y	$\perp$	y_1	$\perp$	$\perp$	$\perp$	y_2	$\perp$	$\perp$	$\perp$	y_3	$\perp$	$\perp$	....
	z	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8	z_9	z_{10}	z_{11}	$\perp$	....

FIG. 7.10 – *Illustration de l'égalité de relations affines – conditions (b).*

Dans le cas où $pgcd(n,d) \mid \varphi$ et $pgcd(n',d') \mid \varphi'$, l'équation 7.6 est équivalente à

$$nd' = n'd \quad \text{et} \quad n\varphi' = n'\varphi$$

ce qui implique

$$n = kn'', d = kd'', \varphi = k\varphi'' \quad \text{et} \quad n' = k'n'', d' = k'd'', \varphi' = k'\varphi''$$

avec $n'', d'' \in \mathbb{N}^*$ et $\varphi'' \in \mathbb{Z}$. Dans ce cas, on dit que les relations affines $\mathcal{R}^F_{(n,\varphi,d)}$ et $\mathcal{R}^F_{(n',\varphi',d')}$ sont équivalentes ssi leurs triplets de paramètres (n,φ,d) et (n',φ',d') sont des multiples entiers d'un même triplet (n'',φ'',d'').

Dans le cas général, l'équation 7.6 est équivalente à

$$\mathcal{A}^F_y(x)_{(n,\varphi,d)} = \mathcal{A}^F_y(x)_{(n',\varphi',d')}$$

et, en utilisant le Lemme 4, il résulte aussi

$$\mathcal{A}^F_x(y)_{(n,\varphi,d)} = \mathcal{A}^F_x(y)_{(n',\varphi',d')}$$

D'après les définitions des fonctions d'occurrence, l'équation 7.6 précise qu'il existe le même nombre d'occurrence de x entre deux occurrences successives de y et vice-versa, quand $x\ \mathcal{R}^F_{(n,\varphi,d)}\ y$ et $x\ \mathcal{R}^F_{(n',\varphi',d')}\ y$.

L'égalité de relations affines induit une relation d'équivalence entre les triplets de paramètres associés à deux relations égales. Nous notons $\sim$ cette relation d'équivalence définie comme suit :

Définition 5 *On dit que les triplets (n,φ,d) et (n',φ',d') sont équivalents, et on note $(n,\varphi,d) \sim (n',\varphi',d')$, si les relations affines qu'ils caractérisent sont égales :*

$$(n,\varphi,d) \ \sim\ (n',\varphi',d') \quad \Leftrightarrow \quad \mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')}, \ \ \forall\ F \in \mathcal{F}_{A_1}$$

□

Vérifier l'égalité de $\mathcal{R}^F_{(n,\varphi,d)}$ et $\mathcal{R}^F_{(n',\varphi',d')}$ revient à vérifier que les paramètres (n,φ,d) et (n',φ',d') sont équivalents, c'est-à-dire qu'ils vérifient les conditions (a) ou (b) de la Proposition 1. Afin de réduire la complexité de cette vérification, nous avons défini une forme canonique pour toute relation affine : un représentant unique de la classe d'équivalence associée à la relation affine considérée :

$$FC(\mathcal{R}^F_{(n,\varphi,d)}) = \mathcal{R}^F_{(n_{FC},\varphi_{FC},d_{FC})}$$

avec $(n_{FC},\varphi_{FC},d_{FC})$ la forme canonique de (n,φ,d), i.e. le représentant de la classe d'équivalence de (n,φ,d). Deux relations affines sont équivalentes si elles partagent le même représentant, c'est-à-dire elles possèdent des formes canoniques identiques. Les résultats exposés dans la section suivante ont été présentés en [SL97a] [SL97b].

7.1.4 Forme canonique d'une relation affine

On définit sur l'ensemble des triplets $(n,\varphi,d) \in \mathbb{N}^* \times \mathbb{Z} \times \mathbb{N}^*$ un ordre partiel $\leq_t$ comme suit :

$$(n,\varphi,d) \leq_t (n',\varphi',d') \text{ ssi } n \leq n', \mid \varphi \mid \ \leq \mid \varphi' \mid \ \text{ et } d \leq d'$$

L'ensemble des triplets (n',φ',d') satisfaisant les contraintes (a) ou (b) de la Proposition 1 représente la classe d'équivalence de (n,φ,d). Sur la classe d'équivalence de (n,φ,d), l'ordre $\leq_t$ est total ; chaque classe d'équivalence admet un plus petit élément par rapport à l'ordre $\leq_t$. Cet élément sera désigné comme représentant de la classe d'équivalence ; il constituera la forme canonique de (n,φ,d).

Définition 6 *On définit la forme canonique d'un triplet (n,φ,d) comme le plus petit élément par rapport à l'ordre $\leq_t$ de la classe d'équivalence associée à (n,φ,d).* □

Théorème 2 *Le triplet (n,φ,d) admet une forme canonique $(n_{FC},\varphi_{FC},d_{FC})$ définie comme suit :*

$$\begin{array}{lll} a) \ G \mid \varphi & \Rightarrow & (n_{FC},\varphi_{FC},d_{FC}) = (\frac{n}{G}, \frac{\varphi}{G}, \frac{d}{G}) \\ b) \ G \not\mid \varphi & \Rightarrow & (n_{FC},\varphi_{FC},d_{FC}) = (2\frac{n}{G}, (2 \ [\frac{\varphi}{G}] \ +1), 2\frac{d}{G}) \end{array}$$

avec $G = \ pgcd(n,d)$.

Preuve Pour la preuve, nous renvoyons le lecteur à l'annexe C. □

Par extension, la forme canonique de $\mathcal{R}^F_{(n,\varphi,d)}$ sera la relation $\mathcal{R}^F_{(n_{FC},\varphi_{FC},d_{FC})}$. Le test d'égalité entre deux relations affines $\mathcal{R}^F_{(n,\varphi,d)}$ et $\mathcal{R}^F_{(n',\varphi',d')}$ qui consistait (cf. Proposition 1 et Définition 5) en un test d'équivalence de (n,φ,d) et (n',φ',d'), est réduit, par l'existence d'une forme canonique pour toute relation affine, à une simple vérification que les triplets $(n_{FC},\varphi_{FC},d_{FC})$ et $(n'_{FC},\varphi'_{FC},d'_{FC})$ soient identiques ($n_{FC} = n'_{FC}$, $\varphi_{FC} = \varphi'_{FC}$ et $d_{FC} = d'_{FC}$) :

Corollaire 3

$$\mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')} \quad \Leftrightarrow \quad (n_{FC},\varphi_{FC},d_{FC}) = (n'_{FC},\varphi'_{FC},d'_{FC})$$

Preuve • En supposant $\mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')}$, il résulte que les classes d'équivalence des triplets (n,φ,d) et (n',φ',d') coïncident et elles admettent le même plus petit élément par rapport à l'ordre $\leq_t$, c'est-à-dire $(n_{FC},\varphi_{FC},d_{FC}) = (n'_{FC},\varphi'_{FC},d'_{FC})$.

• Réciproquement, si $(n_{FC},\varphi_{FC},d_{FC}) = (n'_{FC},\varphi'_{FC},d'_{FC})$, on veut démontrer $\mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')}$.

L'égalité des formes canoniques $(n_{FC},\varphi_{FC},d_{FC})$ et $(n'_{FC},\varphi'_{FC},d'_{FC})$ implique évidemment leur équivalence : $(n_{FC},\varphi_{FC},d_{FC}) \sim (n'_{FC},\varphi'_{FC},d'_{FC})$.

On obtient alors :

$$\left.\begin{array}{lcl}(n_{FC},\varphi_{FC},d_{FC}) & \sim & (n,\varphi,d)\\ (n'_{FC},\varphi'_{FC},d'_{FC}) & \sim & (n',\varphi',d')\\ (n_{FC},\varphi_{FC},d_{FC}) & \sim & (n'_{FC},\varphi'_{FC},d'_{FC})\end{array}\right\} \Rightarrow (n,\varphi,d) \sim (n',\varphi',d')$$

$$\Rightarrow \mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')}$$

• En conclusion :

$$\mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')} \Leftrightarrow (n_{FC},\varphi_{FC},d_{FC}) = (n'_{FC},\varphi'_{FC},d'_{FC}) \qquad \text{(qed).}$$

□

7.1.5 Opérations sur relations affines & Expressions induites

Vérifier l'équivalence de relations affines permettrait, dans certaines conditions, de décider sur l'identifiabilité de deux variables arbitraires y_1 et y_2 : par exemple, dans un processus $\mathbf{P} \in \mathcal{P}_{\{x,y_1,y_2\}}$ dans lequel $x\ \mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ y_1$ et $x\ \mathcal{R}^F_{(n_2,\varphi_2,d_2)}\ y_2$, pour tout flot $F \in \mathbf{P}$. De manière plus générale, on se pose la question dans quelle mesure on peut décider sur l'identifiabilité des variables y_1 et y_2 dans $\mathbf{P} \in \mathcal{P}_{\{x,y_1,y_2\}}$, si $x\ \mathcal{R}^F_{t_1}\ y_1$ et $x\ \mathcal{R}^F_{t_2}\ y_2$, pour tout flot $F \in \mathbf{P}$, avec $\mathcal{R}^F_{t_1}$ et $\mathcal{R}^F_{t_2}$ des expressions arbitraires sur des relations affines de type $\mathcal{R}^F_{(n,\varphi,d)}$. Pour cela, il est nécessaire d'étudier la forme de ces expressions et plus particulièrement les propriétés des opérateurs (et opérations induites) qui interviennent. Il s'agit d'opérations standard d'inversion et composition de relations affines – comme défini dans [San80] – mais aussi d'opérations d'*union*, *intersection* et *différence* correspondant aux opérations homonymes définies sur deux variables arbitraires (voir section 6.4.8.4).

Dans les sections qui suivent, nous introduisons ces opérations et étudions leurs propriétés basées sur les valeurs des triplets (n,φ,d) associés aux relations affines opérandes. Nous commençons par la composition de relations affines et mettons en évidence des triplets (n_1,φ_1,d_1) et (n_2,φ_2,d_2) tels que

$$\mathcal{R}^F_{(n_1,\varphi_1,d_1)} * \mathcal{R}^F_{(n_2,\varphi_2,d_2)} = \mathcal{R}^F_{(n,\varphi,d)}$$

Pour le cas général $n_1,d_1,n_2,d_2 \in \mathbb{N}^*$ et $\varphi_1,\varphi_2 \in \mathbb{Z}$, nous nous intéressons à déterminer l'ensemble des relations affines $\mathcal{R}^F_{(n,\varphi,d)}$ telles que

$$\mathcal{R}^F_{(n_1,\varphi_1,d_1)} * \mathcal{R}^F_{(n_2,\varphi_2,d_2)} = \mathcal{R}^F_{(n,\varphi,d)}$$

Ces résultats serviront ensuite pour décider sur l'identifiabilité des variables y_1 et y_2 dans un processus $\mathbf{P}$ comme celui définit antérieurement et quand $\mathcal{R}^F_{t_1}$ et $\mathcal{R}^F_{t_2}$ sont des relations affines ou des expressions construites par composition de relations affines.

Nous définissons ensuite des opérations d'union, intersection et différence sur des relations affines qui correspondent aux opérations homonymes définies sur les variables et plus particulièrement sur les variables d'horloges. Nous leur attribuons des notations spécifiques afin de ne pas les confondre avec les opérateurs classiques d'union, intersection et différence sur relations générales que le lecteur peut retrouver dans [San80] : nous notons $\cup_r, \cap_r$ et $\backslash_r$ nos union, intersection et respectivement différence de relations affines. Ces opérations sont définies telles que, si $u\ \mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ x_1$ et $u\ \mathcal{R}^F_{(n_2,\varphi_2,d_2)}\ x_2$ dans F, on a $(u, x_1\ op_v\ x_2) \in \mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ op_r\ \mathcal{R}^F_{(n_2,\varphi_2,d_2)}$, pour $op_v \in \{\vee_v, \wedge_v, \backslash_v\}$, $op_r \in \{\cup_r, \cap_r, \backslash_r\}$ et op_v est l'opérateur sur variables correspondant à l'opérateur op_r sur relations affines. Comme dans le cas de la composition de relations affines, pour chaque opération $op_r \in \{\cup_r, \cap_r, \backslash_r\}$, nous mettons en évidence des triplets de paramètres (n_1, φ_1, d_1) et (n_2, φ_2, d_2) tels qu'il existe (n, φ, d) avec la propriété

$$\mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ op_r\ \mathcal{R}^F_{(n_2,\varphi_2,d_2)} = \mathcal{R}^F_{(n,\varphi,d)}$$

Pour le cas général, il est intéressant de déterminer l'ensemble de tous les triplets (n, φ, d) tels que

$$\mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ op_r\ \mathcal{R}^F_{(n_2,\varphi_2,d_2)} = \mathcal{R}^F_{(n,\varphi,d)}$$

Pour conclure, on va dire qu'on étudie les conditions dans lesquelles l'espace des relations affines est fermé par rapport aux opérations précédentes, c'est-à-dire toute opération de composition, union, intersection et différence de relations affines produit une relation affine. Dans le cas où la propriété de fermeture était vraie pour toutes ces opérations, le test d'identifiabilité se réduirait à un test d'équivalence de relations affines. Or, dès le début on remarque que l'espace des relations affines n'est pas fermé par rapport à la composition, mais un sous-ensemble de cet espace possède cette propriété : il s'agit des relations affines $\mathcal{R}^F_{(1,\varphi,d)}$, avec $\varphi \geq 0$. Même dans ce cas simple, la propriété de fermeture se perd dès que l'on fait intervenir l'opération d'union de relations affines : uniquement dans certains cas particuliers, l'union des relations $\mathcal{R}^F_{(1,\varphi_1,d_1)}$ et $\mathcal{R}^F_{(1,\varphi_2,d_2)}$ produit une relation affine. Finalement, nous choisissons comme espace de travail l'ensemble des expressions construites avec les opérateurs de composition, union, intersection et différence sur des relations affines de type $\mathcal{R}^F_{(1,\varphi,d)}$, avec $\varphi \geq 0$, et étudions le problème de l'existence d'une *forme unique* d'expression dans cet espace. Nous mettrons en évidence une *forme normale relative* qui nous permettra de décider de l'identifiabilité de deux variables, quand celle-ci fait intervenir des expressions arbitraires $\mathcal{R}^F_{t_1}$ et $\mathcal{R}^F_{t_2}$ de cet espace.

Dans l'espace des relations affines générales de type $\mathcal{R}^F_{(n,\varphi,d)}$, déterminer l'ensemble des relations affines engendrées par la composition de deux relations $\mathcal{R}^F_{(n_1,\varphi_1,d_1)}$ et $\mathcal{R}^F_{(n_2,\varphi_2,d_2)}$ est un problème complexe ; nous nous sommes limités à mettre en évidence un sous-ensemble de cet ensemble et avons dégagé les concepts et les problèmes rencontrés. De manière similaire peuvent être abordés les cas de l'union, l'intersection et la différence de relations affines de type $\mathcal{R}^F_{(n,\varphi,d)}$; nous avons omis d'aborder ces problèmes, en nous limitant à un contexte de preuve restreint et qui s'est montré suffisant pour les cas de simulation que nous avons considéré.

7.1.5.1 Inverse d'une relation affine

L'inverse d'une relation arbitraire $\mathcal{R}$ est la relation $\mathcal{R}^{-1}$ telle que [San80] :

$$x\mathcal{R}^{-1}y \Leftrightarrow y\mathcal{R}x,\ \forall\ x, y$$

Dans le cas des relations affines, la définition antérieure devient :

Définition 7 *L'inverse de la relation affine* $\mathcal{R}^F_{(n,\varphi,d)}$ *est la relation* $(\ \mathcal{R}^F_{(n,\varphi,d)}\ )^{-1}$ *telle que*

$$x\ \mathcal{R}^F_{(n,\varphi,d)}\ y\ \Leftrightarrow\ y\ (\ \mathcal{R}^F_{(n,\varphi,d)}\ )^{-1}\ x\ ,\ \ \forall\ x, y \in A_1$$

□

L'inverse d'une relation affine est aussi affine, comme le démontre le lemme suivant :

Lemme 5 *Pour tout flot $F \in \mathcal{F}_{A_1}$,*

$$\left(\mathcal{R}^F_{(n,\varphi,d)} \right)^{-1} = \mathcal{R}^F_{(d,-\varphi,n)}$$

Preuve Soit $(x,y) \in (\mathcal{R}^F_{(n,\varphi,d)})^{-1}$. Il résulte $(y,x) \in \mathcal{R}^F_{(n,\varphi,d)}$.

D'après la définition d'une relation affine, il existe $g : \mathbb{N} \rightarrow \mathbb{N}$, croissante sur $\mathbb{N}$ et strictement croissante sur $f_2(\mathbb{N}) \cup f_1(\mathbb{N})$ telle que

$$F = (\ (g \circ f_2) \uparrow \Pi_{\{y\}}(F)\ ) \cdot (\ (g \circ f_1) \uparrow \Pi_{\{x\}}(F)\ ) \cdot F$$

et

$$\forall\ t, s \in \mathbb{N},\ f_2(t) = f_1(s) \ \Rightarrow\ y_{(F)}(t) = x_{(F)}(s)$$

avec $f_2 = \lambda\ t.(nt + max(0, -\varphi)), f_1 = \lambda\ t.(dt + max(0, \varphi))$.

Par permutation des termes dans la définition précédente on obtient :

$$F = (\ (g \circ f_1) \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ (g \circ f_2) \uparrow \Pi_{\{y\}}(F)\ ) \cdot F$$

et

$$\forall\ t, s \in \mathbb{N},\ f_1(t) = f_2(s) \ \Rightarrow\ x_{(F)}(t) = y_{(F)}(s)$$

avec $f_1 = \lambda\ t.(dt + max(0, -(-\varphi))), f_2 = \lambda\ t.(nt + max(0, -\varphi))$ et $g : \mathbb{N} \rightarrow \mathbb{N}$, croissante sur $\mathbb{N}$ et strictement croissante sur $f_1(\mathbb{N}) \cup f_2(\mathbb{N})$. En conclusion $(x,y) \in \mathcal{R}^F_{(d,-\varphi,n)}$.

De manière similaire, on démontre $(x,y) \in \mathcal{R}^F_{(d,-\varphi,n)}$ implique $(x,y) \in (\mathcal{R}^F_{(n,\varphi,d)})^{-1}$.

Donc, $\forall\ (x,y) \in (\mathcal{R}^F_{(n,\varphi,d)})^{-1}$, il résulte $(x,y) \in (\mathcal{R}^F_{(d,-\varphi,n)})$ et vice versa. $\square$

7.1.5.2 Composition de relations affines

La composition $\mathcal{R}_1\ *\ \mathcal{R}_2$ des relations arbitraires $\mathcal{R}_1$ et $\mathcal{R}_2$ est définie par la relation suivante [San80] :

$$x(\mathcal{R}_1\ *\ \mathcal{R}_2)z \ \Leftrightarrow\ \exists y, (x\mathcal{R}_1 y) \text{ et } (y\mathcal{R}_2 z)$$

De manière similaire on définit la composition de relations affines :

Définition 8 *Pour tout flot $F \in \mathcal{F}_{A_1}$, la composition des relations $\mathcal{R}^F_{(n_1,\varphi_1,d_1)}$ et $\mathcal{R}^F_{(n_2,\varphi_2,d_2)}$, notée $*$, est définie comme suit :*

$$\mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ *\ \mathcal{R}^F_{(n_2,\varphi_2,d_2)} = \{(x,z) \in A_1 \times A_1\ /\ \exists\ y \in A_1, (x,y) \in \mathcal{R}^F_{(n_1,\varphi_1,d_1)} \textit{ et } (y,z) \in \mathcal{R}^F_{(n_2,\varphi_2,d_2)}\}$$

$\square$

Remarque :

1) Pour tout ensemble de valeurs pour les paramètres (n_1, φ_1, d_1) et (n_2, φ_2, d_2), il existe toujours $F \in \mathcal{F}_{\{x,y,z\}}$ tel que $x\ \mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ y$ et $y\ \mathcal{R}^F_{(n_2,\varphi_2,d_2)}\ z$:

$$\mathbf{P} = \{F \in \mathcal{F}_{\{x,y,z\}}\ |\ x\ \mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ y \text{ et } y\ \mathcal{R}^F_{(n_2,\varphi_2,d_2)}\ z\ \} \neq \emptyset$$

Intuitivement, cette propriété affirme qu'il est possible de construire l'horloge H_y de y comme une transformation affine de paramètres (n_1, φ_1, d_1) sur l'horloge H_x de x et ensuite appliquer une transformation affine de paramètres (n_2, φ_2, d_2) sur H_y afin d'obtenir H_z. $\square$

Soit $F \in \mathcal{F}_{\{x,y,z\}}$ tel que $x \; \mathcal{R}^F_{(n_1,\varphi_1,d_1)} \; y$ et $y \; \mathcal{R}^F_{(n_2,\varphi_2,d_2)} \; z$. Évidemment $x \; \mathcal{R}^F_{(n_1,\varphi_1,d_1)} * \mathcal{R}^F_{(n_2,\varphi_2,d_2)} z$. On se pose la question si la composition de relations affines produit toujours une relation affine ; en d'autres mots, est-ce qu'il existe toujours (n, φ, d) tel que $\mathcal{R}^F_{(n_1,\varphi_1,d_1)} * \mathcal{R}^F_{(n_2,\varphi_2,d_2)} = \mathcal{R}^F_{(n,\varphi,d)}$? Or, compte tenu des valeurs des paramètres (n_1, φ_1, d_1) et (n_2, φ_2, d_2), x et z ne sont pas toujours en relation affine dans le flot F. Nous présentons sur les Figures 7.11 et 7.12 deux flots $F_1, F_2 \in \mathcal{F}_{\{x,y,z\}}$ tels que $x \; \mathcal{R}^{F_i}_{(2,1,2)} \; y$, $y \; \mathcal{R}^{F_i}_{(2,1,2)} \; z$, pour $i \in \{1, 2\}$, et $x \; \mathcal{R}^{F_1}_{(2,1,2)} \; z$, mais x et z ne sont reliés par *aucune* relation affine dans le flot F_2.

t		0	1	2	3	4	5	6	7	8	9	10	11	12
F_1	x	x_1	$\perp$	$\perp$	x_2	$\perp$	$\perp$	x_3	$\perp$	$\perp$	x_4	$\perp$	$\perp$	....
	y	$\perp$	y_1	$\perp$	$\perp$	y_2	$\perp$	$\perp$	y_3	$\perp$	$\perp$	y_4	$\perp$	....
	z	$\perp$	$\perp$	z_1	$\perp$	$\perp$	z_2	$\perp$	$\perp$	z_3	$\perp$	$\perp$	z_4	....

FIG. 7.11 – $x \; \mathcal{R}^{F_1}_{(2,1,2)} \; z$.

t		0	1	2	3	4	5	6	7	8	9	10	11	12
F_2	x	x_1	$\perp$	$\perp$	x_2	$\perp$	x_3	$\perp$	$\perp$	$\perp$	x_4	$\perp$	$\perp$	....
	y	$\perp$	y_1	$\perp$	$\perp$	y_2	$\perp$	$\perp$	y_3	$\perp$	$\perp$	y_4	$\perp$	....
	z	$\perp$	$\perp$	z_1	$\perp$	$\perp$	$\perp$	z_2	$\perp$	z_3	$\perp$	$\perp$	z_4	....

FIG. 7.12 – *Les variables x et z ne sont pas en relation affine.*

Un flot $F \in \mathcal{F}_{\{x,y,z\}}$ tel que $x \; \mathcal{R}^F_{(n_1,\varphi_1,d_1)} \; y$ et $y \; \mathcal{R}^F_{(n_2,\varphi_2,d_2)} \; z$ peut s'écrire

$$\begin{aligned} F \;\; = \;\; & (\; (g_1 \circ f^1_x) \uparrow \Pi_{\{x\}}(F)\;) \cdot (\; (g_1 \circ f^1_y) \uparrow \Pi_{\{y\}}(F)\;) \cdot \\ & \cdot (\; (g_2 \circ f^2_y) \uparrow \Pi_{\{y\}}(F)\;) \cdot (\; (g_2 \circ f^2_z) \uparrow \Pi_{\{z\}}(F)\;) \cdot F \end{aligned}$$

avec $f^1_x = \lambda \; t.(n_1 t + max(0, -\varphi_1))$, $f^1_y = \lambda \; t.(d_1 t + max(0, \varphi_1))$, $f^2_y = \lambda \; t.(n_2 t + max(0, -\varphi_2))$, $f^2_z = \lambda \; t.(d_2 t + max(0, \varphi_2))$ et $g_1, g_2 : \mathbb{N} \to \mathbb{N}$, croissantes sur $\mathbb{N}$ et strictement croissantes sur $f^1_x(\mathbb{N}) \cup f^1_y(\mathbb{N})$, respectivement $f^2_y(\mathbb{N}) \cup f^2_z(\mathbb{N})$, satisfaisant $g_1 \circ f^1_y = g_2 \circ f^2_y$ (les contraintes sur les valeurs sont implicites).

Les contraintes existant sur les fonctions g_1, g_2, et en conséquence la dimension de l'espace de ces fonctions, varient en fonction des paramètres (n_1, φ_1, d_1) et (n_2, φ_2, d_2) ; pour certaines valeurs de ces triplets il existe toujours une relation affine entre x et z pour toute paire de fonctions (g_1, g_2).

En premier lieu, nous mettons en évidence des triplets de paramètres (n_1, φ_1, d_1) et (n_2, φ_2, d_2) tels qu'il existe (n, φ, d) avec la propriété

$$\mathcal{R}^F_{(n_1,\varphi_1,d_1)} \ * \ \mathcal{R}^F_{(n_2,\varphi_2,d_2)} = \mathcal{R}^F_{(n,\varphi,d)}$$

pour tout flot $F \in \mathcal{F}_{\{x,y,z\}}$ avec la propriété $x \ \mathcal{R}^F_{(n_1,\varphi_1,d_1)} \ y$ et $y \ \mathcal{R}^F_{(n_2,\varphi_2,d_2)} \ z$.

Pour le cas général, nous nous proposons de caractériser l'ensemble des relations affines qui peuvent être induites entre x et z dans un flot $F \in \mathcal{F}_{\{x,y,z\}}$ par les relations $\mathcal{R}^F_{(n_1,\varphi_1,d_1)}$ et $\mathcal{R}^F_{(n_2,\varphi_2,d_2)}$ existant entre x, y et y, z.

Composition de relations affines – analyse de cas On considère que n_1, φ_1, d_1 et n_2, φ_2, d_2 satisfont le système de contraintes (C_c) si leurs valeurs vérifient les conditions (1) ou (2) ci-dessous:

$$(C_c) = (1) \text{ ou } (2), \text{ avec } \quad (1) \quad \begin{cases} d_1 = 1 \\ \varphi_1 \leq 0 \end{cases} \quad \text{et} \quad (2) \quad \begin{cases} n_2 = 1 \\ \varphi_2 \in \mathbb{N} \end{cases}$$

Proposition 2 *$\forall \ F \in \mathcal{F}_{\{x,y,z\}}$ satisfaisant $x \ \mathcal{R}^F_{(n_1,\varphi_1,d_1)} \ y$ et $y \ \mathcal{R}^F_{(n_2,\varphi_2,d_2)} \ z$, la relation suivante est valide :*

$$(C_c) \quad \Rightarrow \quad \mathcal{R}^F_{(n_1,\varphi_1,d_1)} \ * \ \mathcal{R}^F_{(n_2,\varphi_2,d_2)} = \mathcal{R}^F_{(n_1 n_2, n_2\varphi_1 + d_1\varphi_2, d_1 d_2)}$$

Preuve Nous prouvons l'identité ci-dessus pour le cas $d_1 = 1, \varphi_1 \leq 0$:

$$\mathcal{R}^F_{(n_1,\varphi_1,1)} \ * \ \mathcal{R}^F_{(n_2,\varphi_2,d_2)} = \mathcal{R}^F_{(n_1 n_2, n_2\varphi_1 + \varphi_2, d_2)}, \ \forall \ \varphi_1 \leq 0$$

(la preuve pour le deuxième cas est similaire).

Comme $(x, y) \in \mathcal{R}^F_{(n_1,\varphi_1,1)}$ et $(y, z) \in \mathcal{R}^F_{(n_2,\varphi_2,d_2)}$, alors $(x, z) \in \mathcal{R}^F_{(n_1,\varphi_1,1)} \ * \ \mathcal{R}^F_{(n_2,\varphi_2,d_2)}$ et il existe $g_1, g_2 : \mathbb{N} \ \rightarrow \ \mathbb{N}$, croissantes sur $\mathbb{N}$ et strictement croissantes sur $f^1_x(\mathbb{N}) \cup f^1_y(\mathbb{N})$ et respectivement $f^2_y(\mathbb{N}) \cup f^2_z(\mathbb{N})$, telles que

$$\begin{array}{l} F = (\ (g_1 \circ f^1_x) \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ (g_1 \circ f^1_y) \uparrow \Pi_{\{y\}}(F)\ )\cdot \\ \cdot(\ (g_2 \circ f^2_y) \uparrow \Pi_{\{y\}}(F)\ ) \cdot (\ (g_2 \circ f^2_z) \uparrow \Pi_{\{z\}}(F)\ ) \cdot F \\ \forall \ t, s \in \mathbb{N}, \ f^1_x(t) = f^1_y(s) \ \Rightarrow \ x_{(F)}(t) = y_{(F)}(s) \\ \forall \ t, s \in \mathbb{N}, \ f^2_y(t) = f^2_z(s) \ \Rightarrow \ y_{(F)}(t) = z_{(F)}(s) \end{array}$$

et

$$g_1 \circ f^1_y = g_2 \circ f^2_y$$

avec $f^1_x = \lambda \ t.(n_1 t - \varphi_1)$, $f^1_y = \lambda \ t.(t)$, $f^2_y = \lambda \ t.(n_2 t + max(0, -\varphi_2))$, $f^2_z = \lambda \ t.(d_2 t + max(0, \varphi_2))$.

$g_1 \circ f^1_y = g_2 \circ f^2_y$ et l'expression de la fonction f^1_y conduisent à $g_1 = g_2 \circ f^2_y$. La relation entre x et z dans F sera caractérisée par les fonctions de densification suivantes :

$$\begin{array}{l} g_1 \circ f^1_x = g_2 \circ (f^2_y \circ f^1_x) = \lambda \ t.(g_2(n_1 n_2 t - n_2\varphi_1 + max(0, -\varphi_2))) \\ g_2 \circ f^2_z = \lambda \ t.(g_2(d_2 t + max(0, \varphi_2))) \end{array}$$

Soit les fonctions strictement croissantes suivantes :

$$\begin{array}{l} f_x, f_z : \mathbb{N} \ \rightarrow \ \mathbb{N}, \\ f_x = \lambda \ t.(n_1 n_2 t - n_2\varphi_1 + max(0, -\varphi_2)), \quad f_z = \lambda \ t.(d_2 t + max(0, \varphi_2)) \end{array}$$

Les fonctions de densification $g_1 \circ f^1_x$ et $g_2 \circ f^2_z$ associées à x et z dans le flot F peuvent se réécrire en :

$$\begin{array}{l} g_1 \circ f^1_x = g_2 \circ f_x \\ g_2 \circ f^2_z = g_2 \circ f_z \end{array}$$

En ce qui concerne les valeurs de x et y dans le flot F, on va démontrer :

$$\forall t, v \in \mathbb{N},\ f_x(t) = f_z(v) \ \Rightarrow\ x_{(F)}(t) = z_{(F)}(v)$$

Soit $t, v \in \mathbb{N},\ f_x(t) = f_z(v)$. Alors $n_1 n_2 t = d_2 v + \varphi_2 + n_2\varphi_1$. Comme $\varphi_1 \leq 0$, alors $(n_1 t - \varphi_1) \in \mathbb{N}$ et on note $s = n_1 t - \varphi_1,\ s \in \mathbb{N}$. En multipliant l'inégalité antérieure par n_2 on obtient $n_1 n_2 t = n_2 s + n_2\varphi_1$. Les identités $n_1 n_2 t = d_2 v + \varphi_2 + n_2\varphi_1$ et $n_1 n_2 t = n_2 s + n_2 \varphi_1$ impliquent $d_2 v + \varphi_2 + n_2\varphi_1 = n_2 s + n_2\varphi_1$, qui est équivalent à $n_2 s = d_2 v + \varphi_2$.

En utilisant maintenant les propriétés sur les valeurs de x et y, respectivement y et z dans le flot F, on obtient :

$$\left.\begin{array}{l} n_1 t - \varphi_1 = s \Rightarrow f_x^1(t) = f_y^1(s) \ \Rightarrow\ x_{(F)}(t) = y_{(F)}(s) \\ n_2 s = d_2 v + \varphi_2 \Rightarrow f_y^2(s) = f_z^2(v) \ \Rightarrow\ y_{(F)}(s) = z_{(F)}(v) \end{array}\right\} \Rightarrow x_{(F)}(t) = z_{(F)}(v)$$

En conclusion $t, v \in \mathbb{N},\ n_1 n_2 t = d_2 v + \varphi_2 + n_2\varphi_1$, on a mis en évidence $s \in \mathbb{N}$ tel que $n_1 t - \varphi_1 = s$ et $n_2 s = d_2 v + \varphi_2$. A partir de ces deux dernières identités on a obtenu $x_{(F)}(t) = y_{(F)}(s)$ et $y_{(F)}(s) = z_{(F)}(v)$, qui ont induit $x_{(F)}(t) = z_{(F)}(v)$. Donc :

$$\forall t, v \in \mathbb{N},\ f_x(t) = f_z(v) \ \Rightarrow\ x_{(F)}(t) = z_{(F)}(s)$$

On a ainsi prouvé que $(x,z) \in \mathcal{R}^F_{((N,\Phi_1),(D,\Phi_2))}$, avec $N = n_1 n_2, \Phi_1 = -n_2\varphi_1 + max(0, -\varphi_2), D = d_2, \Phi_2 = max(0, \varphi_2)$. En utilisant le Corollaire 2, il résulte $(x,z) \in \mathcal{R}^F_{(N,\Phi,D)}$, avec $\Phi = \Phi_2 - \Phi_1$, i.e. $(x,z) \in \mathcal{R}^F_{(n_1 n_2, \varphi_2 + n_2\varphi_1, d_2)}$.

On a obtenu pour les variables x et z les conditions suivantes :

$$\begin{array}{l} (x,z) \in \mathcal{R}^F_{(n_1,\varphi_1,1)} \ * \ \mathcal{R}^F_{(n_2,\varphi_2,d_2)} \\ (x,z) \in \mathcal{R}^F_{(n_1 n_2, n_2\varphi_1+\varphi_2, d_2)} \end{array}$$

Nous considérons le cas "général" $\mathcal{R}^F_{(n_1 n_2, n_2\varphi_1+\varphi_2, d_2)} = \{(x,z)\}$ (dans le cas où $\mathcal{R}^F_{(n_1 n_2, n_2\varphi_1+\varphi_2, d_2)}$ contient au moins un autre élément différent de (x,z), on obtient des contraintes supplémentaires sur les valeurs des paramètres $(n_1, \varphi_1, 1)$ et (n_2, φ_2, d_2) ; les nouvelles valeurs de $(n_1, \varphi_1, 1)$ et (n_2, φ_2, d_2) génèrent des cas particuliers du cas général traité).

On a dans cette situation

$$\mathcal{R}^F_{(n_1 n_2, n_2\varphi_1+\varphi_2, d_2)} \subseteq \mathcal{R}^F_{(n_1,\varphi_1,1)} \ * \ \mathcal{R}^F_{(n_2,\varphi_2,d_2)}$$

et comme pour toute paire $(u,v) \in \mathcal{R}^F_{(n_1,\varphi_1,1)} \ * \ \mathcal{R}^F_{(n_2,\varphi_2,d_2)}$ il résulte $(u,v) \in \mathcal{R}^F_{(n_1 n_2, n_2\varphi_1+\varphi_2, d_2)}$ – la preuve suit un schéma similaire à celui de la démonstration précédente pour $(x,z) \in \mathcal{R}^F_{(n_1 n_2, n_2\varphi_1+\varphi_2, d_2)}$ – il est immédiat que :

$$\mathcal{R}^F_{(n_1,\varphi_1,1)} \ * \ \mathcal{R}^F_{(n_2,\varphi_2,d_2)} = \mathcal{R}^F_{(n_1 n_2, n_2\varphi_1+\varphi_2, d_2)}$$

$\square$

Sur la Figure 7.13, nous avons représenté un flot $F \in \mathcal{F}_{\{x,y,z\}}$ tel que $x\ \mathcal{R}^F_{(3,-1,1)}\ y$, $y\ \mathcal{R}^F_{(3,1,3)}\ z$ et $x\ \mathcal{R}^F_{(9,-2,3)}\ y$ (les fonctions f_y^2, f_x^1 et f_z^2 correspondent à la Proposition 2 précédente et les valeurs de x et y vérifient $x_i = y_{3i-1}, \forall i \in \mathbb{N}^*$).

Le lemme suivant résulte directement de la Proposition 2.

Lemme 6 *$\forall\ F \in \mathcal{F}_{\{x,y,z\}}$ satisfaisant $x\ \mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ y$ et $y\ \mathcal{R}^F_{(1,\varphi_2,d_2)}\ z$, la relation suivante est valide :*

$$\mathcal{R}^F_{(n_1,\varphi_1,d_1)} \ * \ \mathcal{R}^F_{(1,\varphi_2,d_2)} = \mathcal{R}^F_{(n_1, \varphi_1 + d_1\varphi_2, d_1 d_2)}$$

$\forall n_1, d_1, d_2 \in \mathbb{N}^*$, $\varphi_1 \in \mathbb{Z}$ et $\varphi_2 \in \mathbb{N}$.

t	0	1	2	3	4	5	6	7	8	9	10	11	12
$(g_2 \circ f_y^2 \circ f_x^{1})(t)$			$g_2(3)$						$g_2(12)$				
$(g_2 \circ f_y^2)(t)$	$g_2(0)$		$g_2(3)$		$g_2(6)$		$g_2(9)$		$g_2(12)$		$g_2(15)$		
$(g_2 \circ f_z^2)(t)$		$g_2(1)$		$g_2(4)$		$g_2(7)$		$g_2(10)$		$g_2(13)$		$g_2(16)$	

F	x	$\perp$	$\perp$	x_1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	x_2	$\perp$	$\perp$	$\perp$	....
	y	y_1	$\perp$	y_2	$\perp$	y_3	$\perp$	y_4	$\perp$	y_5	$\perp$	y_6	$\perp$	....
	z	$\perp$	z_1	$\perp$	z_2	$\perp$	z_3	$\perp$	z_4	$\perp$	z_5	$\perp$	z_6	....

FIG. 7.13 – *Illustration de la Proposition 2.*

Preuve En appliquant Proposition 2 pour $n_2 = 1$ et $\varphi_2 \in \mathbb{N}$. $\square$

Composition de relations affines – problème général Dans le cas général où on manipule des valeurs arbitraires pour les paramètres (n_1, φ_1, d_1) et (n_2, φ_2, d_2), on se propose de caractériser l'ensemble des relations affines $\mathcal{R}^F_{(n,\varphi,d)}$ telles qu'il existe $F \in \mathbf{P}$ avec la propriété $x\ \mathcal{R}^F_{(n,\varphi,d)}\ z$, où $\mathbf{P}$ est défini comme suit :

$$\mathbf{P} \quad = \quad \{F \in \mathcal{F}_{\{x,y,z\}} \mid \ x\ \mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ y \ \text{ et } \ y\ \mathcal{R}^F_{(n_2,\varphi_2,d_2)}\ z\ \}$$

Connaître cet ensemble permettrait de vérifier l'identifiabilité de z et v dans un processus $\mathbf{P}'$ tel que $x\ \mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ y$, $y\ \mathcal{R}^F_{(n_2,\varphi_2,d_2)}\ z$ et $x\ \mathcal{R}^F_{(n_3,\varphi_3,d_3)}\ v$, pour tout flot F appartenant à $\mathbf{P}'$. Il suffirait de tester qu'il existe (n, φ, d) et $F \in \mathbf{P}'$ tels que $x\ \mathcal{R}^F_{(n,\varphi,d)}\ z$ et $\mathcal{R}^F_{(n_3,\varphi_3,d_3)} = \mathcal{R}^F_{(n,\varphi,d)}$. Or, pour le processus $\mathbf{P}$ introduit précédemment, nous allons définir un sous-ensemble de triplets (n, φ, d) tels que $\exists F \in \mathbf{P}$, $x\ \mathcal{R}^F_{(n,\varphi,d)}\ z$. Dans cette situation, une condition suffisante pour que z et v soient identifiables dans $\mathbf{P}'$ serait qu'il existe (n, φ, d) dans le sous-ensemble de triplets calculé afin que $\mathcal{R}^F_{(n_3,\varphi_3,d_3)} = \mathcal{R}^F_{(n,\varphi,d)}$, pour $F \in \mathbf{P}'$. Vérifier l'existence de (n, φ, d) dans le sous-ensemble de triplets qu'on propose est un problème NP-complet. Des heuristiques peuvent être mises en place basées sur une recherche dans un espace restreint autour d'une solution initiale.

Nous démontrons que pour tout ensemble de triplets $\{(n'_1, \varphi'_1, d'_1), (n'_2, \varphi'_2, d'_2)\}$ tels que $d'_1 = n'_2$ et $(n'_1, \varphi'_1, d'_1) \sim (n_1, \varphi_1, d_1)$, $(n'_2, \varphi'_2, d'_2) \sim (n_2, \varphi_2, d_2)$, il existe $F \in \mathbf{P}$, $x\ \mathcal{R}^F_{(n'_1 n'_2, n'_2\varphi'_1 + d'_1\varphi'_2, d'_1 d'_2)}\ z$. Nous utilisons dans la preuve le résultat donné par le Lemme 7 ci-dessous : pour tout ensemble de triplets $\{(n'_1, \varphi'_1, d'_1), (n'_2, \varphi'_2, d'_2)\}$ tels que $d'_1 = n'_2$, $\exists F \in \mathcal{F}_{\{x,y,z\}}$, $x\ \mathcal{R}^F_{(n'_1,\varphi'_1,d'_1)}\ y$, $y\ \mathcal{R}^F_{(n'_2,\varphi'_2,d'_2)}\ z$ et $x\ \mathcal{R}^F_{(n'_1 n'_2, n'_2\varphi'_1 + d'_1\varphi'_2, d'_1 d'_2)}\ z$.

Lemme 7 *Soit les triplets (n'_1, φ'_1, d'_1) et (n'_2, φ'_2, d'_2) contraints uniquement par $d'_1 = n'_2$. On considère les processus suivants :*

$$\begin{aligned}
\mathbf{P}' &= \{F \in \mathcal{F}_{\{x,y,z\}} \mid \ x\ \mathcal{R}^F_{(n'_1,\varphi'_1,d'_1)}\ y \ \textit{ et } \ y\ \mathcal{R}^F_{(n'_2,\varphi'_2,d'_2)}\ z\ \} \\
\mathbf{Q}' &= \{F \in \mathcal{F}_{\{x,y,z\}} \mid \ x\ \mathcal{R}^F_{(n'_1,\varphi'_1,d'_1)}\ y\ ,\ y\ \mathcal{R}^F_{(n'_2,\varphi'_2,d'_2)}\ z \\
&\qquad\qquad \textit{et } \ x\ \mathcal{R}^F_{(n'_1 n'_2, n'_2\varphi'_1 + d'_1\varphi'_2, d'_1 d'_2)}\ z\ \}
\end{aligned}$$

Alors :

1) $\mathbf{Q}' \neq \emptyset$. *2)* $\mathbf{Q}' \subseteq \mathbf{P}'$.

Preuve L'ensemble des flots sur $\{x, y, z\}$ dans lesquels x et y d'une part, y et z d'autre part sont en relation affine de paramètres respectifs (n'_1, φ'_1, d'_1) et (n'_2, φ'_2, d'_2) est non vide, c'est-à-dire le processus $\mathbf{P}'$ est non vide :

$$\mathbf{P}' = \{F \in \mathcal{F}_{\{x,y,z\}} \mid \; x \; \mathcal{R}^F_{(n'_1,\varphi'_1,d'_1)} \; y \;\text{ et }\; y \; \mathcal{R}^F_{(n'_2,\varphi'_2,d'_2)} \; z \;\} \neq \emptyset$$

Pour tout flot $F \in \mathbf{P}'$, il existe $g_1, g_2 : \mathbb{N} \to \mathbb{N}$, croissantes sur $\mathbb{N}$ et strictement croissantes sur $f^1_x(\mathbb{N}) \cup f^1_y(\mathbb{N})$, respectivement $f^2_y(\mathbb{N}) \cup f^2_z(\mathbb{N})$, telles que F peut s'écrire

$$\begin{array}{rcl} F & = & (\;(g_1 \circ f^1_x) \uparrow \Pi_{\{x\}}(F)\;) \cdot (\;(g_1 \circ f^1_y) \uparrow \Pi_{\{y\}}(F)\;) \cdot \\ & & \cdot(\;(g_2 \circ f^2_y) \uparrow \Pi_{\{y\}}(F)\;) \cdot (\;(g_2 \circ f^2_z) \uparrow \Pi_{\{z\}}(F)\;) \end{array}$$

et

$$\begin{array}{l} \forall t, s \in \mathbb{N}, \;\; f^1_x(t) = f^1_y(s) \Rightarrow x_{(F)}(t) = y_{(F)}(s) \\ \forall t, s \in \mathbb{N}, \;\; f^2_y(t) = f^2_z(s) \Rightarrow y_{(F)}(t) = z_{(F)}(s) \end{array}$$

avec $f^1_x = \lambda\, t.(n'_1 t + max(0, -\varphi'_1) + c'_1)$, $f^1_y = \lambda\, t.(d'_1 t + max(0, \varphi'_1) + c'_1)$, $f^2_y = \lambda\, t.(n'_2 t + max(0, -\varphi'_2) + c'_2)$, $f^2_z = \lambda\, t.(d'_2 t + max(0, \varphi'_2) + c'_2)$ et $c'_1, c'_2 \in \mathbb{Z}$ choisis tels que les relations suivantes soient vérifiées : $max(0, -\varphi'_1) + c'_1 \geq 0$, $max(0, \varphi'_1) + c'_1 = max(0, -\varphi'_2) + c'_2 \geq 0$ et $max(0, \varphi'_2) + c'_2 \geq 0$.

On observe l'égalité de f^1_y et f^2_y :

$$\left.\begin{array}{r} d'_1 = n'_2 \\ max(0, \varphi'_1) + c'_1 = max(0, -\varphi'_2) + c'_2 \end{array}\right\} \;\Rightarrow\; \begin{array}{l} d'_1 t + max(0, \varphi'_1) + c'_1 = \\ = n'_2 t + max(0, -\varphi'_2) + c'_2, \; \forall\, t \in \mathbb{N} \end{array}$$

$$\Rightarrow \;\; f^1_y(t) = f^2_y(t), \; \forall\, t \in \mathbb{N}$$

Conformément au lemme de recomposition de flots, la fonction de densification associée à y dans F est unique :

$$\begin{array}{rcl} (g_1 \circ f^1_y) = (g_2 \circ f^2_y) & \Leftrightarrow & (g_1 \circ f^1_y)(t) = (g_2 \circ f^2_y)(t), \; \forall\, t \in \mathbb{N} \\ & \Leftrightarrow & (g_1 \circ f^1_y)(\mathbb{N}) = (g_2 \circ f^2_y)(\mathbb{N}) \end{array}$$

Les fonctions g_1, g_2 doivent coïncider sur un sous-domaine de leur domaine de définition. Les fonctions g_1, g_2 pour lesquelles il existe $g : \mathbb{N} \to \mathbb{N}$ croissante sur $\mathbb{N}$ et strictement croissante sur $f^1_x(\mathbb{N}) \cup f^1_y(\mathbb{N}) \cup f^2_z(\mathbb{N})$, $g_1 = g_2 = g$, satisfont cette condition.

Tout $F \in \mathbf{P}'$ tel que $g_1 = g_2 = g$ peut se réécrire en :

$$\begin{array}{rcl} F & = & (\;(g \circ f^1_x) \uparrow \Pi_{\{x\}}(F)\;) \cdot (\;(g \circ f^1_y) \uparrow \Pi_{\{y\}}(F)\;) \cdot \\ & & \cdot(\;(g \circ f^2_y) \uparrow \Pi_{\{y\}}(F)\;) \cdot (\;(g \circ f^2_z) \uparrow \Pi_{\{z\}}(F)\;) \end{array}$$

et il résulte :

$$\begin{array}{rcl} (x, z) \in \mathcal{R}^F_{(f^1_x, f^2_z)} & = & \mathcal{R}^F_{((n'_1, max(0, -\varphi'_1) + c'_1), (d'_2, max(0, \varphi'_2) + c'_2))} \\ & = & \mathcal{R}^F_{(n'_1, \varphi'_1 + \varphi'_2, d'_2)} \\ & & (\text{car } \; max(0, \varphi'_1) + c'_1 = max(0, -\varphi'_2) + c'_2) \\ & = & \mathcal{R}^F_{(n'_1 n'_2, n'_2 \varphi'_1 + d'_1 \varphi'_2, d'_1 d'_2)} \\ & & (\text{car } \; n'_2 = d'_1) \end{array}$$

En conclusion : $(\forall F)\ ((F \in \mathbf{P}')$ et $(g_1 = g_2 = g) \Rightarrow (F \in \mathbf{Q}'))$. Il résulte $\mathbf{Q}' \neq \emptyset$.
De plus : $\mathbf{P}' \mid \mathbf{Q}' = \mathbf{Q}' \Rightarrow \Pi_{\{x,y,z\}}(\mathbf{Q}') \subseteq \mathbf{P}' \Rightarrow \mathbf{Q}' \subseteq \mathbf{P}'$ (puisque $\Pi_{\{x,y,z\}}(\mathbf{Q}') = \mathbf{Q}'$).

□

A partir du résultat précédent, nous démontrons que l'ensemble des triplets (n, φ, d) pour lesquels il existe $F \in \mathbf{P}$ dans lequel x et z sont en relation affine de paramètres (n, φ, d) contient l'ensemble de tous les triplets $(n'_1n'_2, n'_2\varphi'_1 + d'_1\varphi'_2, d'_1d'_2)$, avec $(n'_1, \varphi'_1, d'_1) \sim (n_1, \varphi_1, d_1)$, $(n'_2, \varphi'_2, d'_2) \sim (n_2, \varphi_2, d_2)$ et $d'_1 = n'_2$.

Théorème 3 *Étant donné le processus non vide*

$$\mathbf{P} \quad = \quad \{F \in \mathcal{F}_{\{x,y,z\}} \mid \ x\ \mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ y \ \ et \ \ y\ \mathcal{R}^F_{(n_2,\varphi_2,d_2)}\ z\ \}$$

pour tout ensemble de triplets $\{(n'_1, \varphi'_1, d'_1), (n'_2, \varphi'_2, d'_2)\}$ *satisfaisant*

$$\begin{array}{l} (n'_1, \varphi'_1, d'_1) \sim (n_1, \varphi_1, d_1) \\ (n'_2, \varphi'_2, d'_2) \sim (n_2, \varphi_2, d_2) \\ d'_1 = n'_2 \end{array}$$

on note $\mathbf{Q}'$ *le processus non vide défini par*

$$\begin{array}{rcl} \mathbf{Q}' & = & \{F \in \mathcal{F}_{\{x,y,z\}} \mid \ x\ \mathcal{R}^F_{(n'_1,\varphi'_1,d'_1)}\ y\ ,\ y\ \mathcal{R}^F_{(n'_2,\varphi'_2,d'_2)}\ z \\ & & \qquad\qquad et \ \ x\ \mathcal{R}^F_{(n'_1n'_2, n'_2\varphi'_1+d'_1\varphi'_2, d'_1d'_2)}\ z\ \} \end{array}$$

Alors $\mathbf{Q}' \subseteq \mathbf{P}$.

Preuve On note T l'ensemble de triplets (n'_1, φ'_1, d'_1) et (n'_2, φ'_2, d'_2) satisfaisant les conditions de l'hypothèse :

$$T = \{(n'_1, \varphi'_1, d'_1), (n'_2, \varphi'_2, d'_2) \mid (n'_1, \varphi'_1, d'_1) \sim (n_1, \varphi_1, d_1), (n'_2, \varphi'_2, d'_2) \sim (n_2, \varphi_2, d_2) \text{ et } d'_1 = n'_2\}$$

T est non vide puisque $(n_1n_2, n_2\varphi_1, d_1n_2), (n_2d_1, d_1\varphi_2, d_1d_2) \in T$.
Soit $\mathbf{P}'$ définit par

$$\mathbf{P}' \quad = \quad \{F \in \mathcal{F}_{\{x,y,z\}} \mid \ x\ \mathcal{R}^F_{(n'_1,\varphi'_1,d'_1)}\ y \ \text{ et } \ y\ \mathcal{R}^F_{(n'_2,\varphi'_2,d'_2)}\ z\ \}$$

Cf. au lemme précédent, pour toute paire de triplets (n'_1, φ'_1, d'_1), (n'_2, φ'_2, d'_2) appartenant à T, $\mathbf{Q}' \neq \emptyset$ et $\mathbf{Q}' \subseteq \mathbf{P}'$.
De plus, $\forall F \in \mathbf{P}'$:

$$\left.\begin{array}{r} x\ \mathcal{R}^F_{(n'_1,\varphi'_1,d'_1)}\ y \ \text{ et} \\ y\ \mathcal{R}^F_{(n'_2,\varphi'_2,d'_2)}\ z \\ (n'_1, \varphi'_1, d'_1) \sim (n_1, \varphi_1, d_1) \\ (n'_2, \varphi'_2, d'_2) \sim (n_2, \varphi_2, d_2) \end{array}\right\} \Rightarrow \quad \begin{array}{r} x\ \mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ y \ \text{ et} \\ y\ \mathcal{R}^F_{(n_2,\varphi_2,d_2)}\ z \end{array} \quad \Rightarrow F \in \mathbf{P} \Rightarrow \mathbf{P}' \subseteq \mathbf{P}$$

Avec les résultats précédents, on obtient $\mathbf{Q}' \subseteq \mathbf{P}$ (qed). □

D'après la définition de $\sim$ (section 7.1.3), le problème général du calcul de tous les triplets $(n'_1n'_2, n'_2\varphi'_1 + d'_1\varphi'_2, d'_1d'_2)$ est NP-complet. Ce calcul est nécessaire afin de tester si, dans l'espace des triplets $(n'_1n'_2, n'_2\varphi'_1 + d'_1\varphi'_2, d'_1d'_2)$, il en existe un qui soit équivalent à (n, φ, d) donné. Cette recherche peut être réalisée par des heuristiques adaptées qui exploitent les propriétés de l'espace de recherche. On peut remarquer que $(n_1n_2, \varphi_1n_2, d_1n_2) \sim (n_1, \varphi_1, d_1)$,$(n_2d_1, \varphi_2d_1, d_2d_1) \sim (n_2, \varphi_2, d_2)$ et $d_1n_2 = n_2d_1$, donc $(n_1n_2, n_2\varphi_1 + d_1\varphi_2, d_1d_2)$ est un élément de l'espace de recherche qui peut constituer la solution initiale d'une heuristique quelconque. A partir de cette constatation, une étude des techniques de recherche induites par les propriétés des triplets (n'_1, φ'_1, d'_1) et (n'_2, φ'_2, d'_2) peut être envisagée.

Composition de relations affines – conclusion La composition de relations affines ne produit pas dans tous les cas une relation affine, i.e. l'espace des relations affines $\mathcal{R}^F_{(n,\varphi,d)}$, avec $n, d \in \mathbb{N}^*$ et $\varphi \in \mathbb{Z}$, n'est pas fermé par rapport à l'opération de composition. Conformément au Lemme 6, un sous-espace de cet espace, celui restreint aux relations affines $\mathcal{R}^F_{(1,\varphi,d)}$, avec $\varphi \geq 0$, possède la propriété de fermeture souhaitée. Dans ces conditions, nous choisissons de partager, dans la suite, notre étude suivant deux directions principales. D'une part, nous allons étudier l'influence des opérations d'union, intersection et différence sur la propriété de fermeture de l'espace des relations $\mathcal{R}^F_{(1,\varphi,d)}$, avec $\varphi \geq 0$. La deuxième direction va considérer les relations affines générales $\mathcal{R}^F_{(n,\varphi,d)}$ et va traiter les principaux aspects et problèmes soulevés dans cet espace par les opérations mentionnées.

7.1.5.3 Élément neutre (par rapport à la composition)

Généralement, la relation identité I est définie telle que [San80] :

$$xIy \;\Leftrightarrow\; x = y$$

Dans le cas des relations affines :

Définition 9 *Pour tout flot $F \in \mathcal{F}_{A_1}$, $\mathcal{R}^F_{(n,\varphi,d)}$ est une relation d'identité dans l'espace des relations affines ssi la relation suivante est vérifiée :*

$$x\ \mathcal{R}^F_{(n,\varphi,d)}\ y \quad\Leftrightarrow\quad x \stackrel{F}{=} y$$

□

Le lemme suivant introduit $\mathcal{R}^F_{(1,0,1)}$ comme relation affine d'identité.

Lemme 8 *Pour tout flot $F \in \mathcal{F}_{A_1}$,*

$$\mathcal{R}^F_{(1,0,1)} = \{(x, y) \in A_1 \times A_1 \ /\ x \stackrel{F}{=} y\}$$

Preuve Soit $(x, y) \in \mathcal{R}^F_{(1,0,1)}$. Conformément à la définition d'une relation affine, section 7.1.1, et aux Corollaires 1 et 2, section 7.1.2.1, il existe une fonction $g : \mathbb{N} \to \mathbb{N}$, strictement croissante sur $\mathbb{N}$, telle que

$$F = (\ g \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ g \uparrow \Pi_{\{y\}}(F)\ ) \cdot F$$

et

$$\forall\ t \in \mathbb{N},\ x_{(F)}(t) = y_{(F)}(s)$$

En conclusion, x et y sont identiques dans F ($x \stackrel{F}{=} y$, voir section 6.4.8.2) et $\mathcal{R}^F_{(1,0,1)}$ devient :

$$\mathcal{R}^F_{(1,0,1)} = \{(x, y) \in A_1 \times A_1 \ /\ x \stackrel{F}{=} y\}$$

□

La relation identité $\mathcal{R}^F_{(1,0,1)}$ est élément neutre par rapport à la composition.

Lemme 9 *Pour tout flot $F \in \mathcal{F}_{A_1}$,*

$$\mathcal{R}^F_{(n,\varphi,d)} \ *\ \mathcal{R}^F_{(1,0,1)} = \mathcal{R}^F_{(n,\varphi,d)}$$
$$\mathcal{R}^F_{(1,0,1)} \ *\ \mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n,\varphi,d)}$$

Preuve $\forall\ (x,y) \in \mathcal{R}^F_{(n,\varphi,d)}$, on a $(y,y) \in \mathcal{R}^F_{(1,0,1)}$ (puisque $y \stackrel{F}{=} y$), et donc $(x,y) \in \mathcal{R}^F_{(n,\varphi,d)}\ *\ \mathcal{R}^F_{(1,0,1)}$. Il résulte

$$\mathcal{R}^F_{(n,\varphi,d)} \subseteq \mathcal{R}^F_{(n,\varphi,d)}\ *\ \mathcal{R}^F_{(1,0,1)}$$

$\forall\ (x,y) \in \mathcal{R}^F_{(n,\varphi,d)}\ *\ \mathcal{R}^F_{(1,0,1)}, (\exists) u \in A_1$ tel que $(x,u) \in \mathcal{R}^F_{(n,\varphi,d)}$ et $(u,y) \in \mathcal{R}^F_{(1,0,1)}$.

$(u,y) \in \mathcal{R}^F_{(1,0,1)}$ implique $u \stackrel{F}{=} y$, et donc $(x,y) \in \mathcal{R}^F_{(n,\varphi,d)}$. Il résulte

$$\mathcal{R}^F_{(n,\varphi,d)}\ *\ \mathcal{R}^F_{(1,0,1)} \subseteq \mathcal{R}^F_{(n,\varphi,d)}$$

□

Plus généralement, il est facilement vérifiable que $\mathcal{R}^F_{(1,0,1)}$ représente l'élément identité dans l'espace des relations de (f_1,f_2)-codensification générale. Dans ce cadre, la propriété de composabilité énoncée par le lemme précédent est valable pour une relation $\mathcal{R}^F_{(f_1,f_2)}$ arbitraire.

Lemme 10 *Pour tout flot $F \in \mathcal{F}_{A_1}$ et pour toute relation $\mathcal{R}^F_{(f_1,f_2)}$, on a :*

$$\mathcal{R}^F_{(f_1,f_2)}\ *\ \mathcal{R}^F_{(1,0,1)} = \mathcal{R}^F_{(f_1,f_2)}$$
$$\mathcal{R}^F_{(1,0,1)}\ *\ \mathcal{R}^F_{(f_1,f_2)} = \mathcal{R}^F_{(f_1,f_2)}$$

avec f_1, f_2 fonctions de densification arbitraires.

7.1.5.4 *Union* de relations affines

L'union $\mathcal{R}_1 \cup \mathcal{R}_2$ des relations $\mathcal{R}_1$ et $\mathcal{R}_2$ arbitraires est définie [San80] comme suit :

$$x(\mathcal{R}_1 \cup \mathcal{R}_2)y \quad \Leftrightarrow \quad x\mathcal{R}_1 y \text{ ou } x\mathcal{R}_2 y$$

Nous définissons sur l'espace des relations affines un opérateur d'union, noté $\cup_r$, différent de $\cup$ habituel et induit par l'opérateur d'union $\vee_v$ sur variables.

Définition 10 *Pour tout flot $F \in \mathcal{F}_{A_1}$, l'union des relations $\mathcal{R}^F_{(n_1,\varphi_1,d_1)}$ et $\mathcal{R}^F_{(n_2,\varphi_2,d_2)}$, notée $\cup_r$, est définie comme suit :*

$$\mathcal{R}^F_{(n_1,\varphi_1,d_1)} \cup_r \mathcal{R}^F_{(n_2,\varphi_2,d_2)} = \{(x,y) \in A_1 \times A_1\ /\ \exists x_1, x_2 \in A_1, (x,x_1) \in \mathcal{R}^F_{(n_1,\varphi_1,d_1)},$$
$$(x,x_2) \in \mathcal{R}^F_{(n_2,\varphi_2,d_2)} \text{ et } y \stackrel{F}{=} x_1 \vee_v x_2\}$$

□

Nous proposons une étude de l'influence des valeurs des triplets (n_1,φ_1,d_1) et (n_2,φ_2,d_2) sur l'union des relations affines $\mathcal{R}^F_{(n_1,\varphi_1,d_1)}$ et $\mathcal{R}^F_{(n_2,\varphi_2,d_2)}$. Notre étude se déroulera d'abord dans l'espace des relations affines $\mathcal{R}^F_{(1,\varphi,d)}$, avec $\varphi \geq 0$. Dans ce contexte, nous démontrons une proposition qui fixe certains triplets $(1,\varphi_1,d_1)$ et $(1,\varphi_2,d_2)$, avec $\varphi_1,\varphi_2 \geq 0$, pour lesquels l'union $\cup_r$ des relations affines correspondantes est égale à une relation affine dont les paramètres dépendent de $(1,\varphi_1,d_1)$ et $(1,\varphi_2,d_2)$. Cette proposition permet ainsi de constater que l'union de relations affines $\mathcal{R}^F_{(1,\varphi,d)}$, avec $\varphi \geq 0$, n'est pas toujours une relation affine de même type, donc l'espace des relations affines $\mathcal{R}^F_{(1,\varphi,d)}$ n'est pas fermé par rapport à l'opération d'union. Bien que réduite, l'étude permettra de construire des règles d'identifiabilité de variables construites comme unions de variables x_i en relations affines $\mathcal{R}^F_{(1,\varphi_i,d_i)}$, $\varphi_i \geq 0$, avec la même variable x : $(x,x_i) \in \mathcal{R}^F_{(1,\varphi_i,d_i)}$.

Union de relations affines – analyse de cas On considère que $d_1, \varphi_1, d_2, \varphi_2$ et d, φ satisfont le système de contraintes (C_u) si leurs valeurs vérifient les conditions (1), (2) ou (3) ci-dessous ($(C_u) =$ (1) ou (2) ou (3)) :

$$(1)\quad \begin{cases} d_2 = k_2 d_1 \\ \varphi_2 = \varphi_1 + k_1' d_1 \\ d = d_1 \\ \varphi = \varphi_1 \end{cases} \quad (k_2 \in \mathbb{N}^*, k_1' \in \mathbb{N}) \qquad (2)\quad \begin{cases} d_1 = d_2 = 2k \\ \varphi_2 = \varphi_1 + k \\ d = d_1 \\ \varphi = \varphi_1 \end{cases} \quad (k \in \mathbb{N}^*) \qquad (3)\quad \begin{cases} d_1 = k_1 d_2 \\ \varphi_2 = \varphi_1 + \begin{cases} d_2, & \text{si } k_1 > 1 \\ k_2' d_2, & \text{si } k_1 = 1 \\ (k_2' \in \mathbb{N}^*) \end{cases} \\ d = d_2 \\ \varphi = \varphi_1 \end{cases} \quad (k_1 \in \mathbb{N}^*)$$

ou les cas réciproques $(1'), (2'), (3')$ obtenus en permutant d_1 et d_2, φ_1 et φ_2 ci-dessus.

Proposition 3 *$\forall\ F \in \mathcal{F}_{\{x,x_1,x_2,y\}}$ satisfaisant $x\ \mathcal{R}^F_{(1,\varphi_1,d_1)}\ x_1$, $x\ \mathcal{R}^F_{(1,\varphi_2,d_2)}\ x_2$ et $y \stackrel{F}{=} x_1 \vee_v x_2$, la relation suivante est valide :*

$$\mathcal{R}^F_{(1,\varphi_1,d_1)} \cup_r \mathcal{R}^F_{(1,\varphi_2,d_2)} = \mathcal{R}^F_{(1,\varphi,d)} \quad \Leftrightarrow \quad (C_u)$$

Preuve Les hypothèses de la Proposition 3 impliquent $x\ \mathcal{R}^F_{(1,\varphi_1,d_1)}\ \cup_r\ \mathcal{R}^F_{(1,\varphi_2,d_2)}\ y$, i.e. il existe $g : \mathbb{N} \to \mathbb{N}$, strictement croissante sur $\mathbb{N}$, telle que :

$$\begin{aligned} F = (\ (g \circ f_{x_1}) \uparrow \Pi_{\{x_1\}}(F)\ ) \cdot (\ (g \circ f_{x_2}) \uparrow \Pi_{\{x_2\}}(F)\ ) \cdot \\ \cdot(\ g \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ (g \circ f_y) \uparrow \Pi_{\{y\}}(F)\ ) \cdot F \end{aligned}$$

et

$$\begin{aligned} &\forall\ t, s \in \mathbb{N},\ t = f_{x_1}(s) \ \Rightarrow\ x_{(F)}(t) = x_{1(F)}(s) \\ &\forall\ t, s \in \mathbb{N},\ t = f_{x_2}(s) \ \Rightarrow\ x_{(F)}(t) = x_{2(F)}(s) \end{aligned}$$

où les expressions de f_{x_1}, f_{x_2}, f_y vérifient $f_{x_1} = \lambda\ t.(d_1 t + \varphi_1)$, $f_{x_2} = \lambda\ t.(d_2 t + \varphi_2)$ et $f_y(\mathbb{N}) = f_{x_1}(\mathbb{N}) \cup f_{x_2}(\mathbb{N})$.

S'il existe $\varphi \geq 0, d > 0$ tels que $\mathcal{R}^F_{(1,\varphi_1,d_1)} \cup_r \mathcal{R}^F_{(1,\varphi_2,d_2)} = \mathcal{R}^F_{(1,\varphi,d)}$, alors $(x, y) \in \mathcal{R}^F_{(1,\varphi,d)}$ et $f_y = \lambda\ t.(dt + \varphi)$. Dans cette situation, $\varphi_1, d_1, \varphi_2, d_2$ doivent satisfaire les conditions nécessaires et suffisantes pour que $(d_1\mathbb{N} + \varphi_1) \cup (d_2\mathbb{N} + \varphi_2)$ puisse se réécrire en $d\mathbb{N} + \varphi$:

$$(d_1\mathbb{N} + \varphi_1) \cup (d_2\mathbb{N} + \varphi_2) = d\mathbb{N} + \varphi$$

Les propriétés de l'union et de l'intersection de variables :

$$\begin{aligned} (\ x_1 \vee_v x_2\ ) \wedge_v x_1 \stackrel{F}{=} x_1 \quad &\Leftrightarrow \quad y \wedge_v x_1 \stackrel{F}{=} x_1 \\ (\ x_1 \vee_v x_2\ ) \wedge_v x_2 \stackrel{F}{=} x_2 \quad &\Leftrightarrow \quad y \wedge_v x_2 \stackrel{F}{=} x_2 \end{aligned}$$

impliquent :

$$\begin{aligned} f_y(\mathbb{N}) \cap f_{x_1}(\mathbb{N}) = f_{x_1}(\mathbb{N}) \quad &\Rightarrow \quad (d\mathbb{N} + \varphi) \cap (d_1\mathbb{N} + \varphi_1) = d_1\mathbb{N} + \varphi_1 \\ f_y(\mathbb{N}) \cap f_{x_2}(\mathbb{N}) = f_{x_2}(\mathbb{N}) \quad &\Rightarrow \quad (d\mathbb{N} + \varphi) \cap (d_2\mathbb{N} + \varphi_2) = d_2\mathbb{N} + \varphi_2 \end{aligned}$$

Les valeurs de $d_1, d_2, d, \varphi_1, \varphi_2, \varphi$ doivent satisfaire les conditions nécessaires et suffisantes pour que les égalités suivantes soient valides :

$$\begin{aligned} (d_1\mathbb{N} + \varphi_1) \cup (d_2\mathbb{N} + \varphi_2) = d\mathbb{N} + \varphi \\ (d_1\mathbb{N} + \varphi_1) \cap (d\mathbb{N} + \varphi) = d_1\mathbb{N} + \varphi_1 \\ (d_2\mathbb{N} + \varphi_2) \cap (d\mathbb{N} + \varphi) = d_2\mathbb{N} + \varphi_2 \end{aligned} \tag{7.7}$$

La Proposition 18 de l'annexe D propose l'ensemble (C_u) comme conditions nécessaires et suffisantes pour les valeurs de $d_1, d_2, d, \varphi_1, \varphi_2, \varphi$ afin que les égalités (7.7) soient vérifiées.

Le raisonnement antérieur représente la preuve de l'implication suivante :

$$(\mathcal{R}^F_{(1,\varphi_1,d_1)} \cup_r \mathcal{R}^F_{(1,\varphi_2,d_2)}) = \mathcal{R}^F_{(1,\varphi,d)} \quad \Rightarrow \quad (C_u)$$

Supposons maintenant que (d_1,φ_1), (d_2,φ_2) et (d,φ) vérifient (C_u). Dans ce cas, la relation suivante est vérifiée :

$$(d_1\mathbb{N} + \varphi_1) \cup (d_2\mathbb{N} + \varphi_2) = d\mathbb{N} + \varphi$$

et dans le flot F, la fonction f_y est affine : $f_y = \lambda\ t.(dt+\varphi)$. De plus, il est facilement vérifiable, à partir des propriétés des valeurs de x, x_1 et x_2 et en utilisant la définition de y comme union de x_1 et x_2 dans le flot F, que :

$$\forall\ t,s \in \mathbb{N},\ t = f_y(s) \ \Rightarrow\ x_{(F)}(t) = y_{(F)}(s)$$

En conséquence $(x,y) \in \mathcal{R}^F_{(1,\varphi,d)}$.

Les variables x et y satisfont les conditions suivantes :

$$(x,y) \in \mathcal{R}^F_{(1,\varphi_1,d_1)} \cup_r \mathcal{R}^F_{(1,\varphi_2,d_2)}$$
$$(x,y) \in \mathcal{R}^F_{(1,\varphi,d)}$$

Nous considérons le cas général $\mathcal{R}^F_{(1,\varphi,d)} = \{(x,y)\}$ (dans le cas où la relation affine $\mathcal{R}^F_{(1,\varphi,d)}$ contient au moins un autre élément différent de (x,y), on obtient des contraintes supplémentaires sur les valeurs des paramètres $(1,\varphi_1,d_1)$ et $(1,\varphi_2,d_2)$; les nouvelles valeurs de $(1,\varphi_1,d_1)$ et $(1,\varphi_2,d_2)$ génèrent des cas particuliers du cas général traité).

On a dans cette situation

$$\mathcal{R}^F_{(1,\varphi,d)} \subseteq\ \mathcal{R}^F_{(1,\varphi_1,d_1)} \cup_r \mathcal{R}^F_{(1,\varphi_2,d_2)}$$

et comme pour toute paire $(u,v) \in \mathcal{R}^F_{(1,\varphi_1,d_1)} \cup_r \mathcal{R}^F_{(1,\varphi_2,d_2)}$ il résulte $(u,v) \in \mathcal{R}^F_{(1,\varphi,d)}$, il est immédiat que :

$$\mathcal{R}^F_{(1,\varphi_1,d_1)} \cup_r \mathcal{R}^F_{(1,\varphi_2,d_2)} = \mathcal{R}^F_{(1,\varphi,d)}$$

□

Sur la Figure 7.14, nous avons représenté un flot $F \in \mathcal{F}_{\{x,u,v,y\}}$ tel que $y \stackrel{F}{=} u \vee_v v$ et $x\ \mathcal{R}^F_{(1,1,3)}\ u$, $x\ \mathcal{R}^F_{(1,4,6)}\ v$, $x\ \mathcal{R}^F_{(1,1,3)}\ y$ (nous avons renoté les variables x_1 et x_2 en u et v et les fonctions f_u, f_v et f_y correspondent aux fonctions f_{x_1}, f_{x_2} et f_y de la Proposition 3 précédente).

Union de relations affines – cas général Dans le cas général où on manipule des valeurs arbitraires pour les paramètres (n_1,φ_1,d_1) et (n_2,φ_2,d_2), il est intéressant de caractériser l'ensemble des relations affines $\mathcal{R}^F_{(n,\varphi,d)}$ telles que $F \in \mathcal{F}_{\{x,x_1,x_2,y\}}$ avec les propriétés $x\ \mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ x_1$, $x\ \mathcal{R}^F_{(n_2,\varphi_2,d_2)}\ x_2$ et $y \stackrel{F}{=} x_1 \vee_v x_2$, $x\ \mathcal{R}^F_{(n,\varphi,d)}\ y$.

A premier abord, ce problème s'avère complexe. Une stratégie à adopter serait d'utiliser les résultats obtenus pour la composition de relations affines afin de déterminer l'ensemble des triplets (n',φ',d') tels qu'il existe $F \in \mathbf{P}$, $x_1\ \mathcal{R}^F_{(n',\varphi',d')}\ x_2$, avec $\mathbf{P}$ défini comme suit :

$$\mathbf{P} \quad = \quad \{F \in \mathcal{F}_{\{x,x_1,x_2,y\}} \mid\ x\ \mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ x_1\ ,\ x\ \mathcal{R}^F_{(n_2,\varphi_2,d_2)}\ x_2\ \text{ et } y \stackrel{F}{=} x_1 \vee_v x_2\}$$

Ensuite, les résultats sur l'union de relations affines présentés dans le paragraphe précédent peuvent être utilisés[3] afin de déduire des conditions sur les triplets (n',φ',d') – et implicitement

3. Cf. à la Proposition 16, section 7.4.2.2, si $x_1\ \mathcal{R}^F_{(n',\varphi',d')}\ x_2$, il existe x tel que $x\ \mathcal{R}^F_{(1,\phi_1,D_1)}\ x_1$ et $x\ \mathcal{R}^F_{(1,\phi_2,D_2)}\ x_2$ et dans ce cas on se situe dans les hypothèses de la Proposition 3 précédente.

t		0	1	2	3	4	5	6	7	8	9	10	11	12
$g(t)$		0	1	2	3	4	5	6	7	8	9	10	11	12
$(g\,o\,f_u)(t)$			1			4			7			10		
$(g\,o\,f_v)(t)$						4						10		
$(g\,o\,f_y)(t)$			1			4			7			10		
F	x	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	....
	u	$\perp$	x_2	$\perp$	$\perp$	x_5	$\perp$	$\perp$	x_8	$\perp$	$\perp$	x_{11}	$\perp$	....
	v	$\perp$	$\perp$	$\perp$	$\perp$	x_5	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	x_{11}	$\perp$	....
	y	$\perp$	x_2	$\perp$	$\perp$	x_5	$\perp$	$\perp$	x_8	$\perp$	$\perp$	x_{11}	$\perp$	....

FIG. 7.14 – *Illustration de la Proposition 3.*

sur (n_1, φ_1, d_1) et (n_2, φ_2, d_2) – pour qu'il existe $F \in \mathbf{P}$ dans lequel y est en relation affine avec x_1 et x_2.

De manière analogue peuvent être abordées les opérations d'intersection et différence de relations affines qui seront définies dans la suite. Les résultats obtenus permettrons d'accroître le système de règles de synchronisabilité d'horloges que nous proposons en fin de ce chapitre. En ce qui nous concerne, le système de règles telles qu'il est actuellement s'est montré suffisant pour les nécessités de validation d'applications de notre contexte de travail.

7.1.5.5 *Intersection* de relations affines

L'intersection $\mathcal{R}_1 \cap \mathcal{R}_2$ des relations $\mathcal{R}_1$ et $\mathcal{R}_2$ arbitraires est définie [San80] comme suit :

$$x(\mathcal{R}_1 \cap \mathcal{R}_2)y \quad \Leftrightarrow \quad x\mathcal{R}_1 y \text{ et } x\mathcal{R}_2 y$$

Comme dans le cas de l'union de relations affines, nous introduisons un opérateur d'intersection, noté $\cap_r$, différent de l'opérateur $\cap$ habituel et induit par l'intersection $\wedge_v$ sur variables.

Définition 11 *Pour tout flot $F \in \mathcal{F}_{A_1}$, l'intersection des relations $\mathcal{R}^F_{(n_1,\varphi_1,d_1)}$ et $\mathcal{R}^F_{(n_2,\varphi_2,d_2)}$, notée $\cap_r$, est définie comme suit :*

$$\mathcal{R}^F_{(n_1,\varphi_1,d_1)} \cap_r \mathcal{R}^F_{(n_2,\varphi_2,d_2)} = \{(x,y) \in A_1 \times A_1 \;/\; \exists \;\; x_1, x_2 \in A_1, (x,x_1) \in \mathcal{R}^F_{(n_1,\varphi_1,d_1)}, \\ (x,x_2) \in \mathcal{R}^F_{(n_2,\varphi_2,d_2)} \text{ et } y \overset{F}{=} x_1 \wedge_v x_2\}$$

□

Nous étudions l'influence des paramètres (n_1, φ_1, d_1) et (n_2, φ_2, d_2) sur l'intersection des relations $\mathcal{R}^F_{(n_1,\varphi_1,d_1)}$ et $\mathcal{R}^F_{(n_2,\varphi_2,d_2)}$. Comme dans le cas de l'union $\cup_r$, nous divisons notre étude en considérant le cas des relations affines $\mathcal{R}^F_{(1,\varphi,d)}$, avec $\varphi \geq 0$; le cas général $\mathcal{R}^F_{(n,\varphi,d)}$ peut être ensuite abordé comme décrit dans le paragraphe précédent pour l'opération d'union. Concernant les relations affines $\mathcal{R}^F_{(1,\varphi,d)}$, avec $\varphi \geq 0$, nous allons constater que, si $(x,y) \in \mathcal{R}^F_{(1,\varphi_1,d_1)} \cap_r \mathcal{R}^F_{(1,\varphi_2,d_2)}$ et $\mid \Pi_{\{y\}}(F) \mid \neq 0$ (l'horloge $h(y)$ de y est non nulle), alors l'intersection de $\mathcal{R}^F_{(1,\varphi_1,d_1)}$ et $\mathcal{R}^F_{(1,\varphi_2,d_2)}$ produit toujours comme résultat une relation affine. L'hypothèse que l'horloge $h(y)$ soit non nulle impose des contraintes sur les valeurs des paramètres $(1,\varphi_1,d_1)$ et $(1,\varphi_2,d_2)$ (les contraintes (C_{div}) ci-dessous). Cette étude nous permettra de construire des règles d'identifiabilité de variables construites comme intersections de variables x_i en relations affines $\mathcal{R}^F_{(1,\varphi_i,d_i)}$, $\varphi_i \geq 0$, avec la même variable x : $(x,x_i) \in \mathcal{R}^F_{(1,\varphi_i,d_i)}$.

On considère que $d_1, \varphi_1, d_2, \varphi_2$ et d, φ satisfont le système de contraintes (C_i) si leurs valeurs vérifient

$$(C_i) = \begin{cases} (C_{div}) \quad pgcd(d_1,d_2) \mid \varphi_2 - \varphi_1 \\ d = \frac{d_1 d_2}{g} = ppcm(d_1,d_2) \\ \varphi = d_1 s_{10} + \varphi_1 \end{cases}$$

avec $pgcd(d_1,d_2){=}g$ le plus grand diviseur commun de d_1 et d_2 et $s_{10} = s'_{10} - \frac{d_2}{g}\,[(s'_{10}g)/d_2]$ avec (s'_{10}, s'_{20}) une solution particulière de $d_1 s'_{10} - d_2 s'_{20} = \varphi_2 - \varphi_1$. (C_i) regroupe les conditions nécessaires et suffisantes pour que l'intersection de $\mathcal{R}^F_{(1,\varphi_1,d_1)}$ et $\mathcal{R}^F_{(1,\varphi_2,d_2)}$ soit non vide (conditions (C_{div})) et puisse s'exprimer sous la forme de la relation affine $\mathcal{R}^F_{(1,\varphi,d)}$.

Proposition 4 *$\forall\ F \in \mathcal{F}_{\{x,x_1,x_2,y\}}$ satisfaisant $x\ \mathcal{R}^F_{(1,\varphi_1,d_1)}\ x_1$, $x\ \mathcal{R}^F_{(1,\varphi_2,d_2)}\ x_2$ et $y \stackrel{F}{=} x_1 \wedge_v x_2$, la relation suivante est valide :*

$$\mathcal{R}^F_{(1,\varphi_1,d_1)} \cap_r \mathcal{R}^F_{(1,\varphi_2,d_2)} = \mathcal{R}^F_{(1,\varphi,d)} \quad \Leftrightarrow \quad (C_i)$$

Preuve La preuve suit un schéma similaire à celui de la preuve de la Proposition 3 en utilisant Proposition 17 de l'annexe D ; nous ne la présentons pas. □

7.1.5.6 *Différence* de relations affines

La différence $\mathcal{R}_1 \setminus \mathcal{R}_2$ des relations $\mathcal{R}_1$ et $\mathcal{R}_2$ arbitraires est définie [San80] comme suit :

$$x(\mathcal{R}_1 \setminus \mathcal{R}_2)y \quad \Leftrightarrow \quad x\mathcal{R}_1 y \text{ et } x \not\mathcal{R}_2 y$$

Autrement dit, (x,y) appartient à la différence des relations $\mathcal{R}_1$ et $\mathcal{R}_2$ ssi (x,y) appartient à $\mathcal{R}_1$ et n'appartient pas à $\mathcal{R}_2$.

Comme précédemment dans le cas de l'union $\cup_r$ et de l'intersection $\cap_r$, l'opérateur de différence $\setminus_r$ sur l'espace des relations affines est induit par l'opérateur homonyme $\setminus_v$ sur variables et diffère de la différence habituelle.

Définition 12 *Pour tout flot $F \in \mathcal{F}_{A_1}$, la différence des relations $\mathcal{R}^F_{(n_1,\varphi_1,d_1)}$ et $\mathcal{R}^F_{(n_2,\varphi_2,d_2)}$, notée $\setminus_r$, est définie comme suit :*

$$\mathcal{R}^F_{(n_1,\varphi_1,d_1)} \setminus_r \mathcal{R}^F_{(n_2,\varphi_2,d_2)} = \{(x,y) \in A_1 \times A_1 \ / \ \exists\ x_1, x_2 \in A_1, (x,x_1) \in \mathcal{R}^F_{(n_1,\varphi_1,d_1)}, \\ (x,x_2) \in \mathcal{R}^F_{(n_2,\varphi_2,d_2)} \text{ et } y \stackrel{F}{=} x_1 \setminus_v x_2\}$$

□

Comme dans les cas précédents, nous étudions dans quelles conditions $\mathcal{R}^F_{(1,\varphi_1,d_1)} \setminus_r \mathcal{R}^F_{(1,\varphi_2,d_2)}$ produit une relation affine. Nous allons voir que ceci n'est pas vrai dans le cas général.

On considère que $d_1, \varphi_1, d_2, \varphi_2$ et d, φ satisfont les contraintes (C_d) si leurs valeurs vérifient (1) ou (2) ci-dessus:

$$(C_d) = (1) \text{ ou } (2), \text{ avec } \quad (1) \begin{cases} d_2 = 2d_1 \\ \varphi_2 = \varphi_1 \\ d = 2d_1 \\ \varphi = \varphi_1 + d_1 \end{cases} \quad \text{et} \quad (2) \begin{cases} d_2 = 2d_1 \\ \varphi_2 = \varphi_1 + d_1 \\ d = 2d_1 \\ \varphi = \varphi_1 \end{cases}$$

Proposition 5 *$\forall\ F \in \mathcal{F}_{\{x,x_1,x_2,y\}}$ satisfaisant $x\ \mathcal{R}^F_{(1,\varphi_1,d_1)}\ x_1$, $x\ \mathcal{R}^F_{(1,\varphi_2,d_2)}\ x_2$ et $y \overset{F}{=} x_1 \setminus_v x_2$, la relation suivante est valide :*

$$\mathcal{R}^F_{(1,\varphi_1,d_1)} \setminus_r \mathcal{R}^F_{(1,\varphi_2,d_2)} = \mathcal{R}^F_{(1,\varphi,d)} \quad \Leftrightarrow \quad (C_d)$$

Preuve La preuve suit un schéma similaire à celui de la preuve de la Proposition 3 en utilisant Proposition 19 de l'annexe c.appendix ; nous ne la présentons pas. □

7.1.5.7 Forme normale relative d'une expression sur relations affines

Les propriétés des relations affines $\mathcal{R}^F_{(n,\varphi,d)}$ et des termes qui peuvent être construits sur ces relations avec les opérateurs de composition, union, intersection et/ou différence comme nous les avons définis dépendent des valeurs des triplets (n, φ, d). Dans le cas général, l'espace des relations affines $\mathcal{R}^F_{(n,\varphi,d)}$ n'est pas fermé par rapport aux opérations précédentes. En utilisant le Lemme 6, section 7.1.5.2, on obtient

$$\mathcal{R}^F_{(1,\varphi_1,d_1)} \ * \ \mathcal{R}^F_{(1,\varphi_2,d_2)} = \mathcal{R}^F_{(1,\varphi_1+d_1\varphi_2,d_1d_2)}$$

pour tout flot $F \in \mathcal{F}_{\{x,y,z\}}$ tel que $x\ \mathcal{R}^F_{(1,\varphi_1,d_1)}\ y$, avec $\varphi_1 \geq 0$, et $y\ \mathcal{R}^F_{(1,\varphi_2,d_2)}\ z$, avec $\varphi_2 \geq 0$. On peut ainsi remarquer que l'espace des relations affines $\mathcal{R}^F_{(1,\varphi,d)}$, avec $\varphi \geq 0$, est fermé par rapport à la composition. La propriété de fermeture se perd dès que l'on considère les opérations d'union et de différence, comme il a été constaté dans les sections 7.1.5.4 et 7.1.5.6 précédentes. Or, l'espace des expressions construites sur des relations affines de type $\mathcal{R}^F_{(1,\varphi,d)}$, avec $\varphi \geq 0$, en utilisant les opérateurs de composition, union, intersection et différence est évidemment fermé par rapport à ces opérations (les relations affines $\mathcal{R}^F_{(1,\varphi,d)}$ correspondent à des sous-échantillonnages affines dont $\varphi \geq 0$ représente la *phase* et $d \geq 1$ la *période*). Dans cet espace, l'étude de l'identifiabilité de deux variables se réduit à celle de l'existence d'une *forme unique* de représentation d'une expression sur relations affines $\mathcal{R}^F_{(1,\varphi,d)}$, avec $\varphi \geq 0$.

Soit le flot $F \in \mathcal{F}_{A_1}$ dans lequel les relations $\mathcal{R}^F_{t_1}$ et $\mathcal{R}^F_{t_2}$ s'écrivent :

$$\begin{aligned} \mathcal{R}^F_{t_1} &= \mathcal{EXP}_1(\mathcal{R}^F_{(1,\varphi_1,d_1)}, \ldots, \mathcal{R}^F_{(1,\varphi_p,d_p)}) \\ \mathcal{R}^F_{t_2} &= \mathcal{EXP}_2(\mathcal{R}^F_{(1,\xi_1,n_1)}, \ldots, \mathcal{R}^F_{(1,\xi_q,n_q)}) \end{aligned}$$

avec $\mathcal{EXP}_1$, $\mathcal{EXP}_2$ des expressions sur relations affines construites avec les opérateurs $*$, $\cup_r$, $\cap_r$ et $\setminus_r$. La distributivité des opérateurs $\cup_r$, $\cap_r$ et $\setminus_r$ par rapport à $*$ – facile à vérifier – permet d'éliminer l'opérateur de composition $*$ dans les expressions $\mathcal{EXP}_1$ et $\mathcal{EXP}_2$ qui peuvent ainsi s'écrire en utilisant uniquement les opérateurs d'union $\cup_r$, d'intersection $\cap_r$ et de différence $\setminus_r$.

Nous rénotons $\mathcal{EXP}_1$ et $\mathcal{EXP}_2$ les nouvelles expressions contenant uniquement les opérateurs $\cup_r$, $\cap_r$ et $\setminus_r$. Nous étudions les aspects liés à l'existence d'une forme unique d'expression pour $\mathcal{R}^F_{t_1}$ et $\mathcal{R}^F_{t_2}$ qui permettrait de les comparer afin de pouvoir décider de l'identifiabilité de x et y appartenant à A_1 qui satisfont $u\ \mathcal{R}^F_{t_1}\ x$ et $u\ \mathcal{R}^F_{t_2}\ y$. Effectivement, pour un processus $\mathbf{P} \in \mathcal{P}_{\{u,x_1,\dots,x_p,y_1,\dots,y_q,x,y\}}$ tel que $u\ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_i,d_i)}\ x_i$, $\forall i = \overline{1,p}$, et $u\ \mathcal{R}^{\mathbf{P}}_{(1,\xi_j,n_j)}\ y_j$, $\forall j = \overline{1,q}$, si $x \overset{\mathbf{P}}{=} Exp_1(x_1,\dots,x_p)$ et $y \overset{\mathbf{P}}{=} Exp_2(y_1,\dots,y_q)$ (avec Exp_1 et Exp_2 les expressions sur variables correspondant aux $\mathcal{EXP}_1$ et $\mathcal{EXP}_2$; tous les opérateurs $\cup_r$, $\cap_r$ et $\setminus_r$ sur relations affines ont été remplacés par les opérateurs $\vee_v$, $\wedge_v$ et respectivement $\setminus_v$), alors il résulte $u\ \mathcal{R}^{\mathbf{P}}_{t_1}\ x$ et $u\ \mathcal{R}^{\mathbf{P}}_{t_2}\ y$. Dans ces conditions, x et y sont identifiables dans $\mathbf{P}$ si et seulement si il existe $F_s \in \mathbf{P}$ tel que $\mathcal{R}^{F_s}_{t_1} = \mathcal{R}^{F_s}_{t_2}$.

Pour un flot $F \in \mathcal{F}_{A_1}$, on veut comparer $\mathcal{R}^F_{t_1}$ et $\mathcal{R}^F_{t_2}$ en sachant que les expressions $\mathcal{EXP}_1$ et $\mathcal{EXP}_2$ ne se réduisent pas toujours à une relation affine $\mathcal{R}^F_{(1,\varphi,d)}$. Il s'impose de comparer $\mathcal{R}^F_{t_1}$ et $\mathcal{R}^F_{t_2}$ sur les termes de leurs expressions de définition. En l'occurrence, si les expressions $\mathcal{EXP}_1$ et $\mathcal{EXP}_2$ sont identiques ($p = q$ et $\mathcal{R}^F_{(1,\varphi_i,d_i)} = \mathcal{R}^F_{(1,\xi_i,n_i)}$, $\forall\ i = \overline{1,p}$), il résulte $\mathcal{R}^F_{t_1} = \mathcal{R}^F_{t_2}$. De manière générale, il est fort possible que $\mathcal{R}^F_{t_1}$ et $\mathcal{R}^F_{t_2}$ soient égales, sans pour autant qu'elles possèdent des expressions de définition identiques. Une réécriture adéquate des expressions de $\mathcal{R}^F_{t_1}$ et $\mathcal{R}^F_{t_2}$ doit être réalisée afin de décider de leur égalité. Comme toute expression sur relations affines peut être réduite à une union finie de relations affines – cela peut se démontrer facilement à partir des propriétés des opérations d'union $\vee_v$, intersection $\wedge_v$ et différence $\setminus_v$ sur variables qui induisent des propriétés similaires pour les opérations homonymes sur relations affines – nous recherchons des décompositions de $\mathcal{R}^F_{t_1}$ et $\mathcal{R}^F_{t_2}$ parmi les expressions de ce type.

Le lemme suivant propose un test simple pour comparer deux relations $\mathcal{R}^F_{t_1}$ et $\mathcal{R}^F_{t_2}$ qui peuvent être décomposées sous la forme d'unions finies de relations affines de même période. Ensuite, nous présentons une méthode générique de décomposition de $\mathcal{R}^F_{t_1}$ et $\mathcal{R}^F_{t_2}$ en vue de leur comparaison.

Lemme 11 *Soit $F \in \mathcal{F}_{A_1}$ tel que les relations non vides $\mathcal{R}^F_{t_1}$ et $\mathcal{R}^F_{t_2}$ se décomposent en*

$$\mathcal{R}^F_{t_1} = (\cup_r)_{\varphi \in I_1} \mathcal{R}^F_{(1,\varphi,n)},\quad \mathcal{R}^F_{t_2} = (\cup_r)_{\varphi \in I_2} \mathcal{R}^F_{(1,\varphi,n)}$$

avec $I_1, I_2 \subseteq \mathbb{N}$ finis et $n \in \mathbb{N}^*$ arbitraire.

Alors

$$\mathcal{R}^F_{t_1} = \mathcal{R}^F_{t_2} \quad \Leftrightarrow \quad I_1 = I_2$$

Preuve

- On suppose $\mathcal{R}^F_{t_1} = \mathcal{R}^F_{t_2}$ et on démontre $I_1 = I_2$.

  Toute paire de variables (u, x) appartenant à $\mathcal{R}^F_{t_1}$ appartient également à $\mathcal{R}^F_{t_2}$. Par définition de $\mathcal{R}^F_{t_1}$ et $\mathcal{R}^F_{t_2}$ on obtient : $\exists\ x_1,\dots,x_p,y_1,\dots,y_q \in A_1$ tels que

$$u\ \mathcal{R}^F_{(1,\varphi_i,n)}\ x_i\ ,\ \forall\ i = \overline{1,p}$$
$$u\ \mathcal{R}^F_{(1,\xi_j,n)}\ y_j\ ,\forall\ j = \overline{1,q}$$
$$x \overset{F}{=} (\vee_v)_{i=1}^p x_i \overset{F}{=} (\vee_v)_{j=1}^q y_j$$

  avec $I_1 = \{\varphi_1,\dots,\varphi_p\}$ et $I_2 = \{\xi_1,\dots,\xi_q\}$ (on a $p = |\ I_1\ |$, $q = |\ I_2\ |$).

  Conformément à la définition d'une relation affine et aux propriétés de l'union de variables ($\vee_v$), il existe $g : \mathbb{N} \to \mathbb{N}$ strictement croissante sur $\mathbb{N}$ telle que F s'écrit

$$F = (\ (g \circ f_{x_i}) \uparrow \Pi_{\{x_i\}}(F)\ ) \cdot (\ (g \circ f_{y_j}) \uparrow \Pi_{\{y_j\}}(F)\ ) \cdot (\ g \uparrow \Pi_{\{u\}}(F)\ ) \cdot (\ (g \circ f_x) \uparrow \Pi_{\{x\}}(F)\ ) \cdot F$$

et

$$\forall\ t, s \in \mathbb{N},\ t = f_{x_i}(s) \ \Rightarrow\ u_{(F)}(t) = x_{i(F)}(s)$$
$$\forall\ t, s \in \mathbb{N},\ t = f_{y_j}(s) \ \Rightarrow\ u_{(F)}(t) = y_{j_{(F)}}(s)$$

où les expressions de f_{x_i}, f_{y_j}, f_x vérifient $f_{x_i} = \lambda\ t.(nt + \varphi_i)$, $f_{y_j} = \lambda\ t.(nt + \xi_j)$ et $f_x(\mathbb{N}) = \cup_{i=0}^{p} f_{x_i}(\mathbb{N}) = \cup_{j=0}^{q} f_{y_j}(\mathbb{N})$.

A partir de la dernière égalité on obtient :

$$\begin{aligned} \cup_{i=0}^{p} f_{x_i}(\mathbb{N}) = \cup_{j=0}^{q} f_{y_j}(\mathbb{N}) \ &\Rightarrow\ \cup_{\varphi_1 \in I_1}(n\mathbb{N} + \varphi_1) = \cup_{\varphi_2 \in I_2}(n\mathbb{N} + \varphi_2) \\ &\Rightarrow\ I_1 = I_2 \end{aligned}$$

(conformément au Lemme 19, annexe D).

- On suppose $I_1 = I_2$ et on démontre $\mathcal{R}_{t_1}^F = \mathcal{R}_{t_2}^F$.

  Soit $(u, x) \in \mathcal{R}_{t_1}^F = (\cup_r)_{\varphi \in I_1} \mathcal{R}_{(1,\varphi,n)}^F$. Comme $I_1 = \{\varphi_1, \varphi_2, \ldots, \varphi_p\}$, il existe $x_1, \ldots, x_p \in A_1$ tels que

$$\begin{aligned} &u\ \mathcal{R}_{(1,\varphi_i,n)}^F\ x_i\ ,\ \forall\ i = \overline{1,p} \\ &x \overset{F}{=} (\vee_v)_{i=1}^{p} x_i \end{aligned}$$

  Or $I_1 = I_2$ et $\mathcal{R}_{t_2}^F = (\cup_r)_{\varphi \in I_2} \mathcal{R}_{(1,\varphi,n)}^F$ implique $(u, x) \in \mathcal{R}_{t_2}^F$, donc $\mathcal{R}_{t_1}^F \subseteq \mathcal{R}_{t_2}^F$.

  De manière analogue on démontre $\mathcal{R}_{t_2}^F \subseteq \mathcal{R}_{t_1}^F$ et il résulte $\mathcal{R}_{t_2}^F = \mathcal{R}_{t_1}^F$.

□

Conformément au lemme précédent, deux relations $\mathcal{R}_{t_1}^F$ et $\mathcal{R}_{t_2}^F$ écrites sous la forme de deux unions finies de relations affines de même période n et de phases positives peuvent être facilement comparées sur les ensembles de valeurs de leurs phases respectives. Il est donc intéressant d'étudier dans quelles conditions une relation $\mathcal{R}_t^F$ peut être décomposée de cette manière. Dans la proposition suivante, nous présentons une méthode de décomposition en union finie de relations affines de phases positives et de même période pour toute expression $\mathcal{EXP}_1(\mathcal{R}_{(1,\varphi_1,d_1)}^F, \ldots, \mathcal{R}_{(1,\varphi_p,d_p)}^F)$, pourvu que les termes $\mathcal{R}_{(1,\varphi_i,d_i)}^F$ constituant l'expression se décomposent eux-mêmes en unions finies de relations affines de phases positives et de même période.

Proposition 6 *Soit $F \in \mathcal{F}_{A_1}$ et la relation non vide $\mathcal{R}_t^F$ définie comme suit :*

$$\mathcal{R}_t^F \quad = \quad \mathcal{EXP}(\mathcal{R}_{(1,\varphi_1,d_1)}^F, \ldots, \mathcal{R}_{(1,\varphi_p,d_p)}^F)$$

avec $\mathcal{EXP}$ une expression arbitraire contenant les opérateurs $\cup_r$, $\cap_r$ et $\backslash_r$.

Si

$$\mathcal{R}_{(1,\varphi_i,d_i)}^F = (\cup_r)_{\varphi \in I_i} \mathcal{R}_{(1,\varphi,n)}^F,\ \forall\ i = \overline{1,p}$$

avec $n \in \mathbb{N}^*$ un multiple commun de d_i, $\forall i = \overline{1,p}$, et $I_i = \{\varphi_i, \varphi_i + d_i, \ldots, \varphi_i + (\frac{n}{d_i} - 1)d_i\}$, $\forall\ i = \overline{1,p}$, alors

$$\mathcal{R}_t^F = (\cup_r)_{\varphi \in Exp(I_1, \ldots, I_p)} \mathcal{R}_{(1,\varphi,n)}^F$$

avec Exp l'expression sur ensembles correspondant à $\mathcal{EXP}$.

Preuve

- Soit $(u,x) \in \mathcal{R}_t^F$. On démontre que $(u,x) \in (\cup_r)_{\varphi \in Exp(I_1,\ldots,I_p)} \mathcal{R}^F_{(1,\varphi,n)}$.

  Comme $\mathcal{R}_t^F = \mathcal{EXP}(\mathcal{R}^F_{(1,\varphi_1,d_1)}, \ldots, \mathcal{R}^F_{(1,\varphi_p,d_p)})$, il existe $x_1, \ldots, x_p \in A_1$ tels que $u\ \mathcal{R}^F_{(1,\varphi_i,d_i)}\ x_i$, $\forall\ i = \overline{1,p}$, et $x \stackrel{F}{=} E(x_1, \ldots, x_p)$, avec E l'expression sur variables correspondant à $\mathcal{EXP}$ sur relations affines.

  Comme $\mathcal{R}^F_{(1,\varphi_i,d_i)} = (\cup_r)_{\varphi \in I_i} \mathcal{R}^F_{(1,\varphi,n)}$, $\forall\ i = \overline{1,p}$, il existe x_φ, $\varphi \in I_i$, tels que

  $$\begin{array}{l} u\ \mathcal{R}^F_{(1,\varphi,n)}\ x_\varphi\ , \forall \varphi \in I_i \\ x_i \stackrel{F}{=} (\vee_v)_{\varphi \in I_i} x_\varphi \end{array}$$

  La variable x peut se réécrire conformément aux relations suivantes :

  $$\begin{array}{rcl} x & \stackrel{F}{=} & E((x_1, x_2, \ldots, x_p)) \\ & \stackrel{F}{=} & E(((\vee_v)_{\varphi \in I_1} x_\varphi, \ldots, (\vee_v)_{\varphi \in I_p} x_\varphi)) \\ & \stackrel{F}{=} & (\vee_v)_{\varphi \in Exp(I_1,\ldots,I_p)} x_\varphi \end{array}$$

  Il résulte $(u,x) \in (\cup_r)_{\varphi \in Exp(I_1,\ldots,I_p)} \mathcal{R}^F_{(1,\varphi,n)}$, et donc

  $$\mathcal{R}_t^F \subseteq (\cup_r)_{\varphi \in Exp(I_1,\ldots,I_p)} \mathcal{R}^F_{(1,\varphi,n)}$$

- Soit $(u,x) \in (\cup_r)_{\varphi \in Exp(I_1,\ldots,I_p)} \mathcal{R}^F_{(1,\varphi,n)}$. On démontre que $(u,x) \in \mathcal{R}_t^F$.

  Comme $(u,x) \in (\cup_r)_{\varphi \in Exp(I_1,\ldots,I_p)} \mathcal{R}^F_{(1,\varphi,n)}$, il résulte qu'il existe $y_\varphi \in A_1$ pour tout $\varphi \in Exp(I_1,\ldots,I_p)$, tels que $u\ \mathcal{R}^F_{(1,\varphi,n)}\ y_\varphi$ et $x \stackrel{F}{=} (\vee_v)_{\varphi \in Exp(I_1,\ldots,I_p)} y_\varphi$.

  La variable x peut ensuite se réécrire conformément aux relations suivantes :

  $$\begin{array}{rcl} x & \stackrel{F}{=} & (\vee_v)_{\varphi \in Exp(I_1,\ldots,I_p)} y_\varphi \\ & \stackrel{F}{=} & E(((\vee_v)_{\varphi \in I_1} y'_\varphi, \ldots, (\vee_v)_{\varphi \in I_p} y'_\varphi)) \end{array}$$

  avec y'_φ satisfaisant $u\ \mathcal{R}^F_{(1,\varphi,n)}\ y'_\varphi$, $\forall \varphi \in I_1 \cup \ldots \cup I_p$.

  L'existence des y_φ tels que $u\ \mathcal{R}^F_{(1,\varphi,n)}\ y_\varphi$ pour tout $\varphi \in Exp(I_1,\ldots,I_p)$ et $\mathcal{R}^F_{(1,\varphi_i,d_i)} = (\cup_r)_{\varphi \in I_i} \mathcal{R}^F_{(1,\varphi,n)}$ non vide pour tout $i = \overline{1,p}$ (cette dernière condition est induite par $\mathcal{R}_t^F = \mathcal{EXP}(\mathcal{R}^F_{(1,\varphi_1,d_1)}, \ldots, \mathcal{R}^F_{(1,\varphi_p,d_p)})$ non vide) implique l'existence des y'_φ définis comme précédemment.

  A partir de ce résultat et en considérant les expressions $\mathcal{R}^F_{(1,\varphi_i,d_i)} = (\cup_r)_{\varphi \in I_i} \mathcal{R}^F_{(1,\varphi,n)}$, $\forall\ i = \overline{1,p}$, il résulte qu'il existe $y_i \in A_1$ tels que $u\ \mathcal{R}^F_{(1,\varphi_i,d_i)}\ y_i$ et $y_i \stackrel{F}{=} (\vee_v)_{\varphi \in I_i} y'_\varphi$, $\forall i = \overline{1,p}$.

  On obtient $x \stackrel{F}{=} E((y_1, \ldots, y_p))$ et $(u,x) \in \mathcal{R}_t^F$. En conclusion

  $$(\cup_r)_{\varphi \in Exp(I_1,\ldots,I_p)} \mathcal{R}^F_{(1,\varphi,n)} \subseteq \mathcal{R}_t^F$$

$\square$

Pour un flot $F \in \mathcal{F}_{A_1}$ arbitraire, une expression $\mathcal{R}_t^F$ sur un ensemble fini de relations affines $\mathcal{R}^F_{(1,\varphi_i,d_i)}$ se décompose en union finie de relations affines de phases positives et périodes identiques égales à $n \in \mathbb{N}^*$, si chaque relation affine $\mathcal{R}^F_{(1,\varphi_i,d_i)}$ terme de la relation $\mathcal{R}_t^F$ se décompose également en union finie de relations affines de phases positives et période n. La décomposition précédente n'est pas unique : effectivement, n peut varier en restant un multiple commun des périodes d_i qui caractérisent les relations affines $\mathcal{R}^F_{(1,\varphi_i,d_i)}$ qui interviennent dans l'expression $\mathcal{R}^F_t$. Par exemple, afin de comparer $\mathcal{R}^F_{t_1}$ et $\mathcal{R}^F_{t_2}$, il est nécessaire de les décomposer pour se situer dans les conditions du Lemme 11 précédent. Si $\mathcal{R}^F_{t_1}$ et $\mathcal{R}^F_{t_2}$ sont définies par les expressions $\mathcal{EXP}_1$ et $\mathcal{EXP}_2$ comme suit :

$$\begin{array}{rcl} \mathcal{R}^F_{t_1} & = & \mathcal{EXP}_1(\mathcal{R}^F_{(1,\varphi_1,d_1)}, \ldots, \mathcal{R}^F_{(1,\varphi_p,d_p)}) \\ \mathcal{R}^F_{t_2} & = & \mathcal{EXP}_2(\mathcal{R}^F_{(1,\xi_1,n_1)}, \ldots, \mathcal{R}^F_{(1,\xi_q,n_q)}) \end{array}$$

le paramètre n doit être un multiple commun de d_i, $\forall i = \overline{1,p}$, et de n_j, $\forall j = \overline{1,q}$. On peut ainsi s'apercevoir que la décomposition d'une relation $\mathcal{R}^F_t$ dépend du contexte de calcul : la nécessité de comparer $\mathcal{R}^F_{t_1}$ avec $\mathcal{R}^F_{t_3} \neq \mathcal{R}^F_{t_2}$ induit d'autres contraintes sur le paramètre n. Pour un contexte fixé, déterminé par un ensemble de relations affines $\mathcal{R}^F_{(1,\varphi_i,d_i)}$, avec $\varphi_i \geq 0$, et d'expressions construites sur ces relations affines, on peut considérer n le plus petit multiple commun (ppcm) des périodes d_i existantes dans le contexte. Dans ce cas, il existe une décomposition unique relative à n – et donc au contexte considéré – de toute relation $\mathcal{R}^F_t$: on l'appellera *forme normale relative* au contexte de calcul. La forme normale relative permettra de décider de l'identifiabilité de deux variables, quand celle-ci fait intervenir des relations $\mathcal{R}^F_{t_1}$ et $\mathcal{R}^F_{t_2}$ appartenant au contexte.

7.2 Relation affine & Processus

Dans cette section nous étendons aux processus le concept de relation affine introduit au niveau des flots. Nous commençons par définir la relation de (f_1, f_2)-codensification dans un processus $\mathbf{P}$.

- $x \ \mathcal{R}^{\mathbf{P}}_{(f_1,f_2)} \ y$

  Pour f_1, f_2 fonctions de densification, les variables x et y sont (f_1, f_2)*-codensifiées* dans le processus $\mathbf{P}$ ssi elles sont (f_1, f_2)-codensifiées dans tout flot F appartenant à $\mathbf{P}$: $x \ \mathcal{R}^F_{(f_1,f_2)} \ y$, $\forall \ F \in \ \mathbf{P}$.

- Pour tout processus $\mathbf{P} \in \mathcal{P}_{A_1}$, *la relation de* (f_1, f_2)*-codensification* $\mathcal{R}^{\mathbf{P}}_{(f_1,f_2)}$ représente l'ensemble des couples $(x, y) \in A_1 \times A_1$, $x \ \mathcal{R}^{\mathbf{P}}_{(f_1,f_2)} \ y$:

$$\mathcal{R}^{\mathbf{P}}_{(f_1,f_2)} = \{(x,y) \in A_1 \times A_1 \ / \ \ x \ \mathcal{R}^{\mathbf{P}}_{(f_1,f_2)} \ y \ \ \}$$

Pour f_1, f_2 fonctions de densification affines, la relation de (f_1, f_2)-codensification devient une relation affine ; nous présentons la définition en section 7.2.1. Nous étendons ensuite aux processus tous les résultats obtenus sur les flots et concluons avec la *forme normale relative* pour toute expression $\mathcal{R}^{\mathbf{P}}_t$.

7.2.1 Définition

- $x \ \mathcal{R}^{\mathbf{P}}_{((n,\varphi_1),(d,\varphi_2))} \ y$

  Les variables x et y sont *affine-codensifiées* dans le processus $\mathbf{P}$ ssi elles sont affine-codensifiées dans tout flot F appartenant à $\mathbf{P}$: $x \ \mathcal{R}^F_{((n,\varphi_1),(d,\varphi_2))} \ y$, $\forall \ F \in \ \mathbf{P}$.

Dans ce cas, on dit aussi qu'il existe une relation affine de paramètres $((n,\varphi_1),(d,\varphi_2))$ entre x et y.

- Pour tout processus $\mathbf{P} \in \mathcal{P}_{A_1}$, *la relation affine* $\mathcal{R}^{\mathbf{P}}_{((n,\varphi_1),(d,\varphi_2))}$ représente l'ensemble des couples $(x,y) \in A_1 \times A_1,\ x\ \mathcal{R}^{\mathbf{P}}_{((n,\varphi_1),(d,\varphi_2))}\ y$:

$$\mathcal{R}^{\mathbf{P}}_{((n,\varphi_1),(d,\varphi_2))} = \{(x,y) \in A_1 \times A_1 \ /\ \ x\ \mathcal{R}^{\mathbf{P}}_{((n,\varphi_1),(d,\varphi_2))}\ y\ \ \}$$

7.2.2 Normalisation d'une relation affine

Soit $\mathbf{P} \in \mathcal{P}_{A_1}$ un processus arbitraire et les paramètres $n, d, \varphi_1, \varphi_2$ satisfaisant $n, d, \in \mathbb{N}^*$ et $\varphi_1, \varphi_2 \in \mathbb{N}$. La relation $\mathcal{R}^{\mathbf{P}}_{((n,\varphi_1),(d,\varphi_2))}$ admet $\mathcal{R}^{\mathbf{P}}_{((n,max(0,-\varphi)),\ (d,max(0,\varphi)))}$, avec $\varphi = \varphi_2 - \varphi_1$, comme forme normale ; elle sera notée $\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)}$ dans la suite. Nous introduisons le corollaire de normalisation à travers le lemme de translation démontré ci-dessous.

Lemme 12 *(Translation)* $\forall\ c \in \mathbb{Z}$ *tel que* $\varphi_1 + c \geq 0$ *et* $\varphi_2 + c \geq 0$ *on a*

$$\mathcal{R}^{\mathbf{P}}_{((n,\varphi_1),(d,\varphi_2))} = \mathcal{R}^{\mathbf{P}}_{((n,\varphi_1+c),(d,\varphi_2+c))}$$

Preuve

$$\begin{array}{rcl} (x,y) \in \mathcal{R}^{\mathbf{P}}_{((n,\varphi_1),(d,\varphi_2))} & \Leftrightarrow & \forall\ F \in \mathbf{P}, (x,y) \in \mathcal{R}^{F}_{((n,\varphi_1),(d,\varphi_2))} \\ & \overset{Lemme\ 2}{\Leftrightarrow} & \forall\ F \in \mathbf{P}, (x,y) \in \mathcal{R}^{F}_{((n,\varphi_1+c),(d,\varphi_2+c))} \\ & \Leftrightarrow & (x,y) \in \mathcal{R}^{\mathbf{P}}_{((n,\varphi_1+c),(d,\varphi_2+c))} \end{array}$$

□

Pour une relation affine $\mathcal{R}^{\mathbf{P}}_{((n,\varphi_1),(d,\varphi_2))}$, le corollaire suivant se déduit directement du lemme précédent, en prenant $c = max(0,\varphi) - \varphi_2$, avec $\varphi = \varphi_2 - \varphi_1$. De manière générale, les preuves des résultats que nous présentons dans la suite suivent un schéma similaire à celui de la démonstration précédente ; nous ne les introduisons que dans les cas où ce schéma n'est pas respecté.

Corollaire 4 *Soit* $\varphi = \varphi_2 - \varphi_1$. *Alors*

$$\mathcal{R}^{\mathbf{P}}_{((n,\varphi_1),(d,\varphi_2))} = \mathcal{R}^{\mathbf{P}}_{((\ n,max(0,-\varphi)\ ),\ (\ d,max(0,\varphi)\ ))}$$

□

La relation $\mathcal{R}^{\mathbf{P}}_{((\ n,max(0,-\varphi)\ ),\ (\ d,max(0,\varphi)\ ))}$ représente la *forme normale* de $\mathcal{R}^{\mathbf{P}}_{((n,\varphi_1),(d,\varphi_2))}$ et sera notée $\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)}$ dans la suite. Le Corollaire 4 se réécrit dans le Corollaire 5 de normalisation comme suit :

Corollaire 5 *(Normalisation)* $\forall\ \varphi_1, \varphi_2\ \in \mathbb{N}$

$$\mathcal{R}^{\mathbf{P}}_{((n,\varphi_1),(d,\varphi_2))} = \mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)}, \qquad \textit{avec } \varphi = \varphi_2 - \varphi_1$$

□

Dans la suite, toutes les relations $\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)}$ sont considérées non vides et toutes les propriétés déduites pour $\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)}$ sont valables pour toute relation $\mathcal{R}^{\mathbf{P}}_{((n,\varphi_1),(d,\varphi_2))}$ telle que $\varphi = \varphi_2 - \varphi_1$.

7.2.3 Équivalence de relations affines & Forme canonique induite

Définition 13 *Pour tout processus $\mathbf{P} \in \mathcal{P}_{A_1}$, les relations $\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)}$ et $\mathcal{R}^{\mathbf{P}}_{(n',\varphi',d')}$ sont égales ssi la relation suivante est vérifiée :*

$$x \ \mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)} \ y \quad \Leftrightarrow \quad x \ \mathcal{R}^{\mathbf{P}}_{(n',\varphi',d')} \ y \ , \ \ \forall \ x, y \in A_1$$

□

Lemme 13 *(Égalité de relations affines)*
Pour tout processus $\mathbf{P} \in \mathcal{P}_{A_1}$,

$$\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)} = \mathcal{R}^{\mathbf{P}}_{(n',\varphi',d')} \ \Leftrightarrow \ (n,\varphi,d) \ \sim (n',\varphi',d')$$

Preuve L'identité

$$\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)} = \mathcal{R}^{\mathbf{P}}_{(n',\varphi',d')}$$

est équivalente à

$$(\ \ x \ \mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)} \ y \quad \Leftrightarrow \quad x \ \mathcal{R}^{\mathbf{P}}_{(n',\varphi',d')} \ y \ , \ \ \forall \ x, y \in A_1 \ )$$

où encore à

$$(\ (\ x \ \mathcal{R}^{F}_{(n,\varphi,d)} \ y \ , \forall \ F \in \mathbf{P}) \ \Leftrightarrow \ (\ x \ \mathcal{R}^{F}_{(n',\varphi',d')} \ y \ , \forall \ F \in \mathbf{P}), \ \ \forall \ x, y \in A_1 \ )$$

- On suppose $\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)} = \mathcal{R}^{\mathbf{P}}_{(n',\varphi',d')}$ et on démontre $(n,\varphi,d) \ \sim (n',\varphi',d')$.
  Soit $F \in \mathbf{P}$ pour lequel les implications suivantes sont valides :

$$\begin{array}{rcl} x \ \mathcal{R}^{F}_{(n,\varphi,d)} \ y & \Rightarrow & x \ \mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)} \ y \ , \ \forall x, y \in A_1 \\ x' \ \mathcal{R}^{F}_{(n',\varphi',d')} \ y' & \Rightarrow & x' \ \mathcal{R}^{\mathbf{P}}_{(n',\varphi',d')} \ y' \ , \ \forall x', y' \in A_1 \end{array}$$

  (comme $\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)} = \mathcal{R}^{\mathbf{P}}_{(n',\varphi',d')}$ non vides, il existe un flot F avec les propriétés précédentes).
  Dans ces hypothèses, on obtient la suite d'implications :

$$\begin{array}{lcccccc} x \ \mathcal{R}^{F}_{(n,\varphi,d)} \ y & \Rightarrow & x \ \mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)} \ y & \Rightarrow & x \ \mathcal{R}^{\mathbf{P}}_{(n',\varphi',d')} \ y & \Rightarrow \\ & \Rightarrow & x \ \mathcal{R}^{F}_{(n',\varphi',d')} \ y & \Rightarrow & \mathcal{R}^{F}_{(n,\varphi,d)} \subseteq \mathcal{R}^{F}_{(n',\varphi',d')} & \\ x' \ \mathcal{R}^{F}_{(n',\varphi',d')} \ y' & \Rightarrow & x' \ \mathcal{R}^{\mathbf{P}}_{(n',\varphi',d')} \ y' & \Rightarrow & x' \ \mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)} \ y' & \Rightarrow \\ & \Rightarrow & x' \ \mathcal{R}^{F}_{(n,\varphi,d)} \ y' & \Rightarrow & \mathcal{R}^{F}_{(n',\varphi',d')} \subseteq \mathcal{R}^{F}_{(n,\varphi,d)} & \end{array}$$

  Il résulte $\mathcal{R}^{F}_{(n',\varphi',d')} = \mathcal{R}^{F}_{(n,\varphi,d)}$ et cette égalité implique l'équivalence de (n,φ,d) et (n',φ',d') (conformément aux Proposition 1 et Définition 5, section 7.1.3).

- Réciproquement, $(n,\varphi,d) \ \sim (n',\varphi',d')$ implique $\mathcal{R}^{F}_{(n,\varphi,d)} = \mathcal{R}^{F}_{(n',\varphi',d')}$, $\forall \ F \in \mathbf{P}$ et dans ces conditions la relation suivante est vérifiée :

$$(\ (\ x \ \mathcal{R}^{F}_{(n,\varphi,d)} \ y \ , \forall \ F \in \mathbf{P}) \ \Leftrightarrow \ (\ x \ \mathcal{R}^{F}_{(n',\varphi',d')} \ y \ , \forall \ F \in \mathbf{P}), \ \ \forall \ x, y \in A_1 \ )$$

□

En section 7.1.4 nous avons montré qu'il existe un représentant unique de la classe d'équivalence de (n,φ,d) par rapport à la relation $\sim$; nous l'avons noté $(n_{FC},\varphi_{FC},d_{FC})$ et l'avons désigné comme forme canonique de (n,φ,d). Par extension, la forme canonique de $\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)}$ est $\mathcal{R}^{\mathbf{P}}_{(n_{FC},\varphi_{FC},d_{FC})}$; le test d'égalité des relations $\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)}$ et $\mathcal{R}^{\mathbf{P}}_{(n',\varphi',d')}$ se réduit ainsi au test d'égalité de leurs formes canoniques $\mathcal{R}^{\mathbf{P}}_{(n_{FC},\varphi_{FC},d_{FC})}$ et $\mathcal{R}^{\mathbf{P}}_{(n'_{FC},\varphi'_{FC},d'_{FC})}$, qui implique la comparaison terme à terme des triplets $(n_{FC},\varphi_{FC},d_{FC})$ et $(n'_{FC},\varphi'_{FC},d'_{FC})$, comme indiqué dans le corollaire suivant.

Corollaire 6

$$\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)} \;=\; \mathcal{R}^{\mathbf{P}}_{(n',\varphi',d')} \;\Leftrightarrow (n_{FC},\varphi_{FC},d_{FC}) = (n'_{FC},\varphi'_{FC},d'_{FC})$$

□

7.2.4 Opérations sur relations affines & Expressions induites

L'ensemble des résultats concernant les opérations qui peuvent être réalisées sur les relations affines associées aux flots sont valides dans le contexte des processus à travers les propriétés du quantificateur universel présent dans la définition d'une relation $\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)}$: $(x,y) \in \mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)}$ ssi $(x,y) \in \mathcal{R}^{F}_{(n,\varphi,d)}, \forall F \in \mathbf{P}$.

7.2.4.1 Inverse d'une relation affine

Définition 14 *L'inverse de la relation affine* $\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)}$ *est la relation* $\left(\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)}\right)^{-1}$ *telle que :*

$$x\ \mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)}\ y \;\Leftrightarrow\; y\ \left(\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)}\right)^{-1} x\ ,\;\; \forall\ x,y \in A_1$$

□

L'inverse d'une relation affine reste une relation affine, comme le démontre le lemme suivant :

Lemme 14 *Pour tout processus* $\mathbf{P} \in \mathcal{P}_{A_1}$,

$$\left(\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)}\right)^{-1} = \mathcal{R}^{\mathbf{P}}_{(d,-\varphi,n)}$$

□

7.2.4.2 Composition de relations affines

Définition 15 *Pour tout processus* $\mathbf{P} \in \mathcal{P}_{A_1}$*, la composition de* $\mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)}$ *et* $\mathcal{R}^{\mathbf{P}}_{(n_2,\varphi_2,d_2)}$*, notée* $*$*, est définie comme suit :*

$$\mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)} \;*\; \mathcal{R}^{\mathbf{P}}_{(n_2,\varphi_2,d_2)} = \{(x,y) \in A_1 \times A_1 \;/\; (\exists)u \in A_1, (x,u) \in \mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)}\ \textit{et}\ (u,y) \in \mathcal{R}^{\mathbf{P}}_{(n_2,\varphi_2,d_2)}\}$$

Nous considérons (C_c) le système de contraintes défini en section 7.1.5.2. Les Propositions 7 et 8 résument les principales propriétés de l'opérateur de composition défini précédemment :

Proposition 7 *$\forall$ $\mathbf{P} \in \mathcal{P}_{\{x,y,z\}}$ satisfaisant $x \ \mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)} \ y$ et $y \ \mathcal{R}^{\mathbf{P}}_{(n_2,\varphi_2,d_2)} \ z$, la relation suivante est valide :*

$$(C_c) \quad \Rightarrow \quad \mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)} \ * \ \mathcal{R}^{\mathbf{P}}_{(n_2,\varphi_2,d_2)} = \mathcal{R}^{\mathbf{P}}_{(n_1 n_2, n_2\varphi_1 + d_1\varphi_2, d_1 d_2)}$$

□

Proposition 8 *Soit $\mathbf{P} \in \mathcal{P}_{\{x,y,z\}}$ défini comme suit :*

$$\mathbf{P} = \{F \in \mathcal{F}_{\{x,y,z\}} \mid x \ \mathcal{R}^{F}_{(n_1,\varphi_1,d_1)} \ y \quad et \ y \ \mathcal{R}^{F}_{(n_2,\varphi_2,d_2)} \ z\}$$

Pour toute paire de triplets (n'_1, φ'_1, d'_1), (n'_2, φ'_2, d'_2) satisfaisant :

$$\begin{array}{l} (n'_1, \varphi'_1, d'_1) \sim (n_1, \varphi_1, d_1) \\ (n'_2, \varphi'_2, d'_2) \sim (n_2, \varphi_2, d_2) \\ d'_1 = n'_2 \end{array}$$

il existe un processus $\mathbf{P}_s \subseteq \mathbf{P}$ tel que

$$\mathcal{R}^{\mathbf{P}_s}_{(n_1,\varphi_1,d_1)} \ * \ \mathcal{R}^{\mathbf{P}_s}_{(n_2,\varphi_2,d_2)} = \mathcal{R}^{\mathbf{P}_s}_{(n'_1 n'_2, n'_2\varphi'_1 + d'_1\varphi'_2, d'_1 d'_2)}$$

Preuve Pour (n'_1, φ'_1, d'_1) et (n'_2, φ'_2, d'_2) satisfaisant les conditions de l'hypothèse, on défini $\mathbf{P}_s$ comme suit :

$$\begin{array}{rcl} \mathbf{P}_s & = & \{F \in \mathcal{F}_{\{x,y,z\}} \mid \ x \ \mathcal{R}^{F}_{(n'_1,\varphi'_1,d'_1)} \ y \ , \ y \ \mathcal{R}^{F}_{(n'_2,\varphi'_2,d'_2)} \ z \\ & & \qquad \text{et} \ \ x \ \mathcal{R}^{F}_{(n'_1 n'_2, n'_2\varphi'_1 + d'_1\varphi'_2, d'_1 d'_2)} \ z \ \} \end{array}$$

Cf. au Lemme 7 et Théorème 3, section 7.1.5.2, $\mathbf{P}_s \neq \emptyset$ et $\mathbf{P}_s \subseteq \mathbf{P}$.

Nous considérons le cas général $\mathcal{R}^{\mathbf{P}_s}_{(n'_1 n'_2, n'_2\varphi'_1 + d'_1\varphi'_2, d'_1 d'_2)} = \{(x,z)\}$ (dans le cas où la relation affine $\mathcal{R}^{\mathbf{P}_s}_{(n'_1 n'_2, n'_2\varphi'_1 + d'_1\varphi'_2, d'_1 d'_2)}$ contient au moins un autre élément différent de (x,z), on obtient des contraintes supplémentaires sur les valeurs des paramètres (n'_1, φ'_1, d'_1) et (n'_2, φ'_2, d'_2) et ensuite sur celles de (n_1, φ_1, d_1) et (n_2, φ_2, d_2) et les cas particuliers obtenus se traitent de manière similaire au cas général que nous considérons).

Les équivalences $(n'_1, \varphi'_1, d'_1) \sim (n_1, \varphi_1, d_1)$ et $(n'_2, \varphi'_2, d'_2) \sim (n_2, \varphi_2, d_2)$ induisent $x \ \mathcal{R}^{\mathbf{P}_s}_{(n_1,\varphi_1,d_1)} \ y$ et $y \ \mathcal{R}^{\mathbf{P}_s}_{(n_2,\varphi_2,d_2)} \ z$.

Comme $\mathcal{R}^{\mathbf{P}_s}_{(n'_1 n'_2, n'_2\varphi'_1 + d'_1\varphi'_2, d'_1 d'_2)} = \{(x,z)\}$ et $x \ \mathcal{R}^{\mathbf{P}_s}_{(n_1,\varphi_1,d_1)} \ y$, $y \ \mathcal{R}^{\mathbf{P}_s}_{(n_2,\varphi_2,d_2)} \ z$, alors

$$\mathcal{R}^{\mathbf{P}_s}_{(n_1,\varphi_1,d_1)} \ * \ \mathcal{R}^{\mathbf{P}_s}_{(n_2,\varphi_2,d_2)} = \mathcal{R}^{\mathbf{P}_s}_{(n'_1 n'_2, n'_2\varphi'_1 + d'_1\varphi'_2, d'_1 d'_2)}$$

□

7.2.4.3 Élément neutre (par rapport à la composition)

Définition 16 *Pour tout processus $\mathbf{P} \in \mathcal{F}_{A_1}$, $\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)}$ est une relation d'identité dans l'espace des relations affines associées à $\mathbf{P}$ ssi la relation suivante est vérifiée :*

$$x \ \mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)} \ y \quad \Leftrightarrow \quad x \overset{\mathbf{P}}{=} y$$

□

La relation $\mathcal{R}^{\mathbf{P}}_{(1,0,1)}$ constitue l'élément identité dans l'espace des relations affines.

Lemme 15 *Pour tout processus* $\mathbf{P} \in \mathcal{P}_{A_1}$,

$$\mathcal{R}^{\mathbf{P}}_{(1,0,1)} = \{(x,y) \in A_1 \times A_1 \;/\; x \overset{\mathbf{P}}{=} y\}$$

La relation affine d'identité $\mathcal{R}^{\mathbf{P}}_{(1,0,1)}$ est élément neutre par rapport à l'opérateur de composition dans l'espace des relations affines $\mathcal{R}^{\mathbf{P}}_{(n,\varphi,d)}$ et plus généralement dans l'espace des relations de codensification $\mathcal{R}^{\mathbf{P}}_{(f_1,f_2)}$:

Lemme 16 *Pour tout processus* $\mathbf{P} \in \mathcal{P}_{A_1}$ *et pour toute relation* $\mathcal{R}^{\mathbf{P}}_{(f_1,f_2)}$, *on a :*

$$\mathcal{R}^{\mathbf{P}}_{(f_1,f_2)} \;*\; \mathcal{R}^{\mathbf{P}}_{(1,0,1)} = \mathcal{R}^{\mathbf{P}}_{(f_1,f_2)}$$
$$\mathcal{R}^{\mathbf{P}}_{(1,0,1)} \;*\; \mathcal{R}^{\mathbf{P}}_{(f_1,f_2)} = \mathcal{R}^{\mathbf{P}}_{(f_1,f_2)}$$

avec f_1, f_2 *densifications arbitraires.*

7.2.4.4 Union de relations affines

Définition 17 *Pour tout processus* $\mathbf{P} \in \mathcal{P}_{A_1}$, *l'union de* $\mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)}$ *et* $\mathcal{R}^{\mathbf{P}}_{(n_2,\varphi_2,d_2)}$, *notée* $\cup_r$, *est définie comme suit :*

$$\mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)} \cup_r \mathcal{R}^{\mathbf{P}}_{(n_2,\varphi_2,d_2)} = \{(x,y) \in A_1 \times A_1 \;/\; \exists\ x_1, x_2 \in A_1, (x,x_1) \in \mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)}, (x,x_2) \in \mathcal{R}^{\mathbf{P}}_{(n_2,\varphi_2,d_2)} \text{ et } y \overset{\mathbf{P}}{=} x_1 \vee_v x_2\}$$

□

Nous considérons (C_u) le système de contraintes défini en section 7.1.5.4. La Proposition 9 reprend au niveau des processus les résultats de la Proposition 3, section 7.1.5.4.

Proposition 9 $\forall\ \mathbf{P} \in \mathcal{P}_{\{x,x_1,x_2,y\}}$ *satisfaisant* $x\ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_1,d_1)}\ x_1$, $x\ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_2,d_2)}\ x_2$ *et* $y \overset{\mathbf{P}}{=} x_1 \vee_v x_2$, *la relation suivante est vérifiée :*

$$\mathcal{R}^{\mathbf{P}}_{(1,\varphi_1,d_1)} \cup_r \mathcal{R}^{\mathbf{P}}_{(1,\varphi_2,d_2)} = \mathcal{R}^{\mathbf{P}}_{(1,\varphi,d)} \quad \Leftrightarrow \quad (C_u)$$

7.2.4.5 Intersection de relations affines

Définition 18 *Pour tout processus* $\mathbf{P} \in \mathcal{P}_{A_1}$, *l'intersection de* $\mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)}$ *et* $\mathcal{R}^{\mathbf{P}}_{(n_2,\varphi_2,d_2)}$, *notée* $\cap_r$, *est définie comme suit :*

$$\mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)} \cap_r \mathcal{R}^{\mathbf{P}}_{(n_2,\varphi_2,d_2)} = \{(x,y) \in A_1 \times A_1 \;/\; \exists\ x_1, x_2 \in A_1, (x,x_1) \in \mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)}, (x,x_2) \in \mathcal{R}^{\mathbf{P}}_{(n_2,\varphi_2,d_2)} \text{ et } y \overset{\mathbf{P}}{=} x_1 \wedge_v x_2\}$$

□

Soit (C_i) le système de contraintes défini en section 7.1.5.5. La Proposition 10 suivante résume les principaux résultats concernant l'intersection de relations affines associées aux processus.

Proposition 10 *$\forall$ $\mathbf{P} \in \mathcal{P}_{\{x,x_1,x_2,y\}}$ satisfaisant $x \; \mathcal{R}^{\mathbf{P}}_{(1,\varphi_1,d_1)} \; x_1$, $x \; \mathcal{R}^{\mathbf{P}}_{(1,\varphi_2,d_2)} \; x_2$ et $y \overset{\mathbf{P}}{=} x_1 \wedge_v x_2$, la relation suivante est valide :*

$$\mathcal{R}^{\mathbf{P}}_{(1,\varphi_1,d_1)} \cap_r \mathcal{R}^{\mathbf{P}}_{(1,\varphi_2,d_2)} = \mathcal{R}^{\mathbf{P}}_{(1,\varphi,d)} \quad \Leftrightarrow \quad (C_i)$$

7.2.4.6 Différence de relations affines

Définition 19 *Pour tout processus $\mathbf{P} \in \mathcal{P}_{A_1}$, la différence de $\mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)}$ et $\mathcal{R}^{\mathbf{P}}_{(n_2,\varphi_2,d_2)}$, notée $\setminus_r$, est définie comme suit :*

$$\mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)} \setminus_r \mathcal{R}^{\mathbf{P}}_{(n_2,\varphi_2,d_2)} = \{(x,y) \in A_1 \times A_1 \; / \; \exists \; x_1, x_2 \in A_1, (x,x_1) \in \mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)}, \\ (x,x_2) \in \mathcal{R}^{\mathbf{P}}_{(n_2,\varphi_2,d_2)} \; et \; y \overset{\mathbf{P}}{=} x_1 \setminus_v x_2\}$$

□

Avec (C_d) le système de contraintes défini en section 7.1.5.6, nous concluons cette partie sur les opérations sur relations affines associées aux processus avec un résultat sur la différence de relations affines :

Proposition 11 *$\forall$ $P \in \mathcal{P}_{\{x,x_1,x_2,y\}}$ satisfaisant $x \; \mathcal{R}^{\mathbf{P}}_{(1,\varphi_1,d_1)} \; x_1$, $x \; \mathcal{R}^{\mathbf{P}}_{(1,\varphi_2,d_2)} \; x_2$ et $y \overset{\mathbf{P}}{=} x_1 \setminus_v x_2$, la relation suivante est valide :*

$$\mathcal{R}^{\mathbf{P}}_{(1,\varphi_1,d_1)} \setminus_r \mathcal{R}^{\mathbf{P}}_{(1,\varphi_2,d_2)} = \mathcal{R}^{\mathbf{P}}_{(1,\varphi,d)} \quad \Leftrightarrow \quad (C_d)$$

7.2.4.7 Forme normale relative d'expressions sur relations affines

Nous étendons aux processus les principaux résultats sur les relations affines $\mathcal{R}^{\mathbf{P}}_t$ construites comme expressions avec au plus les opérateurs $\{\cup_r, \cap_r, \setminus_r\}$ sur des relations affines $\mathcal{R}^{\mathbf{P}}_{(1,\varphi_i,d_i)}$, où $\varphi_i \geq 0$.

Lemme 17 *Soit $\mathbf{P} \in \mathcal{P}_{A_1}$ tel que les relations non vides $\mathcal{R}^{\mathbf{P}}_{t_1}$ et $\mathcal{R}^{\mathbf{P}}_{t_2}$ se décomposent en*

$$\mathcal{R}^{\mathbf{P}}_{t_1} = (\cup_r)_{\varphi \in I_1} \mathcal{R}^{\mathbf{P}}_{(1,\varphi,n)}, \; \mathcal{R}^{\mathbf{P}}_{t_2} = (\cup_r)_{\varphi \in I_2} \mathcal{R}^{\mathbf{P}}_{(1,\varphi,n)}$$

avec $I_1, I_2 \subseteq \mathbb{N}$ finis et $n \in \mathbb{N}^*$ arbitraire.

Alors :

$$\mathcal{R}^{\mathbf{P}}_{t_1} = \mathcal{R}^{\mathbf{P}}_{t_2} \quad \Leftrightarrow \quad I_1 = I_2$$

Preuve

- On suppose $\mathcal{R}_{t_1}^{\mathbf{P}} = \mathcal{R}_{t_2}^{\mathbf{P}}$ et on démontre $I_1 = I_2$.

  L'identité
  $$\mathcal{R}_{t_1}^{\mathbf{P}} = \mathcal{R}_{t_2}^{\mathbf{P}}$$
  est équivalente à
  $$(\ x\ \mathcal{R}_{t_1}^{\mathbf{P}}\ y \ \Leftrightarrow\ x\ \mathcal{R}_{t_2}^{\mathbf{P}}\ y\ ,\ \forall\ x,y \in A_1\ )$$
  où encore à
  $$(\ (\ x\ \mathcal{R}_{t_1}^{F}\ y\ , \forall\ F \in \mathbf{P})\ \Leftrightarrow\ (\ x\ \mathcal{R}_{t_2}^{F}\ y\ , \forall\ F \in \mathbf{P}),\ \forall\ x,y \in A_1\ )$$
  Soit $F \in \mathbf{P}$ dans lequel les relations suivantes sont valides : 1) $x\ \mathcal{R}_{t_1}^{F}\ y \ \Rightarrow\ x\ \mathcal{R}_{t_1}^{\mathbf{P}}\ y$, $\forall x,y \in A_1$; 2) $x\ \mathcal{R}_{t_2}^{F}\ y \ \Rightarrow\ x\ \mathcal{R}_{t_2}^{\mathbf{P}}\ y$, $\forall x,y \in A_1$ (A_1 ensemble fini de variables et $\mathcal{R}_{t_1}^{\mathbf{P}} = \mathcal{R}_{t_2}^{\mathbf{P}}$ non vides induisent l'existence de F).

  On obtient la suite d'implications suivante :
  $$x\ \mathcal{R}_{t_1}^{F}\ y\ \Rightarrow\ x\ \mathcal{R}_{t_1}^{\mathbf{P}}\ y\ \Rightarrow\ x\ \mathcal{R}_{t_2}^{\mathbf{P}}\ y\ \Rightarrow\ x\ \mathcal{R}_{t_2}^{F}\ y\ \Rightarrow \mathcal{R}_{t_1}^{F} \subseteq \mathcal{R}_{t_2}^{F}$$
  $$x\ \mathcal{R}_{t_2}^{F}\ y\ \Rightarrow\ x\ \mathcal{R}_{t_2}^{\mathbf{P}}\ y\ \Rightarrow\ x\ \mathcal{R}_{t_1}^{\mathbf{P}}\ y\ \Rightarrow\ x\ \mathcal{R}_{t_1}^{F}\ y\ \Rightarrow \mathcal{R}_{t_2}^{F} \subseteq \mathcal{R}_{t_1}^{F}$$
  Il résulte $\mathcal{R}_{t_2}^{F} = \mathcal{R}_{t_1}^{F}$ et en utilisant Lemme 11, section 7.1.5.7, on obtient $I_1 = I_2$.

- Réciproquement, on suppose $I_1 = I_2$ et on démontre $\mathcal{R}_{t_1}^{\mathbf{P}} = \mathcal{R}_{t_2}^{\mathbf{P}}$.

  $I_1 = I_2$ implique $\mathcal{R}_{t_1}^{F} = \mathcal{R}_{t_2}^{F}$, $\forall\ F \in \mathbf{P}$ (Lemme 11, section 7.1.5.7) et dans ces conditions la relation suivante est vérifiée :
  $$(\ (\ x\ \mathcal{R}_{t_1}^{F}\ y\ , \forall\ F \in \mathbf{P})\ \Leftrightarrow\ (\ x\ \mathcal{R}_{t_2}^{F}\ y\ , \forall\ F \in \mathbf{P}),\ \forall\ x,y \in A_1\ )$$
  équivalent à
  $$\mathcal{R}_{t_1}^{\mathbf{P}} = \mathcal{R}_{t_2}^{\mathbf{P}}$$

En conclusion $\mathcal{R}_{t_1}^{\mathbf{P}} = \mathcal{R}_{t_2}^{\mathbf{P}} \ \Leftrightarrow\ I_1 = I_2$ *(qed)*. □

Proposition 12 *Soit $\mathbf{P} \in \mathcal{P}_{A_1}$ et $\mathcal{R}_t^{\mathbf{P}}$ une relation non vide définie comme suit :*
$$\mathcal{R}_t^{\mathbf{P}} = \mathcal{EXP}(\mathcal{R}_{(1,\varphi_1,d_1)}^{\mathbf{P}}, \ldots, \mathcal{R}_{(1,\varphi_p,d_p)}^{\mathbf{P}})$$
avec $\mathcal{EXP}$ une expression arbitraire contenant les opérateurs $\cup_r$, $\cap_r$ et $\setminus_r$.

Si
$$\mathcal{R}_{(1,\varphi_i,d_i)}^{\mathbf{P}} = (\cup_r)_{\varphi \in I_i} \mathcal{R}_{(1,\varphi,n)}^{\mathbf{P}},\ \forall\ i = \overline{1,p}$$
avec $n \in I\!N^*$ un multiple commun de d_i, $\forall\ i = \overline{1,p}$, et $I_i = \{\varphi_i, \varphi_i + d_i, \ldots, \varphi_i + (\frac{n}{d_i} - 1)d_i\}$, $\forall\ i = \overline{1,p}$, alors
$$\mathcal{R}_t^{\mathbf{P}} = (\cup_r)_{\varphi \in Exp(I_1, \ldots, I_p)} \mathcal{R}_{(1,\varphi,n)}^{\mathbf{P}}$$
avec Exp l'expression sur ensembles correspondant à $\mathcal{EXP}$.

Preuve La preuve suit un schéma similaire à celui du lemme précédent et utilise la Proposition 6, section 7.1.5.7. □

7.3 Règles de synchronisabilité dans le contexte des relations affines

Le calcul d'horloges de SIGNAL est un calcul de synchronisations basé sur un certain nombre de *règles de synchronisabilité.* Nous nous proposons d'accroître le système de règles de synchronisabilité existant en utilisant les propriétés des relations affines que nous avons présentées dans les sections précédentes. Plus particulièrement, nous démontrons un certain nombre de règles d'*identifiabilité* de variables de même type (domaine des valeurs). Celles-ci sont ensuite applicables directement aux variables d'horloges. Or, pour les horloges, les relations *identifiable* et *synchronisable* sont équivalentes ; c'est pour cela que nous allons parler dans la suite de règles de synchronisabilité.

L'aspect le plus important que nous devons prendre en considération dans la suite est celui de la non transitivité de la relation de synchronisabilité (voir section 6.5.4.1) : en effet, de manière générale, la synchronisabilité n'est pas transitive, donc elle n'est pas une équivalence. En tenant compte de cet aspect, nous structurons notre présentation en deux parties. Nous introduisons d'abord un ensemble de règles dites de *synchronisme* puisque dans tous les cas analysés la synchronisabilité est équivalente au synchronisme. La relation synchrone étant transitive (c'est une équivalence), les règles que nous présentons forment un système minimal à partir duquel peut être déduite toute autre règle de synchronisme basée sur les propriétés des relations affines.

Dans la deuxième partie de notre présentation, nous introduisons un sous-ensemble de règles de synchronisabilité *générales* basées sur les propriétés de l'opérateur de composition de relations affines. Celles-ci forment le système minimal qui permet de déduire toute autre règle de synchronisabilité dans ce contexte, dans le cas où la relation synchronisable est transitive et le système minimal peut être appliqué. Le système de règles est minimal dans le sens suivant : chaque fois qu'on veut déduire une nouvelle règle et on se place dans des conditions où la relation synchronisable est transitive, l'ensemble des règles de synchronisabilité utilisées dans l'algorithme de déduction appartient au système minimal. La relation synchronisable doit être transitive afin de pouvoir construire un algorithme de déduction basé sur une suite de règles de synchronisabilité appartenant au système minimal.

L'ensemble de règles de synchronisabilité que nous démontrons est suffisant pour les besoins de preuve formelle que nous avons exposés dans la première partie de ce document (codesign SIGNAL -ALPHA). Nous discutons les problèmes que nous avons rencontrés à ce niveau et présentons les extensions possibles de notre calcul.

7.3.1 Règles de synchronisme

L'ensemble de règles que nous déduisons dans la suite sont basées sur les propriétés des expressions construites sur des relations affines $\mathcal{R}^F_{(1,\varphi_i,d_i)}$ et $\mathcal{R}^{\mathbf{P}}_{(1,\varphi_i,d_i)}$, avec $\varphi_i \geq 0$ et $d_i \geq 1$. Nous avons déduit un certain nombre de propriétés pour les opérations de composition, union, intersection et différence ; entre autres, nous avons constaté que l'espace des expressions sur ces relations affines est fermé par rapport aux opérations antérieures et avons mis en évidence une *forme normale relative* unique pour toute expression dans cet espace. Nous utilisons ces résultats dans les règles proposées dans la suite. Sauf autrement indiqué, les paramètres $(1, \varphi_i, d_i)$ de toute relation affine dans cette section satisfont : $\varphi_i \geq 0, d_i \geq 1$.

Dans un processus $\mathbf{P} \in \mathcal{P}_{\{x_1,x_2,u\}}$, deux variables x_1 et x_2 en relation affine de paramètres $(1, \varphi_1, d_1)$ et respectivement $(1, \varphi_2, d_2)$ avec la même variable u sont identifiables en $\mathbf{P}$ si et seulement si $(1, \varphi_1, d_1)$ et $(1, \varphi_2, d_2)$ coïncident : $\varphi_1 = \varphi_2$ et $d_1 = d_2$. Dans cette situation, x_1 et x_2 sont identiques en $\mathbf{P}$.

Théorème 4 *(Règle no. 1)* $\forall\ \mathbf{P} \in \mathcal{P}_{\{x_1,x_2,u\}}$ *satisfaisant les conditions :*

$$u\ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_1,d_1)}\ x_1 \ \textit{et}\ u\ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_2,d_2)}\ x_2, \quad \textit{avec}\ \varphi_1,\varphi_2 \geq 0$$

on a : $$x_1 \overset{\mathbf{P}}{\ominus} x_2 \ \Leftrightarrow\ \begin{cases} \varphi_1 = \varphi_2 \\ d_1 = d_2 \end{cases} \Leftrightarrow\ x_1 \overset{\mathbf{P}}{=} x_2.$$

Preuve Avec Lemme 14 et Proposition 7, section 7.2.4, on obtient :

$$\left.\begin{array}{r} \left.\begin{array}{l} u\ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_1,d_1)}\ x_1 \\ (\mathcal{R}^{\mathbf{P}}_{(1,\varphi_1,d_1)})^{-1} = \mathcal{R}^{\mathbf{P}}_{(d_1,-\varphi_1,1)} \end{array}\right\} \Rightarrow\ x_1\ \mathcal{R}^{\mathbf{P}}_{(d_1,-\varphi_1,1)}\ u \\ u\ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_2,d_2)}\ x_2 \\ \mathcal{R}^{\mathbf{P}}_{(d_1,-\varphi_1,1)}\ *\ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_2,d_2)} = \mathcal{R}^{\mathbf{P}}_{(d_1,\varphi_2-\varphi_1,d_2)} \end{array}\right\} \Rightarrow \quad x_1\ \mathcal{R}^{\mathbf{P}}_{(d_1,\varphi_2-\varphi_1,d_2)}\ x_2$$

Si $x_1 \overset{\mathbf{P}}{\ominus} x_2$, il existe un flot $F_s \in \mathbf{P}$ tel que $x_1 \overset{F_s}{=} x_2$ (voir section 6.5.4.2). Il résulte $(x_1,x_2),(x_2,x_1) \in \mathcal{R}^{F_s}_{(1,0,1)}$.

En utilisant Lemme 9, section 7.1.5.3, il résulte :

$$\left.\begin{array}{l} (u,x_1) \in \mathcal{R}^{F_s}_{(1,\varphi_1,d_1)} \\ (x_1,x_2) \in \mathcal{R}^{F_s}_{(1,0,1)} \\ \mathcal{R}^{F_s}_{(1,\varphi_1,d_1)}\ *\ \mathcal{R}^{F_s}_{(1,0,1)} = \mathcal{R}^{F_s}_{(1,\varphi_1,d_1)} \end{array}\right\} \Rightarrow \quad (u,x_2) \in \mathcal{R}^{F_s}_{(1,\varphi_1,d_1)}$$

De manière analogue on peut prouver $(u,x_1) \in \mathcal{R}^{F_s}_{(1,\varphi_2,d_2)}$.

Nous considérons le cas général[4] $\mathcal{R}^{F_s}_{(1,\varphi_1,d_1)} = \{(u,x_1),(u,x_2)\}$ et $\mathcal{R}^{F_s}_{(1,\varphi_2,d_2)} = \{(u,x_1),(u,x_2)\}$.

Conformément aux Corollaire 3 et Théorème 2, section 7.1.4, $\mathcal{R}^{F_s}_{(1,\varphi_1,d_1)} = \mathcal{R}^{F_s}_{(1,\varphi_2,d_2)}$ implique $(1,\varphi_1,d_1) = (1,\varphi_2,d_2)$ (puisque les triplets $(1,\varphi_1,d_1)$ et $(1,\varphi_2,d_2)$ sont en forme canonique), i.e. $\varphi_1 = \varphi_2$ et $d_1 = d_2$. En conséquence, $x_1\ \mathcal{R}^{\mathbf{P}}_{(d_1,\varphi_2-\varphi_1,d_2)}\ x_2$ est équivalent à $x_1\ \mathcal{R}^{\mathbf{P}}_{(d_1,0,d_1)}\ x_2$ et, en utilisant Proposition 1 et Définition 5, section 7.1.3, ainsi que Lemme 13, section 7.2.3, on a $\mathcal{R}^{\mathbf{P}}_{(d_1,0,d_1)} = \mathcal{R}^{\mathbf{P}}_{(1,0,1)}$ et donc $x_1\ \mathcal{R}^{\mathbf{P}}_{(1,0,1)}\ x_2$, i.e. $x_1 \overset{\mathbf{P}}{=} x_2$, conformément au Lemme 15, section 7.2.4.3.

Il a été démontré que : $x_1 \overset{\mathbf{P}}{\ominus} x_2 \ \Rightarrow\ \begin{cases} \varphi_1 = \varphi_2 \\ d_1 = d_2 \end{cases} \Rightarrow\ x_1 \overset{\mathbf{P}}{=} x_2$. Deux variables identiques étant évidemment identifiables (voir section 6.5.4.2), i.e. $x_1 \overset{\mathbf{P}}{=} x_2 \Rightarrow x_1 \overset{\mathbf{P}}{\ominus} x_2$, on obtient :

$$x_1 \overset{\mathbf{P}}{\ominus} x_2 \ \Leftrightarrow\ \begin{cases} \varphi_1 = \varphi_2 \\ d_1 = d_2 \end{cases} \Leftrightarrow x_1 \overset{\mathbf{P}}{=} x_2$$

□

Le théorème suivant est immédiat en utilisant Lemme 15, section 7.2.4.3, et le fait que deux variables identiques sont identifiables.

Théorème 5 *(Règle no. 2)* $\forall\ \mathbf{P} \in \mathcal{P}_{\{u,x\}}$ *tel que* $u\ \mathcal{R}^{\mathbf{P}}_{(1,0,1)}\ x$, *on a :* $x \overset{\mathbf{P}}{\ominus} u \ \Leftrightarrow\ x \overset{\mathbf{P}}{=} u.$

4. Le cas où une des relations affines $\mathcal{R}^{F_s}_{(1,\varphi_1,d_1)}$ et $\mathcal{R}^{F_s}_{(1,\varphi_2,d_2)}$ contient au moins un autre élément différent de (u,x_1) et (u,x_2) entraîne des contraintes supplémentaires sur les valeurs des paramètres $(1,\varphi_1,d_1)$ et $(1,\varphi_2,d_2)$ et on obtient des cas particuliers du cas général traité. Dans la suite du document nous allons considérer uniquement le cas général.

Les opérations de composition, union, intersection et différence de relations affines induisent des règles d'identifiabilité conformes avec leurs propriétés. Nous énonçons dans la suite la règle de base pour chaque type d'opération ; à partir de celles-ci il est possible de décider de l'identifiabilité de deux variables, dans un contexte dans lequel toute opération sur des relations affines $\mathcal{R}^{\mathbf{P}}_{(1,\varphi_i,d_i)}$, avec $\varphi_i \geq 0$ et $d_i \geq 1$, produit une relation affine.

Nous considérons les systèmes de contraintes (C_u), (C_i) et (C_d) tels qu'ils ont été définis en section 7.1.5.

Théorème 6 *(Règle no. 3)* $\forall\ \mathbf{P} \in \mathcal{P}_{\{u,x_1,x_2,x_3\}}$ *satisfaisant les conditions :*

$$u\ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_1,d_1)}\ x_1\ ,\quad x_1\ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_2,d_2)}\ x_2\quad \textit{et}\quad u\ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_3,d_3)}\ x_3\ ,\quad \textit{avec}\ \varphi_1,\varphi_2,\varphi_3 \geq 0$$

$$\textit{on a :}\quad x_2 \overset{\mathbf{P}}{\ominus} x_3 \ \Leftrightarrow\ \left\{\begin{array}{l}\varphi_1 + d_1\varphi_2 = \varphi_3\\ d_1 d_2 = d_3\end{array}\right. \ \Leftrightarrow\ x_2 \overset{\mathbf{P}}{=} x_3.$$

Preuve On note $\Phi = \varphi_1 + d_1\varphi_2$ et $D = d_1 d_2$.

Soit $\mathbf{P}_1 = \Pi_{\{u,x_1,x_2\}}(\mathbf{P})$. Conformément au Théorème 1, section 6.5.4.5, toutes les relations affines vraies dans $\mathbf{P}$ restent vraies dans $\mathbf{P}_1$: $u\ \mathcal{R}^{\mathbf{P}_1}_{(1,\varphi_1,d_1)}\ x_1$ et $x_1\ \mathcal{R}^{\mathbf{P}_1}_{(1,\varphi_2,d_2)}\ x_2$.

Avec Proposition 7, section 7.2.4.2, on obtient :

$$\left.\begin{array}{l} u\ \mathcal{R}^{\mathbf{P}_1}_{(1,\varphi_1,d_1)}\ x_1\\ x_1\ \mathcal{R}^{\mathbf{P}_1}_{(1,\varphi_2,d_2)}\ x_2\\ \mathcal{R}^{\mathbf{P}_1}_{(1,\varphi_1,d_1)} * \mathcal{R}^{\mathbf{P}_1}_{(1,\varphi_2,d_2)} = \mathcal{R}^{\mathbf{P}_1}_{(1,\Phi,D)}\end{array}\right\} \ \Rightarrow\ u\ \mathcal{R}^{\mathbf{P}_1}_{(1,\Phi,D)}\ x_2$$

En rappliquant à ce dernier résultat le Théorème 1, section 6.5.4.5, on obtient $u\ \mathcal{R}^{\mathbf{P}}_{(1,\Phi,D)}\ x_2$.

Si $x_2 \overset{\mathbf{P}}{\ominus} x_3$, il existe $F_s \in \mathbf{P}$ tel que $x_2 \overset{F_s}{=} x_3$. Alors $(x_2,x_3) \in \mathcal{R}^{F_s}_{(1,0,1)}$ et $(x_3,x_2) \in \mathcal{R}^{F_s}_{(1,0,1)}$ et comme $\mathcal{R}^{F_s}_{(1,0,1)}$ est élément neutre par rapport à la composition, $u\ \mathcal{R}^{\mathbf{P}}_{(1,\Phi,D)}\ x_2$ et $u\ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_3,d_3)}\ x_3$ impliquent $u\ \mathcal{R}^{F_s}_{(1,\Phi,D)}\ x_3$ et $u\ \mathcal{R}^{F_s}_{(1,\varphi_3,d_3)}\ x_2$.

Nous considérons le cas général $\mathcal{R}^{F_s}_{(1,\Phi,D)} = \{(u,x_2),(u,x_3)\}$ et $\mathcal{R}^{F_s}_{(1,\varphi_3,d_3)} = \{(u,x_2),(u,x_3)\}$.

L'identité $\mathcal{R}^{F_s}_{(1,\varphi_3,d_3)} = \mathcal{R}^{F_s}_{(1,\Phi,D)}$ implique $(1,\varphi_3,d_3) = (1,\Phi,D)$, c'est-à-dire $\varphi_1 + d_1\varphi_2 = \varphi_3$ et $d_1 d_2 = d_3$.

$$\left\{\begin{array}{l}\Phi = \varphi_3\\ D = d_3\end{array}\right. \ \Leftrightarrow\ \left\{\begin{array}{l}\varphi_1 + d_1\varphi_2 = \varphi_3\\ d_1 d_2 = d_3\end{array}\right.$$

Soit $\mathbf{P}_2 = \Pi_{\{u,x_2,x_3\}}(\mathbf{P})$. On a $u\ \mathcal{R}^{\mathbf{P}_2}_{(1,\Phi,D)}\ x_2$ et $u\ \mathcal{R}^{\mathbf{P}_2}_{(1,\varphi_3,d_3)}\ x_3$ (voir Théorème 1, section 6.5.4.5).

Avec Lemme 14 et Proposition 7, section 7.2.4, on obtient :

$$(u,x_3) \in \mathcal{R}^{\mathbf{P}_2}_{(1,\varphi_3,d_3)} \ \Leftrightarrow\ (x_3,u) \in \left(\ \mathcal{R}^{\mathbf{P}_2}_{(1,\varphi_3,d_3)}\ \right)^{-1} \Leftrightarrow (x_3,u) \in \mathcal{R}^{\mathbf{P}_2}_{(d_3,-\varphi_3,1)}$$

$$\left.\begin{array}{l}(x_3,u) \in \mathcal{R}^{\mathbf{P}_2}_{(d_3,-\varphi_3,1)}\\ (u,x_2) \in \mathcal{R}^{\mathbf{P}_2}_{(1,\Phi,D)}\\ \mathcal{R}^{\mathbf{P}_2}_{(d_3,-\varphi_3,1)} * \mathcal{R}^{\mathbf{P}_2}_{(1,\Phi,D)} = \mathcal{R}^{\mathbf{P}_2}_{(d_3,\Phi-\varphi_3,D)}\end{array}\right\} \ \Rightarrow\ (x_3,x_2) \in \mathcal{R}^{\mathbf{P}_2}_{(d_3,\Phi-\varphi_3,D)}$$

$$\Rightarrow\ (x_3,x_2) \in \mathcal{R}^{\mathbf{P}_2}_{(D,0,D)}$$

Or $(D,0,D) \sim (1,0,1)$, ce qui implique $\mathcal{R}^{\mathbf{P}_2}_{(D,0,D)} = \mathcal{R}^{\mathbf{P}_2}_{(1,0,1)}$, c'est-à-dire $x_3 \overset{\mathbf{P}_2}{=} x_2$ et $x_3 \overset{\mathbf{P}}{=} x_2$ (voir Théorème 1, section 6.5.4.5).

On a démontré $x_2 \overset{\mathbf{P}}{\ominus} x_3 \;\Rightarrow\; \begin{cases} \varphi_1 + d_1\varphi_2 = \varphi_3 \\ d_1 d_2 = d_3 \end{cases} \;\Rightarrow\; x_2 \overset{\mathbf{P}}{=} x_3$. Un raisonnement similaire au théorème précédent permet de remplacer les implications par des équivalences (qed). □

Théorème 7 *(Règle no. 4)* $\forall\, \mathbf{P} \in \mathcal{P}_{\{u,x_1,x_2,x,x_3\}}$ *satisfaisant les conditions :*

1) $u\, \mathcal{R}^{\mathbf{P}}_{(1,\varphi_1,d_1)}\, x_1\ ,\quad u\, \mathcal{R}^{\mathbf{P}}_{(1,\varphi_2,d_2)}\, x_2$ *et* $u\, \mathcal{R}^{\mathbf{P}}_{(1,\varphi_3,d_3)}\, x_3\ ,$ *avec* $\varphi_1, \varphi_2, \varphi_3 \geq 0$
2) $x \overset{\mathbf{P}}{=} x_1 \vee_v x_2$

on a : $x \overset{\mathbf{P}}{\ominus} x_3 \;\Leftrightarrow\; (C_u) \;\Leftrightarrow\; x \overset{\mathbf{P}}{=} x_3.$

Preuve Si $x \overset{\mathbf{P}}{\ominus} x_3$, il existe $F_s \in \mathbf{P}$ tel que $x \overset{F_s}{=} x_3$. Alors $(x,x_3) \in \mathcal{R}^{F_s}_{(1,0,1)}$ et $(x_3,x) \in \mathcal{R}^{F_s}_{(1,0,1)}$ et comme $\mathcal{R}^{F_s}_{(1,0,1)}$ est élément neutre par rapport à la composition, les relations 1) et 2) impliquent $(u,x) \in \mathcal{R}^{F_s}_{(1,\varphi_3,d_3)}$ et $(u,x_3) \in \mathcal{R}^{F_s}_{(1,\varphi_1,d_1)} \cup_r \mathcal{R}^{F_s}_{(1,\varphi_2,d_2)}$.

Nous considérons le cas général $\mathcal{R}^{F_s}_{(1,\varphi_1,d_1)} \cup_r \mathcal{R}^{F_s}_{(1,\varphi_2,d_2)} = \mathcal{R}^{F_s}_{(1,\varphi_3,d_3)} = \{(u,x),(u,x_3)\}$.

Conformément à la Proposition 3, section 7.1.5.4, les valeurs des paramètres $d_1, \varphi_1, d_2, \varphi_2$ et d_3, φ_3 satisfont le système de contraintes (C_u).

Soit $\mathbf{P}_1 = \Pi_{\{u,x_1,x_2,x\}}(\mathbf{P})$. On a $u\, \mathcal{R}^{\mathbf{P}_1}_{(1,\varphi_1,d_1)}\, x_1$, $u\, \mathcal{R}^{\mathbf{P}_1}_{(1,\varphi_2,d_2)}\, x_2$ et $x \overset{\mathbf{P}_1}{=} x_1 \vee_v x_2$ (voir Théorème 1, section 6.5.4.5).

Conformément à la Proposition 9, section 7.2.4.4, il résulte $\mathcal{R}^{\mathbf{P}_1}_{(1,\varphi_1,d_1)} \cup_r \mathcal{R}^{\mathbf{P}_1}_{(1,\varphi_2,d_2)} = \mathcal{R}^{\mathbf{P}_1}_{(1,\varphi_3,d_3)}$ et en conséquence $(u,x) \in \mathcal{R}^{\mathbf{P}_1}_{(1,\varphi_3,d_3)}$, ce qui et équivalent ensuite à $(u,x) \in \mathcal{R}^{\mathbf{P}}_{(1,\varphi_3,d_3)}$.

Soit $\mathbf{P}_2 = \Pi_{\{u,x,x_3\}}(\mathbf{P})$. On a $u\, \mathcal{R}^{\mathbf{P}_2}_{(1,\varphi_3,d_3)}\, x$ et $u\, \mathcal{R}^{\mathbf{P}_2}_{(1,\varphi_3,d_3)}\, x_3$ (Théorème 1, section 6.5.4.5). A partir de ce résultat, en utilisant Lemme 14 et Proposition 7, section 7.2.4, ainsi que les propriétés de la relation d'équivalence de relations affines, une suite de pas similaires au théorème précédent nous conduit à $x_3 \overset{\mathbf{P}}{=} x$.

Nous avons démontré la suite d'implications : $x \overset{\mathbf{P}}{\ominus} x_3 \;\Rightarrow\; (C_u) \Rightarrow\; x \overset{\mathbf{P}}{=} x_3$. La règle no. 4 se déduit immédiatement avec l'implication évidente $x \overset{\mathbf{P}}{=} x_3 \;\Rightarrow\; x \overset{\mathbf{P}}{\ominus} x_3$. □

Théorème 8 *(Règle no. 5)* $\forall\, \mathbf{P} \in \mathcal{P}_{\{u,x_1,x_2,x,x_3\}}$ *satisfaisant les conditions :*

1) $u\, \mathcal{R}^{\mathbf{P}}_{(1,\varphi_1,d_1)}\, x_1\ ,\quad u\, \mathcal{R}^{\mathbf{P}}_{(1,\varphi_2,d_2)}\, x_2$ *et* $u\, \mathcal{R}^{\mathbf{P}}_{(1,\varphi_3,d_3)}\, x_3\ ,$ *avec* $\varphi_1, \varphi_2, \varphi_3 \geq 0$
2) $x \overset{\mathbf{P}}{=} x_1 \wedge_v x_2$

on a : $x \overset{\mathbf{P}}{\ominus} x_3 \;\Leftrightarrow\; (C_i) \;\Leftrightarrow\; x \overset{\mathbf{P}}{=} x_3.$

Preuve La preuve suit le même schéma que celui de la règle no. 4 précédente ; nous ne la présentons pas. □

Théorème 9 *(Règle no. 6)* $\forall\, \mathbf{P} \in \mathcal{P}_{\{u,x_1,x_2,x,x_3\}}$ *satisfaisant les conditions :*

1) $u\, \mathcal{R}^{\mathbf{P}}_{(1,\varphi_1,d_1)}\, x_1\ ,\quad u\, \mathcal{R}^{\mathbf{P}}_{(1,\varphi_2,d_2)}\, x_2$ *et* $u\, \mathcal{R}^{\mathbf{P}}_{(1,\varphi_3,d_3)}\, x_3\ ,$ *avec* $\varphi_1, \varphi_2, \varphi_3 \geq 0$
2) $x \overset{\mathbf{P}}{=} x_1 \setminus_v x_2$

on a : $x \overset{\mathbf{P}}{\ominus} x_3 \ \Leftrightarrow\ (C_d) \ \Leftrightarrow\ x \overset{\mathbf{P}}{=} x_3$.

Preuve La preuve suit le même schéma que celui des preuves pour les règles no. 4 et no. 5 précédentes ; nous ne la présentons pas. □

Nous concluons avec une règle de synchronisme déduite à partir de la propriété d'égalité d'expressions sur relations affines résumée par le Lemme 17, section 7.2.4.7. Pour que la règle soit applicable, il est nécessaire que les expressions soient sous la forme d'unions de relations affines. La Proposition 12, section 7.2.4.7, donne les conditions dans lesquelles une expression sur relations affines $\mathcal{R}^{\mathbf{P}}_{(1,\varphi_i,d_i)}$, avec $\varphi_i \geq 0$ et $d_i \geq 1$, se décompose en union finie de relations affines de même type.

Théorème 10 *(Règle no. 7) Soit* $A_1 = \{u, x_1, \ldots, x_p, y_1, \ldots, y_q, x, y\}$. $\forall\ P \in \mathcal{P}_{A_1}$ *satisfaisant les conditions :*

1) $u\ \mathcal{R}^{\mathbf{P}}_{(1,\varphi'_i,n)}\ x_i\ , \forall i = \overline{1,p}$ *et* $u\ \mathcal{R}^{\mathbf{P}}_{(1,\varphi''_j,n)}\ y_j\ , \forall j = \overline{1,q}$, *avec* $n \in I\!N^*$ *arbitraire*
2) $x \overset{\mathbf{P}}{=} (\vee_v)_{i=1}^{p} x_i$ *et* $y \overset{\mathbf{P}}{=} (\vee_v)_{j=1}^{q} y_j$

si on note $I_1 = \{\varphi'_1, \varphi'_2, \ldots, \varphi'_p\}$ *et* $I_2 = \{\varphi''_1, \varphi''_2, \ldots, \varphi''_q\}$, *alors :* $x \overset{\mathbf{P}}{\ominus} y \ \Leftrightarrow\ I_1 = I_2 \ \Leftrightarrow\ x \overset{\mathbf{P}}{=} y$.

Preuve On note $\mathcal{R}^{\mathbf{P}}_{t_1} = (\cup_r)_{\varphi \in I_1} \mathcal{R}^{\mathbf{P}}_{(1,\varphi,n)}$ et $\mathcal{R}^{\mathbf{P}}_{t_2} = (\cup_r)_{\varphi \in I_2} \mathcal{R}^{\mathbf{P}}_{(1,\varphi,n)}$. On a évidemment $(u, x) \in \mathcal{R}^{\mathbf{P}}_{t_1}$ et $(u, y) \in \mathcal{R}^{\mathbf{P}}_{t_2}$.

Supposons $x \overset{\mathbf{P}}{\ominus} y$; il existe en conséquence un flot $F_s \in \mathbf{P}$ dans lequel x et y sont identiques ($x \overset{F_s}{=} y$). Alors $(x, y), (y, x) \in \mathcal{R}^{F_s}_{(1,0,1)}$.

Comme $\mathcal{R}^{F_s}_{t_1} * \mathcal{R}^{F_s}_{(1,0,1)} = \mathcal{R}^{F_s}_{t_1}$ et $\mathcal{R}^{F_s}_{t_2} * \mathcal{R}^{F_s}_{(1,0,1)} = \mathcal{R}^{F_s}_{t_2}$ (Lemme 10, section 7.1.5.3), on obtient, par définition de la composition de relations affines, $(u, x) \in \mathcal{R}^{F_s}_{t_2}$ et $(u, y) \in \mathcal{R}^{F_s}_{t_1}$.

Nous considérons le cas général $\mathcal{R}^{F_s}_{t_1} = \{(u, x), (u, y)\}$ et $\mathcal{R}^{F_s}_{t_2} = \{(u, x), (u, y)\}$; il résulte $\mathcal{R}^{F_s}_{t_1} = \mathcal{R}^{F_s}_{t_2}$.

Avec Lemme 11, section 7.1.5.7, et Lemme 17, section 7.2.4.7, on obtient $I_1 = I_2$ et $\mathcal{R}^{\mathbf{P}}_{t_1} = \mathcal{R}^{\mathbf{P}}_{t_2}$. En conséquence, $p = q$ et on note $I_1 = I_2 = I = \{\varphi_1, \varphi_2, \ldots, \varphi_m\}$.

A partir de ces résultats il résulte $u\ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_i,n)}\ x_i$ et $u\ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_i,n)}\ y_i$, $\forall i = \overline{1,m}$, et il est facilement vérifiable que $x_i \overset{\mathbf{P}}{=} y_i$, $\forall\ i = \overline{1,m}$, ce qui implique $(\ (\vee_v)_{i=1}^{m} x_i\ ) \overset{\mathbf{P}}{=} (\ (\vee_v)_{i=1}^{m} y_i\ )$ et $x \overset{\mathbf{P}}{=} y$.

On a démontré $x \overset{\mathbf{P}}{\ominus} y \ \Rightarrow\ I_1 = I_2 \ \Rightarrow\ x \overset{\mathbf{P}}{=} y$. En utilisant $x \overset{\mathbf{P}}{=} y \ \Rightarrow\ x \overset{\mathbf{P}}{\ominus} y$ (évident), il résulte $x \overset{\mathbf{P}}{\ominus} y \ \Leftrightarrow\ I_1 = I_2 \ \Leftrightarrow\ x \overset{\mathbf{P}}{=} y$ (qed). □

7.3.2 Règles de synchronisabilité générales

Nous proposons et démontrons maintenant un ensemble de règles de synchronisabilité basées sur les propriétés des relations affines de paramètres généraux (n, φ, d), $n, d \in I\!N^*$ et $\varphi \in \mathbb{Z}$. Nous commençons par prouver l'identifiabilité de deux variables x_1 et x_2 qui sont en relation affine avec u_1 et u_2 entre lesquelles il n'existe aucune contrainte. Intuitivement, l'identifiabilité de x_1 et x_2 et immédiate puisqu'il n'existe aucune contrainte explicite ou implicite entre x_1 et x_2 suite à l'absence de contrainte entre u_1 et u_2. Nous étendons ensuite ce résultat au cas où x_1 et x_2 sont en relation affine avec un ensemble fini de variables $u_i, i = \overline{1,p}$, et respectivement $u'_j, j = \overline{1,q}$, et $u_i \neq u'_j$,

$\forall i = \overline{1,p}$ et $\forall j = \overline{1,q}$. Dans la suite, tous processus $\mathbf{P} \in \mathcal{P}_{A_1}$ sera considéré comme *le plus large (maximal)* satisfaisant les conditions indiquées : $\mathbf{P}$ sera l'ensemble de *tous* les flots vérifiant les relations précisées dans chaque cas.

Théorème 11 *(Règle no. 8) Soit* $\mathbf{P} \in \mathcal{P}_{\{u_1,u_2,x_1,x_2\}}$ *maximal satisfaisant les conditions :*

$$u_1 \ \mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)} \ x_1 \quad et \quad u_2 \ \mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)} \ x_2 \ , \ \ avec \ n_1, d_1, n_2, d_2 \in I\!N^* \ \ et \ \varphi_1, \varphi_2 \in \mathbb{Z}.$$

Alors $x_1 \overset{\mathbf{P}}{\ominus} x_2$.

Preuve On considère les processus suivants :

$\mathbf{Q} \in \mathcal{P}_{\{u_1,u_2,x_1,x_2\}}$ maximal tel que $u_1 \ \mathcal{R}^{\mathbf{Q}}_{(n_1,\varphi_1,d_1)} \ x_1$, $u_2 \ \mathcal{R}^{\mathbf{Q}}_{(n_2,\varphi_2,d_2)} \ x_2$ et $x_1 \overset{\mathbf{Q}}{=} x_2$.

$\mathbf{R} \in \mathcal{P}_{\{u_1,u_2,x_1\}}$ maximal tel que $u_1 \ \mathcal{R}^{\mathbf{R}}_{(n_1,\varphi_1,d_1)} \ x_1$ et $u_2 \ \mathcal{R}^{\mathbf{R}}_{(n_2,\varphi_2,d_2)} \ x_1$.

Comme $u_2 \ \mathcal{R}^{\mathbf{R}}_{(n_2,\varphi_2,d_2)} \ x_1$ est équivalent à $x_1 \ \mathcal{R}^{\mathbf{R}}_{(d_2,-\varphi_2,n_2)} \ u_2$ (conformément au Lemme 14, section 7.2.4.1), le processus $\mathbf{R}$ se réécrit en

$$\mathbf{R} = \{F \in \mathcal{F}_{\{u_1,x_1,u_2\}} \mid \ u_1 \ \mathcal{R}^{F}_{(n_1,\varphi_1,d_1)} \ x_1 \ \text{ et } \ x_1 \ \mathcal{R}^{F}_{(d_2,-\varphi_2,n_2)} \ u_2 \ \}$$

et il est non vide conformément à la remarque 1, section 7.1.5.2 : $\mathbf{R} \neq \emptyset$.

D'autre part, $x_1 \overset{\mathbf{Q}}{=} x_2$ est équivalent à $x_2 \ \mathcal{R}^{\mathbf{Q}}_{(1,0,1)} \ x_1$ (Lemme 15, section 7.2.4.3) et $u_2 \ \mathcal{R}^{\mathbf{Q}}_{(n_2,\varphi_2,d_2)} \ x_2$ et $x_2 \ \mathcal{R}^{\mathbf{Q}}_{(1,0,1)} \ x_1$ impliquent $u_2 \ \mathcal{R}^{\mathbf{Q}}_{(n_2,\varphi_2,d_2)} \ x_1$ ($\mathcal{R}^{\mathbf{Q}}_{(1,0,1)}$ est élément neutre par rapport la composition).

On a ainsi $\Pi_{\{u_1,x_1,u_2\}}(\mathbf{Q}) = \mathbf{R}$ et comme $\mathbf{R} \neq \emptyset$, alors $\mathbf{Q} \neq \emptyset$.

On vérifie facilement $\mathbf{P} \mid \mathbf{Q} = \mathbf{Q}$, ce qui implique $\Pi_{\{u_1,u_2,x_1,x_2\}}(\mathbf{Q}) \subseteq \mathbf{P}$, i.e. $\mathbf{Q} \subseteq \mathbf{P}$. En conséquence, il existe $\mathbf{Q} \subseteq \mathbf{P}$ non vide tel que $x_1 \overset{\mathbf{Q}}{=} x_2$, c'est-à-dire il existe $F \in \mathbf{P}$ tel que $x_1 \overset{F}{=} x_2$. Il résulte $x_1 \overset{\mathbf{P}}{\ominus} x_2$. □

Proposition 13 *(Règle no. 8') Soit* $A_1 = \{u_1, \ldots, u_p, u'_1, \ldots, u'_q, x_1, x_2\}$ *et* $\mathbf{P} \in \mathcal{P}_{A_1}$ *maximal satisfaisant les conditions :*

$$\begin{array}{ll} 1) & u_i \ \mathcal{R}^{\mathbf{P}}_{(n_i,\varphi_i,d_i)} \ x_1 \ , \ \forall i = \overline{1,p} \ \ et \ \forall n_i, d_i \in I\!N^*, \varphi_i \in \mathbb{Z} \\ 2) & u'_j \ \mathcal{R}^{\mathbf{P}}_{(n'_j,\varphi'_j,d'_j)} \ x_2 \ , \ \forall j = \overline{1,q} \ \ et \ \forall n'_j, d'_j \in I\!N^*, \varphi'_j \in \mathbb{Z} \end{array}$$

Alors $x_1 \overset{\mathbf{P}}{\ominus} x_2$.

Preuve Soit $i \in \{1, \ldots, p\}$, $j \in \{1, \ldots, q\}$ arbitraires. $\mathbf{P}_{ij} = \Pi_{\{u_i,u'_j,x_1,x_2\}}(\mathbf{P}) \in \mathcal{P}_{\{u_i,u'_j,x_1,x_2\}}$ est le processus maximal satisfaisant $u_i \ \mathcal{R}^{\mathbf{P}_{ij}}_{(n_i,\varphi_i,d_i)} \ x_1$ et $u'_j \ \mathcal{R}^{\mathbf{P}_{ij}}_{(n'_j,\varphi'_j,d'_j)} \ x_2$. Conformément au théorème précédent, on a $x_1 \overset{\mathbf{P}_{ij}}{\ominus} x_2$, d'où $x_1 \overset{\mathbf{P}}{\ominus} x_2$. □

Dans les situations exposées dans le Théorème 11 et la Proposition 13, les variables x_1 et x_2 étaient libres de toute contrainte, d'où leur identifiabilité. Nous étudions maintenant l'identifiabilité de x_1 et x_2 quand elles sont contraintes par des relations affines ou des compositions de relations affines.

En première instance, nous démontrons que si les variables x_1 et x_2 sont en relations affines de paramètres (n_1,φ_1,d_1) et (n_2,φ_2,d_2) avec la variable u, alors leur identifiabilité dépend directement des valeurs de (n_1,φ_1,d_1) et (n_2,φ_2,d_2). Plus spécifiquement, le Théorème 12 présente une situation dans laquelle x_1 et x_2 sont identifiables si et seulement si (n_1,φ_1,d_1) et (n_2,φ_2,d_2) sont équivalents $((n_1,\varphi_1,d_1) \sim (n_2,\varphi_2,d_2))$. Le test d'équivalence des deux triplets se réduit ensuite à un test d'égalité de leurs formes canoniques (voir les résultats présentés en section 7.1.3).

Théorème 12 *(Règle no. 9) Soit* $\mathbf{P} \in \mathcal{P}_{\{u,x_1,x_2\}}$ *maximal satisfaisant les conditions :*

$$u\ \mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)}\ x_1 \quad et \quad u\ \mathcal{R}^{\mathbf{P}}_{(n_2,\varphi_2,d_2)}\ x_2$$

Alors : $x_1 \overset{\mathbf{P}}{\ominus} x_2 \ \Leftrightarrow\ (n_1,\varphi_1,d_1) \sim (n_2,\varphi_2,d_2)$.

Preuve • Nous supposons $x_1 \overset{\mathbf{P}}{\ominus} x_2$ et démontrons $(n_1,\varphi_1,d_1) \sim (n_2,\varphi_2,d_2)$.

Si $x_1 \overset{\mathbf{P}}{\ominus} x_2$, il existe un flot $F_s \in \mathbf{P}$ tel que $x_1 \overset{F_s}{=} x_2$. Alors $(x_1,x_2),(x_2,x_1) \in \mathcal{R}^{F_s}_{(1,0,1)}$ et comme $\mathcal{R}^{F_s}_{(1,0,1)}$ est élément neutre par rapport à la composition, $u\ \mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)}\ x_1$ et $u\ \mathcal{R}^{\mathbf{P}}_{(n_2,\varphi_2,d_2)}\ x_2$ impliquent $u\ \mathcal{R}^{F_s}_{(n_1,\varphi_1,d_1)}\ x_2$ et $u\ \mathcal{R}^{F_s}_{(n_2,\varphi_2,d_2)}\ x_1$.

Nous considérons le cas général $\mathcal{R}^{F_s}_{(n_1,\varphi_1,d_1)} = \{(u,x_1),(u,x_2)\}$ et $\mathcal{R}^{F_s}_{(n_2,\varphi_2,d_2)} = \{(u,x_1),(u,x_2)\}$. L'identité des relations $\mathcal{R}^{F_s}_{(n_1,\varphi_1,d_1)}$ et $\mathcal{R}^{F_s}_{(n_2,\varphi_2,d_2)}$ implique $(n_1,\varphi_1,d_1) \sim (n_2,\varphi_2,d_2)$ (voir Définition 5, section 7.1.3).

• A partir de $(n_1,\varphi_1,d_1) \sim (n_2,\varphi_2,d_2)$ on veut démontrer $x_1 \overset{\mathbf{P}}{\ominus} x_2$.

Comme $(d_1,-\varphi_1,n_1) \sim (d_2,-\varphi_2,n_2)$ (évident, voir Proposition 1 et Définition 5, section 7.1.3) et $(n_1,\varphi_1,d_1) \sim (n_2,\varphi_2,d_2)$ (hypothèse), les conditions de l'applicabilité de la Proposition 8, section 7.2.4.2, sont satisfaites ; il existe en conséquence $\mathbf{P}_s \subseteq \mathbf{P}$ tel que

$$\mathcal{R}^{\mathbf{P}_s}_{(d_1,-\varphi_1,n_1)}\ *\ \mathcal{R}^{\mathbf{P}_s}_{(n_2,\varphi_2,d_2)} = \mathcal{R}^{\mathbf{P}_s}_{(d_1n_1,0,d_1n_1)}$$

Comme $(d_1n_1,0,d_1n_1) \sim (1,0,1)$ (voir Proposition 1 et Définition 5, section 7.1.3), on a $\mathcal{R}^{\mathbf{P}_s}_{(d_1n_1,0,d_1n_1)} = \mathcal{R}^{\mathbf{P}_s}_{(1,0,1)}$ (Lemme 13, section 7.2.3) et donc

$$\mathcal{R}^{\mathbf{P}_s}_{(d_1,-\varphi_1,n_1)}\ *\ \mathcal{R}^{\mathbf{P}_s}_{(n_2,\varphi_2,d_2)} = \mathcal{R}^{\mathbf{P}_s}_{(1,0,1)}$$

Avec $\mathcal{R}^{\mathbf{P}}_{(d_1,-\varphi_1,n_1)} = (\ \mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)}\ )^{-1}$ (Lemme 14, section 7.2.4) et en utilisant Définition 15, section 7.2.4, on a les implications suivantes :

$$\left.\begin{array}{r} (x_1,u) \in \mathcal{R}^{\mathbf{P}}_{(d_1,-\varphi_1,n_1)} \\ (u,x_2) \in \mathcal{R}^{\mathbf{P}}_{(n_2,\varphi_2,d_2)} \\ \mathbf{P}_s \subseteq \mathbf{P} \end{array}\right\} \begin{array}{l} \Rightarrow\quad (x_1,x_2) \in \mathcal{R}^{\mathbf{P}_s}_{(d_1,-\varphi_1,n_1)}\ *\ \mathcal{R}^{\mathbf{P}_s}_{(n_2,\varphi_2,d_2)} \\ \\ \Rightarrow\quad (x_1,x_2) \in \mathcal{R}^{\mathbf{P}_s}_{(1,0,1)} \\ \Rightarrow\quad x_1 \overset{\mathbf{P}_s}{=} x_2 \\ \Rightarrow\quad x_1 \overset{\mathbf{P}}{\ominus} x_2 \end{array}$$

□

Nous étendons le théorème précédent au cas de plusieurs variables $u_1, \ldots, u_p$ avec lesquelles x_1 et x_2 sont en relation affine.

Proposition 14 *(Règle no. 9') Soit $A_1 = \{u_1, \ldots, u_p, x_1, x_2\}$ et $\mathbf{P} \in \mathcal{P}_{A_1}$ maximal satisfaisant les conditions :*

$$\begin{array}{ll} 1) & u_i \ \mathcal{R}^{\mathbf{P}}_{(n_i^1,\varphi_i^1,d_i^1)} \ x_1 \ , \ \forall i = \overline{1,p} \ \textit{et} \ \forall n_i^1, d_i^1 \in I\!N^*, \varphi_i^1 \in \mathbb{Z} \\ 2) & u_i \ \mathcal{R}^{\mathbf{P}}_{(n_i^2,\varphi_i^2,d_i^2)} \ x_2 \ , \ \forall i = \overline{1,p} \ \textit{et} \ \forall n_i^2, d_i^2 \in I\!N^*, \varphi_i^2 \in \mathbb{Z} \end{array}$$

Alors $x_1 \overset{\mathbf{P}}{\ominus} x_2 \ \Leftrightarrow \ (n_i^1, \varphi_i^1, d_i^1) \sim (n_i^2, \varphi_i^2, d_i^2), \forall i = \overline{1,p}.$

Preuve • Nous supposons $x_1 \overset{\mathbf{P}}{\ominus} x_2$ et démontrons $(n_i^1, \varphi_i^1, d_i^1) \sim (n_i^2, \varphi_i^2, d_i^2), \forall i = \overline{1,p}$.

Si $x_1 \overset{\mathbf{P}}{\ominus} x_2$, il existe un flot $F_s \in \mathbf{P}$ tel que $x_1 \overset{F_s}{=} x_2$.Dans ce cas, $(x_1, x_2), (x_2, x_1) \in \mathcal{R}^{F_s}_{(1,0,1)}$ et comme $\mathcal{R}^{F_s}_{(1,0,1)}$ est élément neutre par rapport à la composition, $u_i \ \mathcal{R}^{\mathbf{P}}_{(n_i^1,\varphi_i^1,d_i^1)} \ x_1$ et $u_i \ \mathcal{R}^{\mathbf{P}}_{(n_i^2,\varphi_i^2,d_i^2)} \ x_2$ impliquent $u_i \ \mathcal{R}^{F_s}_{(n_i^1,\varphi_i^1,d_i^1)} \ x_2$ et $u_i \ \mathcal{R}^{F_s}_{(n_i^2,\varphi_i^2,d_i^2)} \ x_1$.

Dans le cas général où $\mathcal{R}^{F_s}_{(n_i^1,\varphi_i^1,d_i^1)} = \{(u_i, x_1), (u_i, x_2)\}$ et $\mathcal{R}^{F_s}_{(n_i^2,\varphi_i^2,d_i^2)} = \{(u_i, x_1), (u_i, x_2)\}$, $\forall i = \overline{1,p}$, on obtient $\mathcal{R}^{F_s}_{(n_i^1,\varphi_i^1,d_i^1)} = \mathcal{R}^{F_s}_{(n_i^2,\varphi_i^2,d_i^2)}$, $\forall i = \overline{1,p}$, et donc $(n_i^1, \varphi_i^1, d_i^1) \sim (n_i^2, \varphi_i^2, d_i^2), \forall i = \overline{1,p}$ (qed).

• Nous supposons maintenant $(n_i^1, \varphi_i^1, d_i^1) \sim (n_i^2, \varphi_i^2, d_i^2), \forall i = \overline{1,p}$ et démontrons $x_1 \overset{\mathbf{P}}{\ominus} x_2$.

Pour tout $i \in \{1, \ldots, p\}$, on note $\mathbf{P}_i = \Pi_{\{u_i, x_1, x_2\}}(\mathbf{P})$. Conformément au Théorème 1, section 6.5.4.5, on a $u_i \ \mathcal{R}^{\mathbf{P}_i}_{(n_i^1,\varphi_i^1,d_i^1)} \ x_1$ et $u_i \ \mathcal{R}^{\mathbf{P}_i}_{(n_i^2,\varphi_i^2,d_i^2)} \ x_2$. Avec un raisonnement similaire à celui du théorème précédent, on obtient qu'il existe $\mathbf{P}_s^i \subseteq \mathbf{P}_i$ tels que $x_1 \overset{\mathbf{P}_s^i}{=} x_2$, $\forall i = \overline{1,p}$.

On note $\mathbf{P}_s = \mathbf{P}_s^1 | \mathbf{P}_s^2 | \ldots \mathbf{P}_s^p$. Comme $\mathbf{P}_s^i \subseteq \mathbf{P}_i$, alors $\mathbf{P}_s \subseteq \mathbf{P}_1 | \mathbf{P}_2 | \ldots \mathbf{P}_p = \mathbf{P}$. Finalement, comme $x_1 \overset{\mathbf{P}_s^i}{=} x_2$, $\forall i = \overline{1,p}$, on a $x_1 \overset{\mathbf{P}_s}{=} x_2$ et $x_1 \overset{\mathbf{P}}{\ominus} x_2$ (qed). □

Les règles no. 9 et 9' précisent des conditions nécessaires et suffisantes pour l'identifiabilité de deux variables. Dès que l'on se place dans des conditions où on peut faire des opérations de composition et/ou d'intersection, union et différence sur des relations affines générales, on ne peut spécifier que des conditions suffisantes d'identifiabilité. Plus particulièrement, nous nous intéressons uniquement à l'opération de composition et démontrons deux règles d'identifiabilité basées sur les propriétés de l'opération de composition de relations générales et sur les résultats présentés aux sections 7.1.5.2 et 7.2.4.2.

Théorème 13 *(Règle no. 10) Soit $\mathbf{P} \in \mathcal{P}_{\{u, x_1, x_2, x_3\}}$ maximal satisfaisant les conditions :*

$$u \ \mathcal{R}^{\mathbf{P}}_{(n_1,\varphi_1,d_1)} \ x_1 \ , \ x_1 \ \mathcal{R}^{\mathbf{P}}_{(n_2,\varphi_2,d_2)} \ x_2 \ \ \textit{et} \ \ u \ \mathcal{R}^{\mathbf{P}}_{(n_3,\varphi_3,d_3)} \ x_3$$

S'il existe (n_1', φ_1', d_1'), (n_2', φ_2', d_2') tels que $d_1' = n_2'$ et satisfaisant en plus les conditions suivantes :

$$\begin{array}{l} (n_1', \varphi_1', d_1') \sim (n_1, \varphi_1, d_1) \\ (n_2', \varphi_2', d_2') \sim (n_2, \varphi_2, d_2) \\ (n_1' n_2', n_2' \varphi_1' + d_1' \varphi_2', d_1' d_2') \sim (n_3, \varphi_3, d_3) \end{array}$$

alors $x_2 \overset{\mathbf{P}}{\ominus} x_3$.

Preuve Soit $\mathbf{P}' = \Pi_{\{u,x_1,x_2\}}(\mathbf{P})$. On a $u\ \mathcal{R}^{\mathbf{P}'}_{(n_1,\varphi_1,d_1)}\ x_1$ et $x_1\ \mathcal{R}^{\mathbf{P}'}_{(n_2,\varphi_2,d_2)}\ x_2$ (conformément au Théorème 1, section 6.5.4.5).

Les triplets (n'_1,φ'_1,d'_1) et (n'_2,φ'_2,d'_2) et le processus $\mathbf{P}'$ satisfont les conditions de l'applicabilité de la Proposition 8, section 7.2.4.2 ; il résulte qu'il existe un processus $\mathbf{P}'_s \subseteq \mathbf{P}'$ tel que

$$\mathcal{R}^{\mathbf{P}'_s}_{(n_1,\varphi_1,d_1)} * \mathcal{R}^{\mathbf{P}'_s}_{(n_2,\varphi_2,d_2)} = \mathcal{R}^{\mathbf{P}'_s}_{(n'_1 n'_2, n'_2\varphi'_1 + d'_1\varphi'_2, d'_1 d'_2)}$$

Soit $\mathbf{P}'' = \Pi_{\{u,x_3\}}(\mathbf{P})$. Comme $\mathbf{P}'_s \subseteq \mathbf{P}'$, il résulte $\mathbf{P}'_s \mid \mathbf{P}'' \subseteq \mathbf{P}' \mid \mathbf{P}'' = \mathbf{P}$.

On note $\mathbf{P}_s = \mathbf{P}'_s \mid \mathbf{P}''$. Comme $\Pi_{\{u,x_1,x_2\}}(\mathbf{P}_s) = \mathbf{P}'_s$, conformément au Théorème 1, section 6.5.4.5, toute propriété vraie dans $\mathbf{P}'_s$ reste vraie dans $\mathbf{P}_s$; il résulte :

$$\mathcal{R}^{\mathbf{P}_s}_{(n_1,\varphi_1,d_1)} * \mathcal{R}^{\mathbf{P}_s}_{(n_2,\varphi_2,d_2)} = \mathcal{R}^{\mathbf{P}_s}_{(n'_1 n'_2, n'_2\varphi'_1 + d'_1\varphi'_2, d'_1 d'_2)}$$

On a mis en évidence un processus $\mathbf{P}_s \subseteq \mathbf{P}$ qui vérifie :

$$\mathcal{R}^{\mathbf{P}_s}_{(n_1,\varphi_1,d_1)} * \mathcal{R}^{\mathbf{P}_s}_{(n_2,\varphi_2,d_2)} = \mathcal{R}^{\mathbf{P}_s}_{(n'_1 n'_2, n'_2\varphi'_1 + d'_1\varphi'_2, d'_1 d'_2)}$$

On a aussi $u\ \mathcal{R}^{\mathbf{P}_s}_{(n_3,\varphi_3,d_3)}\ x_3$ (puisque $u\ \mathcal{R}^{\mathbf{P}}_{(n_3,\varphi_3,d_3)}\ x_3$, $\mathbf{P}'' = \Pi_{\{u,x_3\}}(\mathbf{P})$ et $\mathbf{P}_s = \mathbf{P}'_s|\mathbf{P}''$, $\mathbf{P}'_s \in \mathcal{P}_{\{u,x_1,x_2\}}$).

Soit $\mathbf{Q}_s = \Pi_{\{u,x_2,x_3\}}(\mathbf{P}_s)$. Les relations suivantes sont valides dans $\mathbf{Q}_s$:

$$\begin{array}{l} u\ \mathcal{R}^{\mathbf{Q}_s}_{(n'_1 n'_2, n'_2\varphi'_1 + d'_1\varphi'_2, d'_1 d'_2)}\ x_2 \\ u\ \mathcal{R}^{\mathbf{Q}_s}_{(n_3,\varphi_3,d_3)}\ x_3 \\ (n'_1 n'_2, n'_2\varphi'_1 + d'_1\varphi'_2, d'_1 d'_2) \sim (n_3,\varphi_3,d_3) \end{array}$$

Conformément à la règle de synchronisabilité no. 9, $x_2 \overset{\mathbf{Q}_s}{\ominus} x_3$ et avec $\mathbf{Q}_s = \Pi_{\{u,x_2,x_3\}}(\mathbf{P}_s)$, le Théorème 1, section 6.5.4.5 et $\mathbf{P}_s \subseteq \mathbf{P}$, il résulte $x_2 \overset{\mathbf{P}}{\ominus} x_3$. □

Nous étendons le résultat précédent au cas où la variable u est remplacée par un ensemble de variables.

Proposition 15 *(Règle no. 10') Soit $A_1 = \{u_1,\ldots,u_p,x_1,x_2,x_3\}$ et $\mathbf{P} \in \mathcal{P}_{A_1}$ maximal satisfaisant les conditions : $u_i\ \mathcal{R}^{\mathbf{P}}_{(n_i^1,\varphi_i^1,d_i^1)}\ x_1$, $x_1\ \mathcal{R}^{\mathbf{P}}_{(n_i^2,\varphi_i^2,d_i^2)}\ x_2$ et $u\ \mathcal{R}^{\mathbf{P}}_{(n_i^3,\varphi_i^3,d_i^3)}\ x_3$, $\forall i = \overline{1,p}$ et $\forall n_i^j, d_i^j \in \mathbb{N}^*, \varphi_i^j \in \mathbb{Z}, j = \overline{1,3}$.*

S'il existe (m_i^1,ξ_i^1,c_i^1) et (m_i^2,ξ_i^2,c_i^2) tels que $c_i^1 = m_i^2$, $\forall i = \overline{1,p}$, et satisfaisant aussi :

$$\begin{array}{l} (m_i^1,\xi_i^1,c_i^1) \sim (n_i^1,\varphi_i^1,d_i^1), \forall i = \overline{1,p} \\ (m_i^2,\xi_i^2,c_i^2) \sim (n_i^2,\varphi_i^2,d_i^2), \forall i = \overline{1,p} \\ (m_i^1 m_i^2, m_i^2\xi_i^1 + c_i^1\xi_i^2, c_i^1 c_i^2) \sim (n_i^3,\varphi_i^3,d_i^3), \forall i = \overline{1,p} \end{array}$$

alors $x_2 \overset{\mathbf{P}}{\ominus} x_3$.

Preuve Pour tout $i \in \{1,\ldots,p\}$, on note $\mathbf{P}_i = \Pi_{\{u_i,x_1,x_2,x_3\}}(\mathbf{P})$. Conformément au Théorème 1, section 6.5.4.5, on a $u_i\ \mathcal{R}^{\mathbf{P}_i}_{(n_i^1,\varphi_i^1,d_i^1)}\ x_1$, $x_1\ \mathcal{R}^{\mathbf{P}_i}_{(n_i^2,\varphi_i^2,d_i^2)}\ x_2$ et $u\ \mathcal{R}^{\mathbf{P}_i}_{(n_i^3,\varphi_i^3,d_i^3)}\ x_3$.

Dans chaque processus $\mathbf{P}_i$, un raisonnement similaire à celui du théorème précédent induit l'existence des processus $\mathbf{Q}_s^i$ et $\mathbf{P}_s^i$ satisfaisant : $\mathbf{Q}_s^i = \Pi_{\{u_i,x_2,x_3\}}(\mathbf{P}_s^i)$, $\mathbf{P}_s^i \subseteq \mathbf{P}_i$ et $\mathbf{P}_s^i \in \mathcal{P}_{\{u_i,x_1,x_2,x_3\}}$, et $x_2 \overset{\mathbf{Q}_s^i}{\ominus} x_3$,

$\forall i = \overline{1,p}$. En notant $\mathbf{Q}_{ss}^i \subseteq \mathbf{Q}_s^i$ dans lesquels $x_2 \overset{\mathbf{Q}_{ss}^i}{=} x_3$, $\forall i = \overline{1,p}$, on a :

$$\left.\begin{array}{l} \mathbf{Q}_{ss}^i \in \mathcal{P}_{\{u_i,x_2,x_3\}}, \forall i = \overline{1,p} \\ x_2 \overset{\mathbf{Q}_{ss}^i}{=} x_3, \forall i = \overline{1,p} \\ \mathbf{Q}_{ss} = \mathbf{Q}_{ss}^1 | \dots | \mathbf{Q}_{ss}^p \end{array}\right\} \Rightarrow \mathbf{Q}_{ss} \neq \emptyset \text{ et } x_2 \overset{\mathbf{Q}_{ss}}{=} x_3$$

et

$$\left.\begin{array}{l} \mathbf{Q}_{ss} = \mathbf{Q}_{ss}^1 | \dots | \mathbf{Q}_{ss}^p \\ \mathbf{Q}_{ss}^i \subseteq \mathbf{Q}_s^i \\ \mathbf{Q}_s = \mathbf{Q}_s^1 | \dots | \mathbf{Q}_s^p \end{array}\right\} \Rightarrow \mathbf{Q}_{ss} \subseteq \mathbf{Q}_s$$

En conclusion, $x_2 \overset{\mathbf{Q}_s}{\ominus} x_3$ et comme $\mathbf{Q}_s = \Pi_{\{u_1,\dots,u_p,x_2,x_3\}}(\mathbf{P}_s)$, avec $\mathbf{P}_s = \mathbf{P}_s^1 | \dots | \mathbf{P}_s^p$ et $\mathbf{P}_s \subseteq \mathbf{P}$, alors $x_2 \overset{\mathbf{P}}{\ominus} x_3$. □

D'après la définition de la relation d'équivalence $\sim$ (présentée en section 7.1.3), le problème du calcul de tous les triplets $(n_1'n_2', n_2'\varphi_1' + d_1'\varphi_2', d_1'd_2')$, satisfaisant les conditions du Théorème 13 pour un triplet (n_3, φ_3, d_3) donné, est NP-complet. La recherche d'un triplet $(n_1'n_2', n_2'\varphi_1' + d_1'\varphi_2', d_1'd_2')$ peut être réalisée par des heuristiques adaptées qui exploitent les propriétés de l'espace de recherche. On peut remarquer que $(n_1n_2, \varphi_1n_2, d_1n_2) \sim (n_1, \varphi_1, d_1)$,$(n_2d_1, \varphi_2d_1, d_2d_1) \sim (n_2, \varphi_2, d_2)$ et $d_1n_2 = n_2d_1$, donc $(n_1n_2, n_2\varphi_1 + d_1\varphi_2, d_1d_2)$ est un élément de l'espace de recherche qui peut constituer la solution initiale d'une heuristique de recherche. A partir de cette constatation, une étude des techniques de recherche induites par les propriétés des triplets (n_1', φ_1', d_1') et (n_2', φ_2', d_2') peut être envisagée.

7.3.3 Synchronisabilité dans le contexte affine – conclusions

Nous avons déduit un ensemble de règles de synchronisabilité qui forme le système de base pour les besoins de preuve formelle sur les synchronisations que nous avons exposé dans la première partie de notre document (codesign SIGNAL - ALPHA). Nous avons commencé par définir un système minimal de règles de synchronisme : toute autre règle utilisée dans le calcul d'horloges peut être déduite à partir de ce système.

Commentaire En effet, si la règle R est valide dans $\mathbf{P} \in \mathcal{P}_{A_1}$, R reste valide dans $\mathbf{Q} \in \mathcal{P}_{A_2}$ tel que $A_1 \subseteq A_2$ et $\mathbf{P} = \Pi_{A_1}(\mathbf{Q})$ (voir Théorème 1, section 6.5.4.5). A partir de ce résultat, on peut déduire, par exemple, la règle R3' à partir de R3 (règle no. 3, section 7.3.1). Nous énonçons R3' comme suit : *Dans tout processus* $\mathbf{P} \in \mathcal{P}_{\{u,x_1,x_2,y_1,y_2\}}$ *satisfaisant les conditions* $u \ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_1,d_1)} \ x_1$, $x_1 \ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_2,d_2)} \ x_2$ *et* $u \ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_1',d_1')} \ y_1$, $y_1 \ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_2',d_2')} \ y_2$, *on a* $x_2 \overset{\mathbf{P}}{\ominus} y_2 \Leftrightarrow \left\{\begin{array}{l} \varphi_1 + d_1\varphi_2 = \varphi_1' + d_1'\varphi_2' \\ d_1d_2 = d_1'd_2' \end{array}\right. \Leftrightarrow x_2 \overset{\mathbf{P}}{=} y_2$.

Pour la preuve, on considère $\mathbf{Q} \in \mathcal{P}_{\{u,x_1,x_2,y_1,y_2,z\}}$ tel que $u \ \mathcal{R}^{\mathbf{Q}}_{(1,\varphi,d)} \ z$ et $\mathbf{P} = \Pi_{\{u,x_1,y_1,x_2,y_2\}}(\mathbf{Q})$.

Soit $\mathbf{P}_x = \Pi_{\{u,x_1,x_2,z\}}(\mathbf{Q})$ et $\mathbf{P}_y = \Pi_{\{u,y_1,y_2,z\}}(\mathbf{Q})$. D'après R3, $x_2 \overset{\mathbf{P}_x}{\ominus} z \Leftrightarrow \left\{\begin{array}{l} \varphi_1 + d_1\varphi_2 = \varphi \\ d_1d_2 = d \end{array}\right. \Leftrightarrow$ $x_2 \overset{\mathbf{P}_x}{=} z$ et $y_2 \overset{\mathbf{P}_y}{\ominus} z \Leftrightarrow \left\{\begin{array}{l} \varphi_1' + d_1'\varphi_2' = \varphi \\ d_1'd_2' = d \end{array}\right. \Leftrightarrow y_2 \overset{\mathbf{P}_y}{=} z$. Avec les définitions de $\mathbf{P}_x$ et $\mathbf{P}_y$, on obtient les mêmes résultats dans $\mathbf{Q}$: $x_2 \overset{\mathbf{Q}}{\ominus} z \Leftrightarrow \left\{\begin{array}{l} \varphi_1 + d_1\varphi_2 = \varphi \\ d_1d_2 = d \end{array}\right. \Leftrightarrow x_2 \overset{\mathbf{Q}}{=} z$ et $y_2 \overset{\mathbf{Q}}{\ominus} z \Leftrightarrow \left\{\begin{array}{l} \varphi_1' + d_1'\varphi_2' = \varphi \\ d_1'd_2' = d \end{array}\right. \Leftrightarrow$ $y_2 \overset{\mathbf{Q}}{=} z$.

Dans le processus $\mathbf{Q}$, la relation identifiable est équivalente à la relation identique et elle est donc transitive. En conséquence, $x_2 \overset{\mathbf{Q}}{\ominus} z$ et $y_2 \overset{\mathbf{Q}}{\ominus} z$ impliquent $x_2 \overset{\mathbf{Q}}{\ominus} y_2$ qui est équivalent à $x_2 \overset{\mathbf{Q}}{=} y_2$. Comme $\mathbf{P} = \Pi_{\{u,x_1,y_1,x_2,y_2\}}(\mathbf{Q})$, on a : $x_2 \overset{\mathbf{P}}{\ominus} y_2 \Leftrightarrow \left\{ \begin{array}{l} \varphi_1 + d_1\varphi_2 = \varphi_1' + d_1'\varphi_2' \\ d_1 d_2 = d_1' d_2' \end{array} \right. \Leftrightarrow x_2 \overset{\mathbf{P}}{=} y_2.$ □

Le Théorème 1, section 6.5.4.5, et un raisonnement similaire permettent de déduire de nouvelles règles de synchronisabilité générales à partir de celles proposées en section 7.3.2, à condition que la propriété de transitivité soit vérifiée pour la relation d'identifiabilité. Dans ce cas on peut construire pour toute nouvelle règle un algorithme de déduction qui consiste dans l'application itérative d'un ensemble de règles au plus égal à l'ensemble minimal que l'on a proposé.

Pour tout processus $\mathbf{P} \in \mathcal{P}_{A_1}$ dans lequel on veut déduire la règle R, on peut construire $\mathbf{Q} \in \mathcal{P}_{A_2}$, $A_1 \subseteq A_2$ et $\mathbf{P} = \Pi_{A_1}(\mathbf{Q})$, dans lequel $\ominus$ est transitive. On peut ensuite choisir $\mathbf{Q}_i \in \mathcal{P}_{A_2^i}$, $A_2^i \subseteq A_2$ et $\mathbf{Q}_i = \Pi_{A_2^i}(\mathbf{Q})$ dans lesquels $R_i \in \{R8, R8', R9, R9', R10, R10'\}$ sont valides. Il résulte R_i valides dans $\mathbf{Q}$ et $\{R_i\}_{i\in I}$ permettra de déduire R itérativement. Finalement, R valide dans $\mathbf{Q}$ implique R valide dans $\mathbf{P}$.

En ce qui concerne les règles de synchronisabilité générales, notre système minimal traite uniquement les aspects relatifs à l'opération de composition. Nous nous sommes imposé cette limite qui s'est avérée réaliste pour le contexte d'application choisi (codesign). Ainsi, notre but n'était pas d'énumérer l'ensemble des règles de synchronisabilité que l'on peut déduire dans le contexte affine, mais de poser le problème de l'étude de la propriété de synchronisabilité en général et plus particulièrement dans le contexte des relations affines et de souligner les principaux aspects et soulever les principaux problèmes. Le système de règles que nous avons déduit peut être étendu avec de nouvelles règles obtenues à partir d'une étude complète sur les opérations d'union, intersection et différence de relations affines générales. Nous avons omis cette étude, mais avons présenté quelques idées qui peuvent constituer un point de départ en section 7.1.5.4.

7.4 Synchronisabilité & Calcul d'horloges

Nous avons introduit en section 6.4.8.3 la notion d'horloge : à chaque variable du langage SIGNAL peut être associé une variable d'horloge dont les instants de présence coïncident avec ceux de la variable et qui possède uniquement la valeur booléenne *true*. Les variables d'horloge décrivent complètement les synchronisations d'un programme SIGNAL. Le calcul de ces synchronisations, connu sous le nom de *calcul d'horloges* en SIGNAL, repose sur un ensemble de règles de synchronisabilité déduites directement à partir des propriétés des horloges sous-échantillonnages par conditions booléennes. Si nous considérons l'ensemble des horloges en relation affine, un calcul similaire peut être défini sur celles-ci en utilisant le système de règles de synchronisabilité que nous avons déduit précédemment. Ces règles, déduites pour des variables de même type (domaine de valeurs), sont directement applicables aux horloges qui sont des variables de même domaine de valeurs égal à $\{true\}$.

7.4.1 Synchronisabilité & Calcul d'horloges booléen

Comme il a été déjà présenté en section 2.2.6, le compilateur du langage SIGNAL associe une équation sur horloges à chaque processus élémentaire du langage. Par composition d'équations élémentaires on obtient un système qui décrit les synchronisations d'un programme SIGNAL ; leur cohérence est vérifiée au cours du calcul d'horloges booléen. Il s'agit en fait de la résolution du système d'équations sur horloges en utilisant les propriétés du treillis d'ordre $\preceq$ qui peut être défini sur l'espace des horloges. La méthode de résolution est une triangularisation qui se réduit principalement à la vérification de contraintes de synchronisation de type $h_1 = h_2$. L'idée du calcul d'horloges booléen est de tenter de réécrire les formules de définition de h_1 et h_2 dans une formule unique afin de vérifier leur équivalence. Les règles de réécriture sont déduites des axiomes du treillis booléen :

$$\begin{array}{l} h_1 \wedge h_2 = h_2 \wedge h_1 \\ h_1 \vee (h_2 \wedge h_3) = (h_1 \vee h_2) \wedge (h_1 \vee h_3) \\ (h_1 \wedge h_2) \vee h_1 = h_1 \\ \ldots \end{array} \tag{7.8}$$

et des équations de partition associées à chaque condition booléenne C :

$$\begin{array}{l} [C] \vee [\neg C] = \hat{C} \\ [C] \wedge [\neg C] = \emptyset \end{array} \tag{7.9}$$

Si la tentative de réécriture de h_1 et h_2 aboutit, l'égalité de h_1 et h_2 est prouvée; sinon, on peut seulement conclure que les méthodes de vérification implémentées actuellement dans le compilateur ne peuvent rien décider en ce qui concerne l'égalité de h_1 et h_2. En effet, si la contrainte de synchronisation $h_1 = h_2$ ne peut pas être vérifiée, cela veut dire qu'on ne dispose pas d'une règle de synchronisabilité concernant h_1 et h_2. Si on note $\mathbf{P}$ le processus associé au programme Signal en cours de compilation, $h_1 = h_2$ est équivalent à $h_1 \stackrel{\mathbf{P}}{=} h_2$, c'est-à-dire h_1 et h_2 sont synchrones dans $\mathbf{P}$. Soit $\mathbf{P}_e$ le processus $\mathbf{P}$ dans lequel on a enlevé la contrainte de synchronisation de h_1 et h_2 ; si $h_1 \stackrel{\mathbf{P}_e}{\odot} h_2$, c'est-à-dire il existe $\{F_i\}_{i \in I} \subseteq \mathbf{P}_e$ tels que $h_1 \stackrel{F_i}{\hat{=}} h_2$, alors on défini $\mathbf{P} = \{F_i\}_{i \in I} \neq \emptyset$ et on a $h_1 \stackrel{\mathbf{P}}{\hat{=}} h_2$ qui est équivalent à $h_1 \stackrel{\mathbf{P}}{=} h_2$ (deux horloges sont synchrones ssi elles sont identiques). Il est donc nécessaire de disposer d'un ensemble de règles qui permettraient de décider sur la synchronisabilité de h_1 et h_2. En ce qui concerne l'actuel calcul d'horloges booléen, les règles de synchronisabilité sont déduites directement à partir des axiomes du treillis d'ordre $\preceq$ et représentent des règles de synchronisme :

$$\begin{cases} h_1 \wedge h_2 \stackrel{\mathbf{P}}{\odot} h_2 \wedge h_1 & (\text{équivalent à } h_1 \wedge h_2 \stackrel{\mathbf{P}}{\hat{=}} h_2 \wedge h_1) \\ h_1 \vee (h_2 \wedge h_3) \stackrel{\mathbf{P}}{\odot} (h_1 \vee h_2) \wedge (h_1 \vee h_3) & (\text{équivalent à } h_1 \vee (h_2 \wedge h_3) \stackrel{\mathbf{P}}{\hat{=}} (h_1 \vee h_2) \wedge (h_1 \vee h_3)) \\ (h_1 \wedge h_2) \vee h_1 \stackrel{\mathbf{P}}{\odot} h_1 & (\text{équivalent à } (h_1 \wedge h_2) \vee h_1 \stackrel{\mathbf{P}}{\hat{=}} h_1) \\ \ldots & \end{cases}$$

$$\begin{cases} [C] \vee [\neg C] \stackrel{\mathbf{P}}{\odot} \hat{C} & (\text{équivalent à } [C] \vee [\neg C] \stackrel{\mathbf{P}}{\hat{=}} \hat{C}) \\ & \\ [C] \wedge [\neg C] \stackrel{\mathbf{P}}{\odot} \emptyset & (\text{équivalent à } [C] \wedge [\neg C] \stackrel{\mathbf{P}}{\hat{=}} \emptyset) \end{cases}$$

A partir de ces règles on ne peut pas déduire $h_1 \stackrel{\mathbf{P}}{\odot} h_2$ dans un processus dans lequel h_1 et h_2 sont définies respectivement par $h_1 = h_1' \vee h_2'$ et $h_2 = h_1' \wedge h_2'$ (h_1', h_2' horloges libres). Et pourtant pour tout flot $F \in \mathbf{P}$ tel que $h_1' \stackrel{F}{=} h_2'$ on a aussi $h_1' \vee h_2' \stackrel{F}{=} h_1' \wedge h_2'$, donc $h_1' \vee h_2' \stackrel{\mathbf{P}}{\odot} h_1' \wedge h_2'$.

A partir de cet exemple simple, on s'aperçoit qu'il est possible d'élargir le système de règles de synchronisabilité considérées comme mises à la disposition du calcul d'horloges; il est nécessaire pour cela d'étudier l'ensemble de contraintes (relations) existantes entre horloges dans un processus **P** donné. Quand ces contraintes sont des relations affines ou des expressions sur des relations de ce type, nous avons déduit un nouveau système de règles de synchronisabilité qui seront à la base du calcul affine d'horloges.

7.4.2 Synchronisabilité & Calcul affine d'horloges

Les règles de synchronisabilité déduites dans le contexte des relations affines peuvent être utilisées pour la mise en place d'un calcul affine d'horloges d'après le modèle du calcul booléen existant. Premièrement, trois opérateurs nouveaux vont permettre d'exprimer des relations affines entre horloges dans le langage SIGNAL. Toute relation affine entre h_1 et h_2 sera ensuite construite en définissant h_1 et h_2 comme des *sous-échantillonnages affines* de phase φ positive et période d strictement positive sur une même horloge h. Cette implémentation d'une relation affine permettra de définir $\preceq_{aff}$ d'après le modèle de $\preceq$ et d'utiliser les règles de synchronisabilité que nous avons proposé pour la résolution du système d'équations affines déduites à partir des trois opérateurs affines que nous proposons. La méthode de résolution suivra en principe le schéma de la triangularisation existante et pourra être intégrée dans le calcul d'horloges booléen.

7.4.2.1 Processus affines en SIGNAL

Afin d'exprimer des relations affines en SIGNAL, nous avons introduit trois opérateurs nouveaux (processus affines). L'élément syntaxique

$$\texttt{y := sample\{ } \varphi, n \texttt{ \}(} x \texttt{)}$$

avec $x, y \in A \cup C$ et $\varphi \geq 0$, définit du point de vue sémantique le processus $\mathbf{P}_s \in \mathcal{P}_{\{x,y\}}$,

$$\mathbf{P}_s =_{\Delta} \{F \in \mathcal{F}_{\{x,y\}} \ / \ x \ \mathcal{R}^F_{(1,\varphi,n)} \ y\}$$

Dans $\mathbf{P}_s$, y est défini comme un sous-échantillonnage affine de phase φ et période n sur x. Pour $\varphi = 3$ et $n = 4$, le processus $\mathbf{P}_s$ a été illustré sur la Figure 7.15.

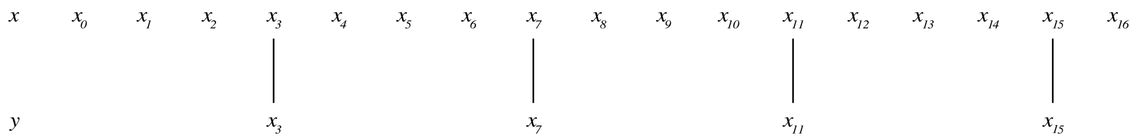

FIG. 7.15 – *Exemple :* y `:=` `sample`{ 3,4 }(x).

L'opérateur

$$\texttt{clk\_affine\{ } n, \varphi, d \texttt{ \}(} x, y \texttt{)}$$

avec $x, y \in A \cup C$, définit du point de vue sémantique le processus $\mathbf{P}_{ha} \in \mathcal{P}_{\{x,y\}}$,

$$\mathbf{P}_{ha} =_{\Delta} \{F \in \mathcal{F}_{\{x,y\}} \ / \ h(x) \ \mathcal{R}^F_{(n,\varphi,d)} h(y)\}$$

Le processus $\mathbf{P}_{ha}$ représente ainsi l'ensemble des flots dans lesquels les *horloges* de x et y sont en relation affine de paramètres (n, φ, d). Nous avons illustré sur la Figure 7.16 la représentation[5] en SIGNAL du processus affine `clk_affine` pour le cas $n = 5$, $\varphi = 4$ et $d = 7$.

5. I représente l'horloge de la variable u conformément à l'implémentation d'une relation affine introduite en Proposition 16, section 7.4.2.2.

$H = H_x$	i_0					i_5					i_{10}					i_{15}				
I	i_0	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}	i_{11}	i_{12}	i_{13}	i_{14}	i_{15}	i_{16}	i_{17}	i_{18}	i_{19}
$K = H_y$					i_4							i_{11}							i_{18}	

FIG. 7.16 – *Exemple :* `clk_affine`{ 5,4,7 }(x,y).

Le constructeur syntaxique

$$y \texttt{ := unsample}\{\, n,\varphi \,\}(\, x,z \,)$$

avec $n > 0$ et $\varphi \in \mathbb{Z}$, définit du point de vue sémantique le processus $\mathbf{P}_{un}$ dans lequel y est défini comme un sur-échantillonnage à partir de x ; le signal z sert à fixer les valeurs de y quand x n'est pas présent (voir Figure 7.17 pour un exemple).

x	x_0		x_1			x_2			x_3			x_4			x_5
z		z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8	z_9	z_{10}	z_{11}	z_{12}	z_{13}
y		z_0	$\boxed{x_1}$	z_2	z_3	$\boxed{x_2}$	z_5	z_6	$\boxed{x_3}$	z_8	z_9	$\boxed{x_4}$	z_{11}	z_{12}	$\boxed{x_5}$

FIG. 7.17 – *Exemple :* y `:= unsample`{ 3,2 }(x,z).

7.4.2.2 L'ordre $\preceq_{aff}$

Le but du calcul d'horloges est non seulement de vérifier les contraintes de synchronisation d'un programme SIGNAL, mais de construire une horloge plus grande que toutes les horloges d'un programme et à partir de laquelle toutes les autres horloges peuvent être calculées. Le même principe s'applique dans le contexte affine. L'horloge la plus grande d'un programme sera synthétisée à partir des horloges d'exécution de chaque relation affine. En effet, dans tout processus $\mathbf{P} \in \mathcal{P}_{\{h_1,h_2\}}$ tel que $h_1 \; \mathcal{R}^{F}_{(n,\varphi,d)} \; h_2$, on met en évidence une horloge h à partir de laquelle h_1 et h_2 peuvent être construites comme des sous-échantillonnages affines ; l'horloge h sera supérieure à h_1 et h_2 et représentera l'horloge d'exécution de la relation affine. La proposition suivante fixe formellement cette propriété.

Proposition 16 *Soit le processus* $\mathbf{P}_1 \in \mathcal{P}_{\{x,y\}}$ *tel que* $x \; \mathcal{R}^{\mathbf{P}_1}_{(n,\varphi,d)} \; y$ *. Il existe* $u \in A \setminus \{x,y\}$ *et* $\mathbf{P} \in \mathcal{P}_{\{x,y,u\}}$ *avec les propriétés* $u \; \mathcal{R}^{\mathbf{P}}_{(1,max(0,-\varphi),n)} \; x$ *,* $u \; \mathcal{R}^{\mathbf{P}}_{(1,max(0,\varphi),d)} \; y$ *et* $\mathbf{P}_1 = \Pi_{\{x,y\}}(\mathbf{P})$ *.*

Preuve Soit[6] $u \in A \setminus \{x,y\}$.

6. L'ensemble A est supposé suffisamment grand pour garantir l'existence de la variable u.

Pour tout flot $F_1 \in \mathbf{P}_1$, soit $\mathcal{F}_1^{F_1}$ l'ensemble des flots $F_{F_1} \in \mathcal{F}_{\{u\}}$ pour lesquels les valeurs du signal u satisfont les relations suivantes :

$$u_{(F_{F_1})}(nt + max(0, -\varphi)) = x_{(F_1)}(t), \qquad \forall t \in \mathbb{N}$$
$$u_{(F_{F_1})}(dt + max(0, \varphi)) = y_{(F_1)}(t), \qquad \forall t \in \mathbb{N}$$

Le processus $\mathbf{P} \in \mathcal{P}_{\{x,y,u\}}$ sera l'ensemble des flots F obtenus comme suit : pour tout flot $F_1 \in \mathbf{P}_1$ et tout flot $F_{F_1} \in \mathcal{F}_1^{F_1}$ associé à F_1, on construit un flot $F \in \mathcal{F}_{\{x,y,u\}}$ tel que

$$F = (\ g \uparrow F_{F_1}\ ) \cdot (\ (g \circ f_1) \uparrow \Pi_{\{x\}}(F_1)\ ) \cdot (\ (g \circ f_2) \uparrow \Pi_{\{y\}}(F_1)\ )$$

avec $g = \lambda\ t.(t)$, $f_1(t) = \lambda\ t.(nt + max(0, -\varphi))$, $f_2(t) = \lambda\ t.(dt + max(0, \varphi))$. On note $\mathcal{F}_2^{F_1}$ l'ensemble de flots F construits comme précédemment pour chaque flot $F_1 \in \mathbf{P}_1$.

Pour tout flot $F \in \mathcal{F}_2^{F_1}$, les relations suivantes sont vérifiées : $\Pi_{\{u\}}(F) = \Pi_{\{u\}}(F_{F_1}) = F_{F_1}$ et $\Pi_{\{x\}}(F) = \Pi_{\{x\}}(F_1)$, $\Pi_{\{y\}}(F) = \Pi_{\{y\}}(F_1)$.

On construit $\mathbf{P}$ comme l'ensemble de tous les flots $F \in \mathcal{F}_2^{F_1}$, pour chaque flot $F_1 \in \mathbf{P}_1$. On obtient pour $\mathbf{P}$ la définition suivante :

$$\mathbf{P} \quad = \quad \{F \in \mathcal{F}_{\{x,y,u\}} \mid u\ \mathcal{R}^F_{(1,max(0,-\varphi),n)}\ x \ \text{ et } u\ \mathcal{R}^F_{(1,max(0,\varphi),d)}\ y\}$$

D'après la construction de $\mathbf{P}$, pour tout flot $F_1 \in \mathbf{P}_1$, $F_1 = \Pi_{\{x,y\}}(F)$, $\forall\ F \in \mathcal{F}_2^{F_1}$. Comme $\mathbf{P}$ représente l'union des ensembles $\mathcal{F}_2^{F_1}$, $\forall\ F_1 \in \mathbf{P}_1$, il résulte $\mathbf{P}_1 = \Pi_{\{x,y\}}(\mathbf{P})$. □

Le processus $\mathbf{P}$ représente une implémentation de $\mathbf{P}_1$ quand x et y sont des variables arbitraires de même domaine de valeurs. Dans le cas où x et y représentent des variables d'horloges, remplacer $\mathbf{P}_1$ par son implémentation $\mathbf{P}$ revient à construire l'horloge u d'exécution de $\mathbf{P}_1$. Comme $\mathbf{P}_1 = \Pi_{\{x,y\}}(\mathbf{P})$, toute propriété concernant une relation affine dans $\mathbf{P}_1$ reste vraie dans l'implémentation $\mathbf{P}$.

Opérateurs affines	Description	Équations sur horloges
y `:= sample{` φ,d `}(` x `)`	Sous-échantillonnage par fonction affine $\lambda t(dt + \varphi)$	$\hat{y} = [\hat{x}]_{(\varphi,d)}$
`clk_affine{` n,φ,d `}(` x,y `)`	Relation affine entre les horloges de x et y	$\hat{x} = [\hat{i}]_{(max(0,-\varphi),n)}$ $\hat{y} = [\hat{i}]_{(max(0,\varphi),d)}$
y `:= unsample{` n,φ `}(` x,z `)`	Sur-échantillonnage affine en utilisant les signaux x et z	$\hat{y} = [\hat{i}]_{(max(0,\varphi),1)}$ $\hat{y} = \hat{z}$ $\hat{x} = [\hat{i}]_{(max(0,-\varphi),n)}$

TAB. 7.1 – **Opérateurs affines en** SIGNAL .

L'horloge u est supérieure à x et y par rapport à l'ordre $\preceq^{\mathbf{P}}_{aff}$ qui peut être défini sur l'espace des horloges d'un processus $\mathbf{P}$ par l'expression suivante :

$$\preceq^{\mathbf{P}}_{aff} = \{(x, y) \in A_1 \times A_1 \mid \exists\ \varphi_i \geq 0, d_i > 1,\ \mathcal{R}^{\mathbf{P}}_t = \mathcal{EXP}(\ldots, \mathcal{R}^{\mathbf{P}}_{(1,\varphi_i,d_i)}, \ldots) \text{ et } \ y\ \mathcal{R}^{\mathbf{P}}_t\ x\ \}$$

Dans le cas où $\mathcal{R}_t^{\mathbf{P}} = \mathcal{R}_{(1,\varphi,d)}^{\mathbf{P}}$, avec $\varphi \geq 0$ et $d \geq 1$, la relation $y\ \mathcal{R}_{(1,\varphi,d)}^{\mathbf{P}}\ x$ peut être notée $x \overset{\mathbf{P}}{=} [y]_{(\varphi,d)}$ afin d'exprimer le fait que x est un sous-échantillonnage de phase φ positive et période d strictement positive sur y ($x_{(F)}(s) = y_{(F)}(ds + \varphi)$, $\forall\ s \in \mathbb{N}$ et $\forall F \in \mathbf{P}$, conformément à la définition d'une relation affine $\mathcal{R}_{(1,\varphi,d)}^{\mathbf{P}}$). Avec cette notation, on dit que $x \preceq_{aff}^{\mathbf{P}} y$ si et seulement si il existe une expression Exp construite avec les opérateurs $\{\vee_v, \wedge_v, \backslash_v\}$ telle que $x \overset{\mathbf{P}}{=} Exp(\ldots [y]_{(\varphi_i,d_i)} \ldots)$.

L'ordre partiel $\preceq_{aff}^{\mathbf{P}}$ défini une structure de treillis sur l'espace des horloges en relation affine ; il sera noté $\preceq_{aff}$ quand le processus $\mathbf{P}$ est implicite. Les règles de synchronisabilité déduites dans le même espace constituent des propriétés du treillis d'ordre $\preceq_{aff}$ qui représenteront le support de la méthode de résolution des équations sur horloges *affines*[7] associées à chaque processus affine dans le tableau 7.1.

Conformément à l'implémentation que nous avons choisi pour une relation affine (Proposition 16), les processus `clk_affine` et `unsample` peuvent être réécrits en

```
                                   (|  H := sample{ max(0,-φ),n }( I )
                                    |  K := sample{ max(0,φ),d }( I )
clk_affine{ n,φ,d }( x,y )   ⇔      |  H ^= x
  ( n,d > 0  and  φ ∈ ℤ )           |  K ^= y
                                    |) /  H,K,I
```

```
                                     (|  clk_affine{ n,φ,1 }( x,y )
y := unsample{ n,φ }( x,z )   ⇔       |  y := ( x when ^y ) default z
     ( n > 0  and  φ ∈ ℤ )            |  y ^= z  |)
```

d'où les systèmes d'équations qui leur sont associés dans le tableau 7.1.

7.4.2.3 Prélude au calcul affine d'horloges

Avec les règles de synchronisabilité disponibles, il est possible de mettre en place un calcul d'horloges affines d'après le modèle du calcul booléen existant. L'idée du calcul est de synthétiser l'horloge la plus rapide par rapport à l'ordre $\preceq_{aff}$ – horloge *maîtresse* – qui cadencera l'exécution du programme et à partir de laquelle toutes les autres horloges peuvent être calculées. Toute horloge sera définie en fonction de l'horloge maîtresse ou en fonction d'autres horloges calculées précédemment à partir de celle-ci. Un système d'équations triangulaire sera progressivement construit à partir du système initial. Au cours de la triangularisation, il est possible d'obtenir plusieurs définitions pour la même horloge. Dans ce cas, la synchronisabilité de toutes les expressions de définition doit être vérifiée, et ceci en utilisant le système de règles de synchronisabilité dont on dispose.

Nous avons divisé notre système en deux parties. Premièrement, il y a les règles de synchronisme qui sont des lois d'équivalence entre expressions construites avec des sous-échantillonnages affines de phase φ positive et période d strictement positive sur une même horloge de base. Nous reprenons ici quelques unes et renvoyons le lecteur au calcul réalisé en section 7.3.1 pour une vue complète

7. Horloges sous-échantillonnages affines.

de l'ensemble de règles de synchronisme que nous avons proposé :

$$[h_u]_{(\varphi,d)} \overset{\mathbf{P}}{\odot} [h_u]_{(\varphi',d')}, \forall\varphi = \varphi', d = d' \qquad \text{(équivalent à } [h_u]_{(\varphi,d)} \overset{\mathbf{P}}{\hat{=}} [h_u]_{(\varphi',d')}, \forall\varphi = \varphi', d = d')$$

$$[h_u]_{(0,1)} \overset{\mathbf{P}}{\odot} h_u \qquad \text{(équivalent à } [h_u]_{(0,1)} \overset{\mathbf{P}}{\hat{=}} h_u)$$

$$[[h_u]_{(\varphi_1,d_1)}]_{(\varphi_2,d_2)} \overset{\mathbf{P}}{\odot} [h_u]_{(\varphi_1+d_1\varphi_2,d_1d_2)} \qquad \text{(équivalent à } [[h_u]_{(\varphi_1,d_1)}]_{(\varphi_2,d_2)} \overset{\mathbf{P}}{\hat{=}} [h_u]_{(\varphi_1+d_1\varphi_2,d_1d_2)})$$

...

Nous avons proposé un système minimal de règles de synchronisme qui s'avérera suffisant pour les besoins du calcul d'horloges affines que nous avons implémenté. Nous détaillons les principaux aspects et problèmes de ce calcul dans la partie dédié à l'implémentation ; dans ce contexte, les résultats concernant les règles de synchronisme auront une applicabilité immédiate. A celles-ci s'ajouteront les règles de synchronisabilité générales que nous avons proposées et démontrées en section 7.3.2. Or, toutes les règles de synchronisabilité que nous avons déduites pour des processus $\mathbf{P}$ restreints à un ensemble minimal de variables A_1 sont valides dans tout processus $\mathbf{Q} \in \mathcal{P}_{A_2}$ tel que $A_1 \subseteq A_2$ et $\Pi_{A_1}(\mathbf{Q}) \subseteq \mathbf{P}$. Cette propriété garantit la validité de nos règles de synchronisabilité dans la mise en œuvre du calcul affine d'horloges qu'on propose dans la suite. Plus précisément, nous allons utiliser ces règles de synchronisabilité dans des processus $\mathbf{Q} \in \mathcal{P}_{A_2}$ pour lesquels il existe $A_1 \subseteq A_2$ et $\mathbf{P} \in \mathcal{P}_{A_1}$ tels que les théorèmes et propositions de la section 7.3 soient valides en $\mathbf{P}$ et $\Pi_{A_1}(\mathbf{Q}) \subseteq \mathbf{P}$. Dans ce cas, d'après le Lemme 1 et le Théorème 1, section 6.5.4.5, les théorèmes et propositions de la section 7.3 sont valides aussi en $\mathbf{Q}$.

7.5 Conclusion

Le contexte de départ de notre travail de thèse – codesign avec SIGNAL et ALPHA – a généré des *transformations affines* sur des variables d'horloges et la nécessité de fournir un calcul formel sur les synchronisations des horloges transformées. Dans ce chapitre nous avons proposé un tel calcul dans le modèle sémantique de traces du langage SIGNAL et en utilisant un ensemble de propriétés des *relations affines* qui sont définies entre les horloges opérandes d'une transformation affine. Nous avons proposé un système de *règles de synchronisabilité* d'horloges sur lequel sera basé le calcul *affine* d'horloges que nous introduisons dans la troisième partie de notre document.

Au cours de notre calcul formel de synchronisabilité, nous avons manipulé des fonctions affines $nt + \varphi$, $t \in \mathbb{N}$, et avons trouvé un ensemble de propriétés qui dépendent essentiellement des valeurs des entiers positifs n et φ. Or, les manipulations de programmes dans l'environnement de synthèse du langage ALPHA sont basées sur un calcul formel de fonctions affines unimodulaires et de domaines polyédraux. Récemment, ALPHA a été étendu afin de permettre la description d'une classe d'équations de récurrence dont les variables sont définies sur un ensemble plus large de domaines – des unions finies de Z-polyèdres définis comme l'intersection entre un polyèdre et un treillis (des domaines à "trous réguliers") [QRR96b] – et les fonctions affines qui leurs sont appliquées sont des fonctions rationnelles inversibles (pseudo-unimodulaires).

Nous avons constaté des similitudes entre notre calcul sur relations affines et le calcul sur Z-polyèdres[8] (l'union de Z-polyèdres ne produit pas toujours un Z-polyèdre, il peuvent exister plusieurs décompositions d'un Z-polyèdre en union de Z-polyèdres) et nous pensons que notre travail peut constituer le point de départ vers l'intégration en SIGNAL des techniques de manipulation

8. Dans le cas unidimensionnel un Z-polyèdre est défini par une fonction affine $nt + \varphi$, avec $t \in A \subseteq \mathbb{N}$ et $n \geq 1$, $\varphi \geq 0$.

de structures de données régulières – polyèdres convexes et Z-polyèdres – comme ils apparaissent dans les méthodes d'implémentation optimale de programmes [DPR94] [QRR96a] [Fea96]. Nous présentons brièvement quelques arguments en faveur de cette idée. D'abord, nous avons remplacé une éventuelle étude de synchronisabilité sur des variables d'horloges par une étude un peu plus générale sur l'identifiabilité de variables de même domaine de valeurs et d'horloges différentes indépendantes des valeurs. Dans le cas des variables de domaines de valeurs différents, l'identifiabilité doit prendre en considération ces domaines. Un calcul de domaines convexes uni-dimensionnels peut être envisagé d'après le modèle existant en ALPHA ; il impliquera essentiellement des opérations sur polyèdres – intersection, union, test d'égalité – et sera extensible aux variables multi-dimensionnelles pour lesquelles la bibliothèque polyédrale de ALPHA est disponible.

Ensuite, le calcul sur relations affines peut constituer un bon point de départ pour l'étude des horloges *vectorielles* en SIGNAL. Actuellement, toute structure multi-dimensionnelle est supposée synchrone, c'est-à-dire tous les éléments de la structure doivent avoir la même horloge. Or, dans certains cas il peut être intéressant d'avoir des tableaux d'éléments asynchrones. Dans ce cas, l'étude que nous avons réalisée devrait être étendue au cas multi-dimensionnel et intégrée avec les résultats disponibles dans le monde des Z-polyèdres.

Troisième partie

Implémentation des transformations affines d'horloges en Signal

Chapitre 8

Introduction

Cette partie est consacrée essentiellement à la description du prototype de mise en œuvre, dans le compilateur de SIGNAL, de nos résultats théoriques sur les relations affines entre horloges. Le compilateur du langage SIGNAL est un outil de preuve qui effectue des calculs formels sur les synchronisations des signaux et les dépendances de données. Une étape importante du processus de compilation est constitué par le *calcul d'horloges* dont le rôle est de vérifier la cohérence des contraintes de synchronisation spécifiées dans un programme en utilisant un système de règles de synchronisabilité basées sur les propriétés de l'espace des horloges.

Dans le contexte de codesign que nous avons étudié, nous étions confrontés aux problèmes induits par le raffinement de spécifications correspondant à l'exploration de diverses architectures d'un système. Le calcul des synchronisations sur des expressions construites avec des horloges obtenues suite à des transformations affines est un de ces problèmes. Pour la mise en œuvre de ce calcul, nous avons proposé un nouveau système de règles de synchronisabilité basées sur les propriétés des relations affines entre horloges générées par les transformations affines. Ces règles, ajoutées à celles déjà existantes dans le calcul d'horloges de SIGNAL, permettent d'augmenter le nombre de situations dans lesquelles on peut décider si deux horloges peuvent être *synchronisées* (c'est-à-dire qu'elles sont synchronisables). La puissance de spécification et de validation de SIGNAL est ainsi augmentée par un calcul formel plus puissant à l'intérieur de son compilateur.

Dans cette partie, nous abordons les problèmes de la mise en œuvre du calcul *affine* d'horloges – le calcul formel des nouvelles synchronisations – et de son intégration avec le calcul booléen existant, selon le plan suivant :

Nous commençons par un chapitre concernant l'environnement de programmation du langage SIGNAL. Nous décrivons succinctement les principales étapes de la compilation d'un programme SIGNAL, en insistant plus particulièrement sur le calcul d'horloges et la production du graphe hiérarchisé. Nous introduisons ensuite les principaux outils de spécification, validation et implémentation logicielle et/ou matérielle d'applications disponibles dans l'environnement de SIGNAL.

Nous continuons avec un chapitre sur la mise en œuvre proprement dite du calcul affine d'horloges. Nous décrivons l'implémentation du calcul booléen existant, basée sur une représentation d'un système d'équations sur horloges sous la forme d'un arbre booléen. La résolution du système d'équations est une triangularisation réalisée par fusion récursive d'arbres booléens et preuve d'équivalence des expressions d'horloges situées dans le même arbre. La structure d'arbre booléen permet d'accélérer les réécritures de formules d'horloges (pour les preuves d'équivalences) et constitue une bonne représentation du système d'équations triangulaire que l'on veut construire.

La même méthode de résolution (triangularisation) peut être utilisée dans le cas d'un système d'équations sur des horloges affines et/ou des expressions sur celles-ci (dans la suite, on dénotera par *horloge affine* une horloge obtenue par sous-échantillonnage de phase positive et période strictement positive sur une autre horloge). La structure de représentation correspondant à l'arbre booléen est le *graphe affine* et ses propriétés influencent le déroulement de l'algorithme de triangularisation, comme c'était le cas pour le calcul booléen. Nous étudions ces propriétés et proposons des scénarios pour la résolution du système d'équations sur horloges affines dans le cas où les graphes affines se réduisent aux arbres, ainsi que dans le cas général. Concernant les arbres, nous nous intéressons à l'existence d'une forme canonique d'une formule d'horloges dans un arbre affine et à la fusion d'arbres. La manipulation des graphes affines quant à elle soulève des problèmes plus importants en ce qui concerne la fusion de graphes ou la preuve de synchronisabilité de formules d'horloges appartenant aux graphes différents.

Ces premiers résultats nous permettent d'intégrer les calcul affine et booléen en utilisant comme structure support le *graphe mixte* dans lequel coexistent des expressions d'horloges affines et booléennes. Le prototype que nous avons développé manipule uniquement des arbres mixtes qui regroupent les propriétés des arbres booléens et affines exposées précédemment. En ce qui concerne le cas général des graphes mixtes, nous nous limitons à décrire les principaux problèmes posés par leur manipulation et la manière dont notre prototype peut être étendu afin de les prendre en considération.

Dans le dernier chapitre de cette partie, nous illustrons l'apport de notre travail par des exemples simples de programmes Signal qui nous permettent de présenter le calcul affine d'horloges tel qu'il a été défini et implémenté. Notre travail devient ainsi une contribution au développement de l'environnement de spécification, validation et implémentation de systèmes temps-réel existant autour de Signal. Finalement, les travaux que nous avons effectués sur l'interfaçage des langages Signal et Alpha et le calcul sur les synchronisations des horloges affines en Signal ont été testés sur une application de codage d'images vidéo qui est une application représentative dans le contexte de codesign que nous avons proposé.

Chapitre 9

Le contexte de mise en œuvre

Le prototype de mise en œuvre du calcul formel défini dans le chapitre 7 a été intégré dans le calcul d'horloges existant qui est une étape essentielle de la compilation d'un programme SIGNAL. Le compilateur de SIGNAL est lui-même une partie de l'environnement de programmation existant autour de SIGNAL.

9.1 Environnement de SIGNAL

L'environnement de programmation de SIGNAL, représenté sur la Figure 9.1, contient essentiellement : une interface graphique pour la saisie des programmes (notée `ssignal` sur la figure) ; un compilateur qui communique avec l'interface graphique par une représentation textuelle de programmes et qui associe à chaque programme une représentation interne sous la forme d'un *graphe hiérarchisé* ; un ensemble de logiciels qui utilisent le graphe hiérarchisé pour la génération d'implémentations logicielles ou matérielles, pour la preuve de propriétés, etc.

Avec l'interface (l'éditeur) graphique, la spécification d'une application se fait de manière structurée en précisant ses composants sous la forme de blocs-diagrammes et en fixant leurs connexions. L'éditeur graphique communique avec le compilateur par l'intermédiaire de fichiers. Il utilise également un module du compilateur pour synthétiser l'interface des expressions de processus au fur et à mesure de l'édition graphique.

Toute spécification graphique génère une représentation textuelle qui est soumise ensuite aux diverses actions du compilateur ou des autres logiciels développés autour de SIGNAL. Parmi ces actions, on peut distinguer :

– un ensemble de traitements "frontaux" qui produisent un graphe de flots synchrones à partir d'un programme source en SIGNAL ;

– un ensemble de transformations du graphe de flots synchrones qui restituent un graphe hiérarchisé ; ceci constitue le cœur du compilateur ;

– un ensemble de traitements "dorsaux" qui produisent les sources d'autres outils (implémentations logicielle ou matérielle, outils de preuve formelle, etc...).

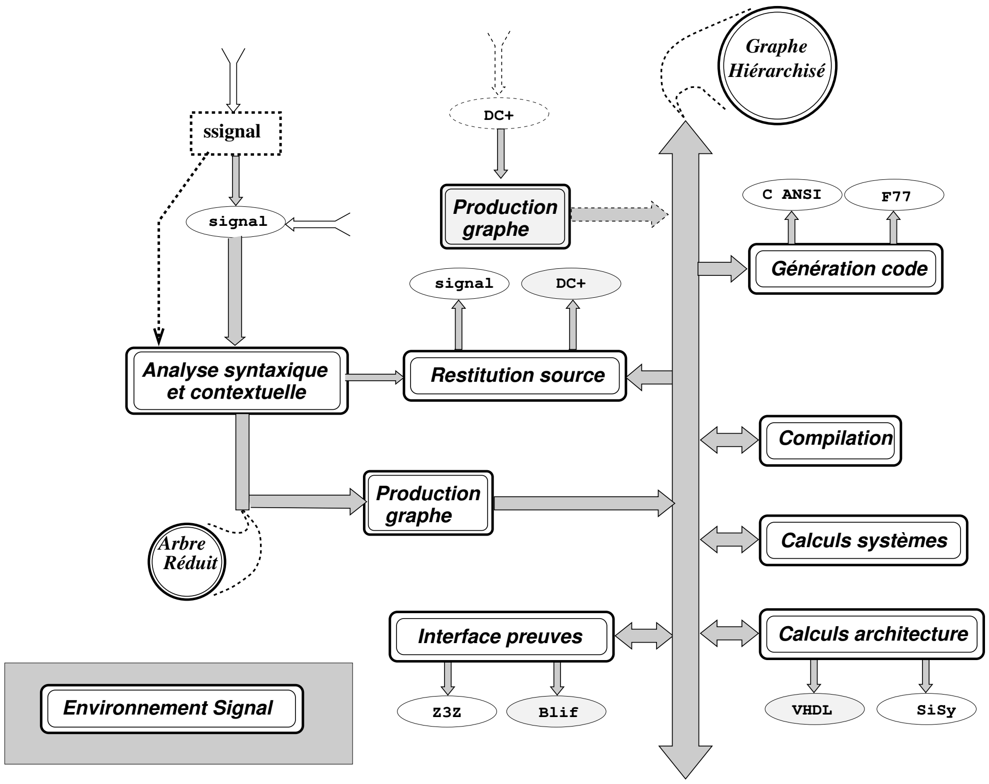

FIG. 9.1 – *Environnement de* SIGNAL.

9.1.1 Traitements frontaux

L'ensemble des traitements pour la production du graphe de flots synchrones est constitué de :

- `l'analyse syntaxique et contextuelle` qui fournit la représentation interne d'un programme source sous la forme d'un arbre (réduit) de syntaxe abstraite ;
- `la production du graphe` qui associe à tout programme du langage réduit un graphe de flots synchrones constitué d'un graphe aux dépendances conditionnées et un système d'équations d'horloges.

9.1.2 Transformations du graphe

L'ensemble des traitements qui transforment le graphe de flots synchrones sont :

- `la compilation` qui comprend principalement les calculs formels sur les synchronisations et les dépendances de données des variables (signaux) ; le graphe hiérarchisé est produit à la fin du processus de compilation ;
- `les opérations de partitionnement du graphe` comme la séparation du contrôle et des calculs ;
- `les calculs systèmes` qui regroupent un ensemble de transformations appliquées au graphe parmi lesquelles la synthèse d'interfaces (l'extraction des éléments de la représentation interne

en vue de la compilation séparée de programmes SIGNAL), le retiming et la projection d'horloges.

9.1.3 Traitements dorsaux

Parmi ceux-ci, nous citons les plus importants :

- `la génération de code séquentiel` (qui passe par un tri topologique du graphe et qui produit du code C pour une implémentation logicielle ou du code VHDL pour l'accès à des outils de synthèse matérielle) `et distribué` ;

- `la restitution du programme source` qui fournit à l'utilisateur le graphe hiérarchisé sous la forme d'un nouveau programme SIGNAL faisant apparaître la hiérarchie obtenue et les synchronisations calculées ; la restitution du source peut également être effectuée en partant d'une représentation sous forme d'arbre de syntaxe abstraite.

- `l'interfaçage avec des systèmes de preuves` : il s'agit actuellement d'un générateur de format intermédiaire pour l'accès à l'outil de vérification de propriétés dynamiques SIGALI et pour la synthèse de contrôleurs;

- `le profiling` qui permet l'évaluation du temps d'exécution des programmes SIGNAL pour diverses implémentations possibles.

9.2 Compilation de programmes SIGNAL

Le point d'entrée du processus de compilation est représenté par le graphe de dépendances et le système d'équations sur horloges générés suite à l'étape de production de graphe. La compilation réalise principalement la vérification de la consistance des contraintes de synchronisations et des chemins de données. Concernant les synchronisations, il s'agit de résoudre le système d'équations sur horloges et de synthétiser une horloge la plus rapide du programme à partir de laquelle toutes les autres horloges peuvent être exprimées et qui sert à cadencer l'exécution du programme. Cette étape est connue sous le nom de *calcul d'horloges* et consiste en une triangularisation du système d'équations sur horloges en utilisant les propriétés du *treillis booléen* des horloges et celles des horloges définies comme sous-échantillonnages par conditions booléennes. A partir de ces propriétés, un ordre $\preceq$ plus grossier que l'ordre du treillis booléen (que nous notons $\leq$ dans le cas des horloges) peut être défini sur l'ensemble des horloges et une structure d'arbre booléen est utilisée pour capturer les propriétés de cet ordre. La triangularisation se réalise ensuite par un processus de fusion d'arbres booléens dans le but d'intégrer toutes les horloges dans un arbre booléen unique. Ceci n'est pas toujours possible s'il existe plusieurs maxima locaux dans l'espace des horloges. L'ensemble des arbres booléens générés à la fin du calcul d'horloges constitue la *hiérarchie d'horloges*. Le programme SIGNAL est déclaré correct du point de vue de ses synchronisations si toutes les contraintes ont été vérifiées en utilisant les propriétés des arbres booléens. Dans le cas d'un arbre unique, la racine représente l'horloge la plus rapide du programme qui cadence son exécution. Il est donc possible de générer du code exécutable dans cette situation et le programme est appelé *endochrone*. Autrement, le programme est *exochrone* et non complètement spécifié du point de vue de ses synchronisations ; le non-déterminisme engendré par cette situation ne permet pas son exécution.

En ce qui concerne les dépendances de données, les vérifications de consistance concernent la détection des cycles de dépendances dans le graphe.

Chaque dépendance étant conditionnée par l'horloge de sa validité, le résultat de la compilation est un graphe hiérarchisé.

9.3 Outils de traitement du graphe hiérarchisé

Il s'agit d'un ensemble de logiciels qui complètent les possibilités offertes par l'environnement de SIGNAL en ce qui concerne la spécification, la validation et l'implémentation de systèmes temps-réel embarqués. Aux calculs formels effectués par le compilateur du langage s'ajoute un système de preuve de propriétés dynamiques de programmes SIGNAL appelé SIGALI. Celui-ci utilise une représentation d'un programme SIGNAL sous un format intermédiaire généré directement à partir du graphe hiérarchisé [Dut92] [ALMR95]. Les calculs formels réalisés par le compilateur simplifient beaucoup cette représentation et réduisent la complexité des preuves formelles réalisées par SIGALI. Dans le même contexte, des travaux récents ont été effectués pour la synthèse de contrôleurs [Mar97] [BLM97] [LM97].

Les générateurs de code (C et VHDL) permettent des simulations fonctionnelles de programmes, ainsi que leur implémentation séquentielle [Maf93] [MCL92] ou matérielle : comme le langage VHDL représente le standard pour l'accès aux outils de synthèse matérielle, on a choisi de générer du code VHDL comportemental [Bel94b] [Bel94a] et architectural [KW97] [ABK+97] à partir de SIGNAL. Il existe aussi la possibilité de mise en œuvre qualitative [Aub97] [AM95] et quantitative [LSSS91] de programmes synchrones sur des architectures multi-processeurs, ainsi qu'un outil de synthèse de circuits asynchrones à partir de SIGNAL [Bel93] [Bel94a]. Finalement, le profiling d'un programme SIGNAL évalue le temps de son exécution [KL96] en fonction d'une implémentation particulière.

Chapitre 10

Représentation de relations affines et calcul affine d'horloges

Dans ce chapitre, nous présentons le *calcul affine d'horloges* (ou tout simplement *calcul affine*) comme une extension du système de preuve sur les équations d'horloges existant actuellement dans le compilateur, que nous allons identifier comme le *calcul booléen d'horloges* ou *calcul booléen.* Tout d'abord, nous présentons les principes de base du calcul booléen en insistant sur l'influence de la structure de *treillis d'ordre* $\preceq$ sur la méthode de résolution du système d'équations sur horloges. L'idée de la résolution est de *triangulariser* le système d'équations initial en utilisant *l'arbre booléen* comme de structure de représentation.

Nous décrivons ensuite les extensions de ces concepts pour la résolution du système d'équations sur horloges en relation affine – que nous identifierons comme *horloges affines* – en utilisant les règles de synchronisabilité présentées dans la partie précédente. La méthode de résolution proposée est une triangularisation qui tire parti de la structure de treillis d'ordre $\preceq_{aff}$ que l'on définit sur l'espace des horloges affines et de la représentation d'un système triangulaire sous la forme d'un *graphe affine.* Nous analysons les propriétés des graphes affines et leur influence sur le calcul affine d'horloges. A partir de cette étude préliminaire, nous proposons une méthode d'intégration des calculs affine et booléen et en présentons les limites, induites tant par les choix d'implémentation que par les propriétés du treillis $\preceq_{aff}$ existant sur l'espace des horloges affines.

10.1 Calcul booléen d'horloges

10.1.1 Arbre de partition

Le calcul d'horloges implémenté actuellement dans le compilateur ne se propose pas de déterminer l'ensemble de toutes les horloges vérifiant le système d'équations associé à un programme Signal, mais de mettre en évidence un vecteur *d'horloges-paramètres (paramétrisation)* en fonction duquel toutes les autres horloges du programme peuvent être exprimées par *triangularisation.* Le choix fait en Signal a été de considérer le vecteur de paramétrisation formé soit d'horloges[1] libres $\hat{x}$, soit d'éléments appartenant à des triplets de la forme ($\hat{x}$, $[x]$, $[\neg x]$), avec $\hat{x}$ horloge de la variable booléenne x et $[x]$, $[\neg x]$ les horloges des instants où x prend la valeur *true*, respectivement *false.* La résolution du système d'équations sur horloges est réalisée ensuite par triangularisation [Ama95] du système initial : chaque équation de la forme $h = h_1$ ou $h = h_1 \ op \ h_2$, avec $op \in \{\vee, \wedge, \setminus\}$, est orien-

1. Dans la suite, nous adoptons la notation $\hat{x}$ pour l'horloge $h(x)$ de x.

tée de droite à gauche afin de considérer l'horloge h_1, respectivement l'expression $h_1 \; op \; h_2$, comme la définition de l'horloge h[2]. Un système triangulaire (graphe de dépendances horloges-horloges) est progressivement construit à partir de la paramétrisation initiale. Toute équation dont la partie droite est une formule construite avec des éléments appartenant au vecteur de paramétrisation est orientée comme décrit précédemment et l'horloge de la partie gauche de l'équation peut être utilisée ensuite pour définir de nouvelles horloges.

La construction du système triangulaire se réalise en deux étapes principales. La première étape consiste dans la création, pour toute variable x, de l'arbre de partition associé à $\hat{x}$: la structure obtenue en associant à $\hat{x}$ l'ensemble de toutes les partitions générées par les conditions booléennes y d'horloges $\hat{y}$ égales à l'horloge $\hat{x}$, ainsi que l'ensemble des horloges $\hat{z}$, avec z signal non booléen, égales à $\hat{x}$ (voir Figure 10.1).

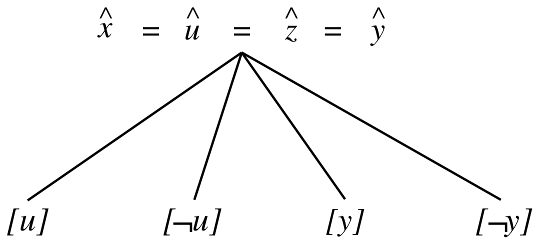

FIG. 10.1 – *Arbre de partition.*

Un arc dans l'arbre modélise la relation d'inclusion $\leq$ père-fils. Chaque racine est définie par un ensemble fini d'horloges égales entre elles ; sur la Figure 10.1, il s'agit des horloges $\hat{x}$, $\hat{u}$, $\hat{z}$ et $\hat{y}$. Un représentant doit être désigné parmi ces horloges. Choisir un représentant est équivalent à orienter l'ensemble des équations associées à la racine, conformément au principe du processus de triangularisation décrit précédemment : toutes les horloges de la racine seront définies en fonction du représentant. Si x est un signal booléen, l'horloge $\hat{x}$ devient le représentant de la racine de l'arbre de partition associé à $\hat{x}$. Dans le cas contraire, toute autre horloge $\hat{y}$ associée à un signal booléen et égale à $\hat{x}$ est choisie comme représentant; si une telle horloge n'existe pas, l'arbre de partition associé à $\hat{x}$ contient comme nœud unique la racine $\hat{x}$ et toute horloge $\hat{z}$ égale à $\hat{x}$ peut être considérée comme représentant.

Le représentant de la racine d'un arbre de partition peut être contraint par d'autres équations sur horloges – *contraintes (équations) externes* – qui doivent être orientées de manière analogue. Si h est l'horloge représentant de la racine de l'arbre de partition associé à $\hat{x}$ et $formule(h_i)$ est une expression d'horloges contenant les opérateurs $\vee$, $\wedge$ et/ou $\setminus$, l'orientation de l'équation (contrainte) externe $h = formule(h_i)$ est impossible dans certains cas. Il existe deux raisons pour cela :

- L'orientation de l'équation $h = formule(h_i)$ induirait un cycle dans le graphe de dépendances horloges-horloges. Dans ce cas, on tente de réécrire $formule(h_i)$ afin de briser le cycle.

- L'horloge h possède déjà une définition et la définition de h par la formule $formule(h_i)$ doit être prouvée équivalente à la définition initiale.

Au cours de la triangularisation, il peut s'avérer nécessaire d'utiliser les axiomes de l'espace des horloges comme règles de réécriture de formules afin de briser des cycles ou pour prouver l'équivalence de formules. Pour cela, un ordre partiel $\preceq$ plus "grossier" que l'ordre du treillis $\leq$ a

2. Toute équation $h_1 \; op \; h_2 = h'_1 \; op \; h'_2$ est réécrite en $h = h_1 \; op \; h_2$ et $h = h'_1 \; op \; h'_2$ et on se situe dans le cas de la règle précédente.

été défini sur l'espace des horloges. Cet espace devient un treillis d'ordre $\preceq$ dans lequel les axiomes du treillis booléen, les équations de partition et toute autre équation induite à partir de celles-ci restent valides. De plus, chaque arbre de partition est un $\vee$-semi-treillis complet par rapport à l'ordre $\preceq$. La fusion des arbres de partition, notés $\mathcal{T}_{\hat{x}_i}^{part}$, permet d'affiner la relation d'inclusion obtenue par union des relations d'inclusion $\preceq$ représentées par les $\mathcal{T}_{\hat{x}_i}^{part}$. Suite à la fusion, des $\vee$-semi-treillis complets plus généraux sont créés. Ils sont représentés sous la forme d'*arbres booléens*. L'algorithme de fusion peut être réitéré sur les arbres booléens dans le but d'intégrer l'ensemble des horloges d'un programme dans un arbre booléen unique, afin d'affiner au plus possible la relation d'inclusion existante et d'obtenir une unique horloge la plus rapide du programme. L'orientation des équations externes associées aux racines des arbres de partition par fusion itérative d'arbres booléens constitue la deuxième étape et le cœur de la triangularisation.

Dans le cas où, à la fin du calcul d'horloges, il existe une unique borne supérieure par rapport à $\preceq$, le programme est endochrone et son exécution peut être cadencée par la borne supérieure (l'horloge la plus rapide). Au cas contraire, le programme est incomplètement spécifié du point de vue de ses contraintes d'horloges et ne peut pas être exécuté ; il s'agit dans ce cas d'un programme exochrone. Dans tous les cas, le programme est déclaré correct par rapport à ses synchronisations si le graphe horloges-horloges ne contient pas de cycle et toutes les définitions multiples ont été prouvées équivalentes.

Dans les sections suivantes, nous présentons l'arbre booléen et ses propriétés qui le rendent parfaitement adapté à la méthode de résolution proposée. Nous décrivons ensuite le cœur de la triangularisation et concluons sur les limites et extensions possibles du calcul d'horloges.

10.1.2 Arbre booléen

On transforme l'ordre $\leq$ dans un ordre plus "grossier" $\preceq$ dont on donne ci-dessus la définition [3].

Définition 20 *Pour tout signal booléen x, on définit sur l'ensemble d'horloges $\mathcal{T}_{\hat{x}}^{\preceq}$ ci-dessous*

$$\begin{array}{lll} \mathcal{T}_{\hat{x}}^{\preceq} & =_{\triangle} & \{\hat{x}, [x], [\neg x]\} \ \bigcup \\ & & \{\ h\ /\ h \textit{ peut être définie seulement par une des expressions qui suivent :} \\ & & \qquad \bullet\ h = \hat{y},\ \textit{où le signal } y \textit{ a la propriété que } \exists\ h' \in \mathcal{T}_{\hat{x}}^{\preceq}, \hat{y} = h', \\ & & \qquad \bullet\ h = [\pm x_1]\ \ \textit{ou } h = [\pm x_1]\ op\ [\pm x_2],\ \textit{avec } x_1, x_2 \textit{ signaux booléens} \\ & & \qquad\ \ \textit{tels que } \exists\ h_1, h_2 \in \mathcal{T}_{\hat{x}}^{\preceq}, \hat{x_1} = h_1, \hat{x_2} = h_2 \textit{ et } \ op \in \{\vee, \wedge, \backslash\}\ \} \end{array}$$

un ordre $\preceq$ de la manière suivante :

(1) *Pour tout triplet ($\hat{y}$, $[y]$, $[\neg y]$) appartenant à $\mathcal{T}_{\hat{x}}^{\preceq}$, on a*

$$\begin{array}{l} [y] \preceq \hat{y} \\ [\neg y] \preceq \hat{y} \end{array}$$

(2) *Pour toute expression $E \in \mathcal{T}_{\hat{x}}^{\preceq}$, construite sur $\{\hat{x_i}\} \subseteq \mathcal{T}_{\hat{x}}^{\preceq}$ avec les opérateurs $\{\vee, \wedge, \backslash\}$, s'il existe $\hat{y} \in \mathcal{T}_{\hat{x}}^{\preceq}$ telle que $\hat{x_i} \preceq \hat{y}, \forall i$, alors :*

$$E(\ldots, \hat{x_i}, \ldots) \preceq \hat{y}$$

3. La notation $[\pm x]$ est utilisée pour $[x]$ ou $[\neg x]$.

(3) *Pour tout doublet* $(\mathcal{T}_{\hat{x}}^{\preceq}, \mathcal{T}_{\hat{y}}^{\preceq})$ *tel que* $\hat{x} \notin \mathcal{T}_{\hat{y}}^{\preceq}$ *et* $\hat{y} \notin \mathcal{T}_{\hat{x}}^{\preceq}$, *on a* $\hat{x_i} \neq \hat{y_i}$, $\forall \hat{x_i} \in \mathcal{T}_{\hat{x}}^{\preceq}$ *et* $\forall \hat{y_i} \in \mathcal{T}_{\hat{y}}^{\preceq}$.

□

Les conditions (1), (2) et (3) définissent complètement l'ordre $\preceq$ sur l'ensemble $\mathcal{T}_{\hat{x}}^{\preceq}$, pour tout signal booléen x. En effet, les horloges des sous-échantillonnages booléens $[y]$ et $[\neg y]$ sont inférieures à l'horloge $\hat{y}$ et ainsi à toute horloge supérieure à $\hat{y}$ (condition (1)). Si les termes d'une expression d'horloges E sont inférieurs à $\hat{y}$, l'expression est aussi inférieure à $\hat{y}$ (condition (2)). Finalement, deux ensembles $\mathcal{T}_{\hat{x}}^{\preceq}$ et $\mathcal{T}_{\hat{y}}^{\preceq}$ ne partagent pas d'éléments s'ils ne partagent pas leurs racines.

$\mathcal{T}_{\hat{x}}^{\preceq}$ est un $\vee$–semi-treillis complet par rapport à $\preceq$: étant données les horloges $h_1, h_2 \in \mathcal{T}_{\hat{x}}^{\preceq}$, la borne supérieure de h_1 et h_2 par rapport à $\preceq$ appartient à $\mathcal{T}_{\hat{x}}^{\preceq}$. En effet, il est facilement vérifiable que la borne supérieure de $[y]$ et $[\neg y]$ est $\hat{y}$, $\forall \hat{y} \in \mathcal{T}_{\hat{x}}^{\preceq}$. Pour deux expressions $E(\ldots, \hat{y_i}, \ldots)$ et $E(\ldots, \hat{z_i}, \ldots)$ appartenant à $\mathcal{T}_{\hat{x}}^{\preceq}$, leur borne supérieure est égale à la borne supérieure de $\hat{y}$ et $\hat{z}$, avec $\hat{y}$ la borne supérieure de $\hat{y_i}$, $\forall i$, et $\hat{z}$ la borne supérieure de $\hat{z_i}$, $\forall i$.

On ne dispose pas actuellement d'un ensemble de règles de synchronisabilité qui permettent d'identifier des horloges appartenant à des semi-treillis $\mathcal{T}_{\hat{x}}^{\preceq}$ et $\mathcal{T}_{\hat{y}}^{\preceq}$ différents (qui ne partagent pas leurs racines). Ceci impliquerait une étude sur les valeurs des conditions booléennes qui définissent les horloges et les formules sur horloges dans des semi-treillis $\mathcal{T}_{\hat{x}}^{\preceq}$. Dans le contexte des horloges affines, nous allons pouvoir éliminer la condition (3) en utilisant les règles de synchronisabilité générales que nous avons déduites dans la deuxième partie de ce document. Sur l'ensemble total des expressions sur sous-échantillonnages par conditions booléennes, l'ordre $\preceq$ définit une structure de $\vee$-semi-treillis (on va également utiliser le terme "treillis" pour dénommer la même structure) : quand h_1, h_2 appartiennent à des semi-treillis $\mathcal{T}_{\hat{x_1}}^{\preceq}$, $\mathcal{T}_{\hat{x_2}}^{\preceq}$ différents, $h_1 \leq h_2$ implique $h_1 \preceq h_2$.

Pour un signal booléen x, un semi-treillis $\mathcal{T}_{\hat{x}}^{\preceq}$ est représenté par un *arbre d'horloges booléen* ou *arbre booléen* : chaque arc de l'arbre modélise la relation d'inclusion $\preceq$ et chaque père représente la borne supérieure de tous ses fils. Les éléments de l'arbre différents de la racine sont des expressions d'horloges construites sur des conditions *booléennes*. Nous représentons dans la Figure 10.2 l'exemple d'un arbre booléen. A chaque nœud de l'arbre est associé un ensemble d'horloges équivalentes (ou synchrones) parmi lesquelles un représentant est choisi de la même manière que pour les racines des arbres de partition.

L'arbre booléen ne représente pas seulement une modélisation d'un treillis $\mathcal{T}_{\hat{x}}^{\preceq}$ d'ordre $\preceq$, mais constitue aussi une structure adéquate pour mettre en évidence la triangularité du système d'équations sur horloges que l'on veut construire : chaque expression E construite sur $\{\hat{x_i}\} \subseteq \mathcal{T}_{\hat{x}}^{\preceq}$ se situe dans l'arbre à droite par rapport aux opérandes $\hat{x_i}$ afin que tout parcours descendant gauche-droite dans l'arbre calcule les opérandes $\hat{x_i}$ avant l'expression E qui les utilise.

Un arbre de partition représente un cas particulier d'arbre booléen. L'idée du calcul d'horloges est de fusionner les arbres de partition construits initialement afin d'obtenir un nombre minimal de treillis d'ordre $\preceq$ représentés par des arbres d'horloges booléens. La relation d'inclusion constituée par l'union des relations d'inclusion individuelles – les arbres de partition ou booléens – est ainsi affinée progressivement jusqu'à ce que ce processus ne puisse plus être réitéré. Les propriétés des arbres booléens sont largement utilisées dans l'algorithme de fusion et pour la preuve d'équivalence d'expressions d'horloges. Le programme SIGNAL est déclaré correct du point de vue de ses synchronisations si, à la fin du processus de triangularisation, le graphe de dépendances horloges-horloges

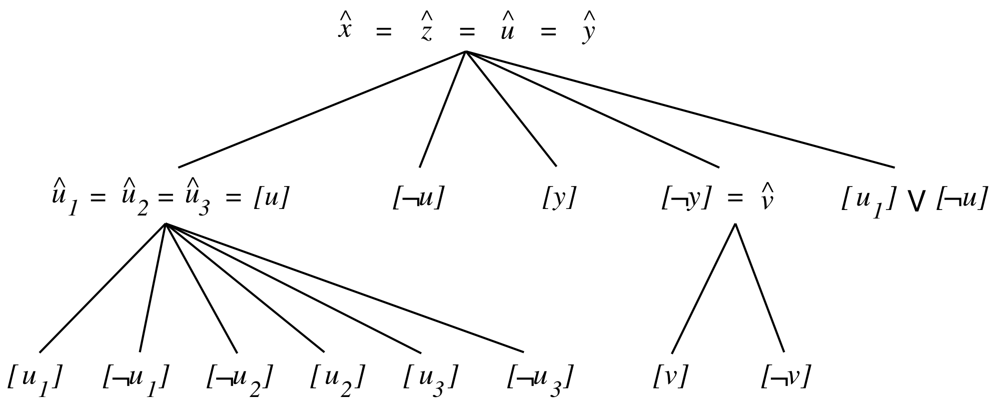

FIG. 10.2 – *Arbre booléen.*

ne contient aucun cycle et toutes les équivalences ont pu être prouvées. Si, de plus, la triangularisation produit un seul arbre booléen, le programme SIGNAL est déclaré exécutable et la racine de l'arbre représente l'horloge la plus rapide du programme, celle qui cadence son exécution et à partir de laquelle toutes les autres horloges peuvent être calculées.

10.1.3 Triangularisation

Nous détaillons dans cette section le cœur de l'algorithme de triangularisation, i.e. la fusion d'arbres booléens et la preuve d'équivalence de formules d'horloges. Par "arbre booléen" on sous-entend dans la suite un arbre de partition ou tout arbre d'horloges construit dans une phase intermédiaire de la triangularisation.

La première étape de la triangularisation consiste dans la construction des arbres de partition : chaque horloge d'une racine sera ainsi définie en fonction du représentant de la racine (voir section 10.1.1 pour le choix du représentant de la racine). L'ensemble des arbres de partition forme *une forêt d'horloges*. La deuxième étape de la triangularisation est un processus itératif de fusion d'arbres que nous décrivons dans la suite ; le but est de triangulariser les équations externes qui définissent les représentants associés aux racines des arbres de partition. La fusion de deux arbres de partition produit un arbre booléen ; en même temps le nombre d'arbres de la forêt d'horloges est réduit et le processus de fusion peut être réitéré sur les nouveaux arbres booléens avec comme but la réduction de la forêt à un seul arbre et la triangularisation complète du système.

Pour une étape arbitraire du processus de triangularisation, considérons deux éléments T et T' de la forêt d'arbres booléens. Les racines de T et T' sont r et r' (r et r' sont les représentants des racines pour chacun des arbres T et T'). On considère que :

- La racine r' est contrainte par une équation externe de la forme $r' = h_1\ op\ h_2$ ou $r' = h_3$;
- Les opérandes h_1, h_2 et h_3 appartiennent à T, h_3 différent de r.

(on considère que le système d'équations sur horloges a été transformé de telle sorte que dans toute équation $r = formule(h_i)$, $formule(h_i)$ contient au plus deux horloges).

Dans ces conditions, la fusion des arbres T et T' produit l'arbre T'' en insérant la racine r' de T' dans T et en faisant de T' un sous-arbre de T. Les Figures 10.3, 10.4 montrent le processus de fusion pour les deux définitions possibles de l'horloge r'.

Dans le premier cas, la racine r' est placée sous le premier ancêtre commun de h_1 et h_2 dans T. L'arbre T'' résultant est un arbre booléen au sens de la définition donnée dans le paragraphe

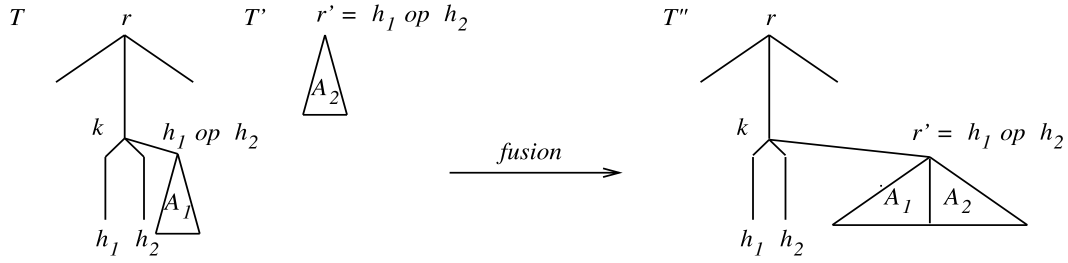

FIG. 10.3 – *Fusion d'arbres - cas 1.*

précédent. La fusion des arbres T et T' est équivalente à une orientation de l'équation externe $r' = formule(h_i)$ ($r' = h_1$ *op* h_2 ou $r' = h_3$) afin que $formule(h_i)$ devienne la définition de r'. Cette orientation est possible puisque r' ne possède pas une autre définition.

Dans les Figures 10.3 et 10.4 on peut remarquer que, après la fusion, le sous-arbre correspondant à h_1 *op* h_2 dans T sera l'union *disjointe*[4] des sous-arbres A_1 et A_2 ; dans le cas des arbres booléens, l'union disjointe coïncide avec l'union classique: effectivement, les sous-arbres A_1 et A_2 ont la propriété que leurs horloges sont différentes[5] et aucune unification ne peut être effectuée entre leurs horloges respectives.

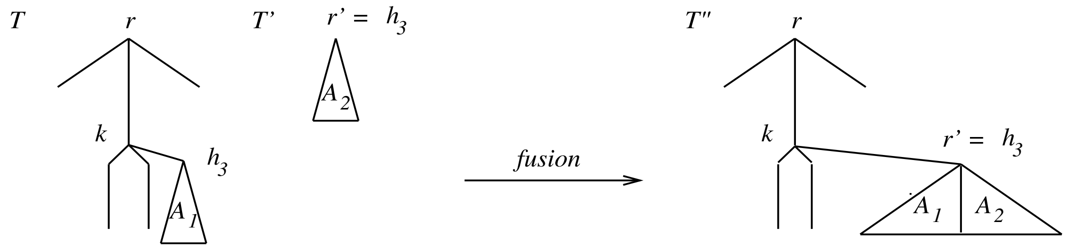

FIG. 10.4 – *Fusion d'arbres - cas 2.*

Suite au processus de fusion ci-dessus, r' devient le représentant d'un nœud de T'' différent de la racine. Toute autre contrainte sur r' de la forme $r' = f'$, avec $f' = h'_1$ *op* h'_2 ou $f' = h'_3$, ne peut plus être orientée afin que f' devienne la définition de r' puisque r' possède déjà une définition f. Dans ce cas, il est nécessaire de prouver l'équivalence de f et f'. Si f' appartient à l'arbre T'' ou f' peut être réécrite pour qu'elle appartienne à T'', la preuve d'équivalence consiste à vérifier l'identité des formes canoniques arborescente de f et f' [Ama95]. Dans le cas où f' n'appartient pas à T'', f et f' ne peuvent pas être prouvées équivalentes suite aux propriétés des arbres booléens (conformément à la condition (3) de la définition de l'ordre $\preceq$, deux arbres booléens différents ne peuvent pas partager des éléments).

En conclusion, la triangularisation peut être vue comme une succession de deux pas principaux: 1) fusion itérative d'arbres booléens ; 2) preuve d'équivalence des formules d'horloges qui définissent la même horloge. Même si en pratique ces deux pas ne sont pas strictement séquentiels, nous préférons les présenter ainsi pour plus de clarté.

L'algorithme suivant résume les principaux pas de la triangularisation.

1. Pour chaque arbre T' de la forêt d'horloges, on tente de réécrire la formule de définition de

4. L'union disjointe des ensembles $\{0, 1, 2, 3\}$ et $\{2, 3, 4, 5, 6\}$ est l'ensemble $\{0, 1, 2, 3, 2, 3, 4, 5, 6\}$; pour deux ensembles disjoints, le résultat de l'union disjointe coïncide avec le résultat de l'opération d'union classique.

5. Les horloges ne peuvent pas être prouvées égales à partir des propriétés de la structure de treillis d'ordre $\preceq$.

sa racine r' afin que les opérandes de r' se trouvent dans le même arbre T (si la condition est vérifiée, mais la formule de définition contient un cycle, une tentative de réécriture est effectuée afin de briser le cycle). En cas de succès, la racine r' peut être insérer dans T.

2. La fusion de T et T' est effectuée afin d'obtenir T'' comme décrit précédemment.

   Après la fusion, une formule d'horloges qui a un opérande dans T et un autre dans T' va avoir ses deux opérandes dans l'arbre T''. La fusion de T et T' entraîne ainsi des fusions ultérieures.

3. On applique les pas 1 et 2 ci-dessus tant qu'il existe des règles de réécriture applicables.

4. On tente de prouver l'équivalence des formules qui définissent la même horloge.

Pour plus de détail sur la triangularisation, avec des exemples concrets de programmes SIGNAL pour illustrer la fusion d'arbres et la preuve d'équivalence de formules placées, nous renvoyons le lecteur à [Ama95]. Pour une équation $r = f$, la réécriture de la formule f pour obtenir une expression dont les opérandes se situent dans le même arbre T utilise des règles de réécriture déduites directement à partir des axiomes du treillis d'ordre $\preceq$. Plus généralement, ces axiomes ont permis de définir une *forme canonique arborescente* [Ama95] de toute formule f par rapport à l'arbre T : la formule f est réécrite en une expression *unique* E construite avec des horloges appartenant à l'arbre T. Cette expression permet ensuite un placement unique de f dans T comme fille de la borne supérieure de toutes les horloges qui interviennent dans l'expression E. La forme canonique arborescente est utilisée pour démontrer l'équivalence de deux formules f_1 et f_2 situées dans le même arbre : les formes canoniques de f_1 et f_2 doivent être prouvées égales.

10.1.4 Conclusion

Le calcul d'horloges booléen est basé sur un système de règles de réécriture déduites à partir des axiomes du treillis induit sur l'espace des horloges par l'ordre $\preceq$. Ces axiomes sont en fait des règles de synchronisabilité d'horloges ; en section 7.4 nous avons présenté la connexion entre le calcul d'horloges et le concept de synchronisabilité.

Le système de règles de synchronisabilité à la disposition du calcul d'horloges booléen est incomplet et extensible ; il peut être enrichi à partir d'une étude de nouvelles relations qui peuvent exister entre horloges. Ainsi nous avons étudié les relations affines entre horloges et nous avons proposé des règles de synchronisabilité nouvelles basées sur leurs propriétés. Avec ces résultats, le calcul affine d'horloges peut être défini d'après le modèle du calcul booléen existant. D'abord, une structure de semi-treillis d'ordre $\preceq_{aff}$ similaire à $\preceq$ peut être définie sur l'espace des expressions construites sur des horloges affines (sous-échantillonnages affines de phase positive et période strictement positive comme définies en section 7.4.2.2). La résolution du système d'équations sur horloges affines peut être réalisée ensuite par triangularisation. Le système triangulaire final sera représenté sous la forme d'un ensemble de $\vee$-semi-treillis d'ordre $\preceq_{aff}$. Les règles de synchronisabilité introduites dans le chapitre 7 deviennent des axiomes d'équivalence de formules d'horloges appartenant à un même semi-treillis d'ordre $\preceq_{aff}$, ou des propriétés liées à la possibilité de synchroniser des horloges appartenant à des semi-treillis différents.

Nous détaillons ces aspects dans les sections suivantes. En première instance, nous considérons un calcul affine indépendant du calcul booléen. Ensuite, nous proposons une méthode d'intégration des deux calculs.

10.2 Calcul affine d'horloges

10.2.1 Principe

Les trois opérateurs affines introduits en section 7.4.2.1 permettent de définir des relations affines entre horloges. A chaque processus affine est associé un ensemble d'équations qui font intervenir des horloges affines ou des expressions sur celles-ci (voir section 7.4.2.2). Une horloge affine est une horloge définie par un sous-échantillonnage affine de phase φ positive et période n strictement positive sur tout autre horloge. Les équations obtenues à partir des opérateurs affines représentent des contraintes d'horloges auxquelles peuvent s'ajouter d'autres contraintes implicites ou explicites exprimées avec les opérateurs SIGNAL existants. Dans ce cas, le calcul d'horloges est confronté au problème de la résolution de ces contraintes de synchronisation en prenant en compte les propriétés des horloges affines.

Nous définissons un tel calcul basé sur l'ensemble des règles de synchronisabilité présentées en section 7.3. D'abord, nous proposons un calcul affine d'horloges indépendant du calcul booléen existant, c'est-à-dire dans l'absence des horloges sous-échantillonnages par conditions booléennes et des expressions sur celles-ci. L'idée la plus naturelle est de voir dans quelle mesure les méthodes présentes dans le calcul booléen sont adaptables au nouveau calcul. A l'appui de cette idée vient l'ordre $\preceq_{aff}$ qui peut être défini sur l'espace des horloges affines de manière similaire à l'ordre $\preceq$ (voir section 10.1.2). Nous élargissons les notions d'arbre de partition et arbre booléen à celles de *graphe affine élémentaire* et *graphe affine*. Nous étudions l'influence de leur propriétés sur la méthode de résolution choisie, qui consistera dans une triangularisation d'après le modèle de celle existante.

10.2.2 Graphe affine

Une horloge affine est un élément différent de la racine d'un *graphe affine élémentaire*: pour chaque variable x on peut associer à son horloge $\hat{x}$, sous la forme d'un graphe affine élémentaire, la structure obtenue en associant à $\hat{x}$ l'ensemble de toutes les horloges sous-échantillonnages affines construites sur $\hat{x}$ ou sur toute autre horloge $\hat{y}$ égale à $\hat{x}$, ainsi que l'ensemble des horloges $\hat{y}$ égales à $\hat{x}$.

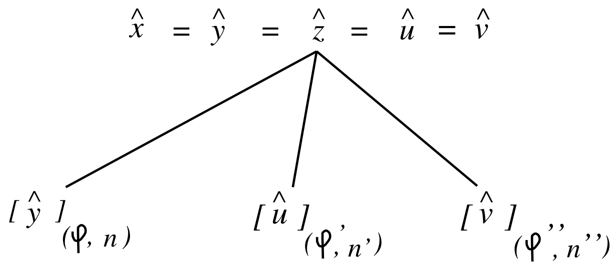

FIG. 10.5 – *Graphe affine élémentaire.*

La Figure 10.5 présente l'exemple d'un graphe affine élémentaire qui contient les horloges affines $[\hat{y}]_{(\varphi,n)}$, $[\hat{u}]_{(\varphi',n')}$ et $[\hat{v}]_{(\varphi'',n'')}$: la notation $[\hat{y}]_{(\varphi,n)}$ indique un sous-échantillonnage affine de phase φ et période n sur l'horloge $\hat{y}$.

Tout graphe affine élémentaire se réduit à un *arbre affine élémentaire*. L'arbre affine élémentaire peut être considéré comme un arbre de partition dans lequel les extractions par conditions booléennes ont été remplacées par des sous-échantillonnages affines de paramètres φ et n. Comme pour les arbres de partition, un ensemble d'horloges égales (ou équivalentes) est affecté à chaque racine d'un arbre affine élémentaire (pour l'exemple de la Figure 10.5, il s'agit de l'ensemble $\{\hat{x}, \hat{y}, \hat{z}, \hat{u}, \hat{v}\}$). Un représentant par racine est choisi de manière similaire au cas des arbres de partition : l'horloge représentante $\hat{y}$ aura la propriété qu'il existe une horloge affine $[\hat{y}]_{(\varphi,n)}$ appartenant à l'arbre considéré; dans le cas où il n'existe pas d'horloge avec la propriété précédente, l'arbre affine élémentaire associé à $\hat{x}$ contient comme nœud unique la racine $\hat{x}$ et toute horloge $\hat{z}$ égale à $\hat{x}$ peut être considérée comme représentant.

Le représentant de la racine d'un arbre affine élémentaire peut être contraint par d'autres équations sur horloges – *contraintes (équations) externes.* Les horloges associées aux autres nœuds de l'arbre (différents de la racine) sont contraintes uniquement par le fait qu'il existe une relation affine de paramètres $((\varphi_1, n_1), (\varphi_2, n_2))$ entre chaque paire de feuilles (comme exemple, dans l'arbre représenté sur la Figure 10.5 il existe une relation affine de paramètres $((\varphi, n), (\varphi', n'))$ entre les horloges $[\hat{y}]_{(\varphi,n)}$ et $[\hat{u}]_{(\varphi',n')}$).

Les *graphes affines* sont des unions d'arbres affines et un *arbre affine* est la représentation d'un ensemble $\mathcal{T}_{\hat{x}}^{\preceq_{aff}}$ dont toutes les horloges sont limitées supérieurement, par rapport à l'ordre $\preceq_{aff}$, par l'horloge $\hat{x}$. Pour la définition formelle de l'ordre $\preceq_{aff}$, nous renvoyons le lecteur à la section 7.4.2.2 ; à partir de cette définition, on peut vérifier facilement les propriétés de reflexivité, symétrie et transitivité de tout ordre partiel.

Nous présentons maintenant une autre définition de l'ordre $\preceq_{aff}$ qui fait apparaître les propriétés de cet ordre dans l'espace $\mathcal{T}_{\hat{x}}^{\preceq_{aff}}$ et qui permet d'introduire les notions d'arbre affine et graphe affine.

Définition 21 *Pour un ensemble de signaux x_i, on définit sur l'union des ensembles d'horloges $\mathcal{T}_{\hat{x}_i}^{\preceq_{aff}}$ donnés par l'expression*

$$\begin{array}{rcl}\mathcal{T}_{\hat{x}_i}^{\preceq_{aff}} & =_{\triangle} & \{\hat{x_i}\} \bigcup \{\ [\hat{x_i}]_{(\varphi,n)}\ /\ \varphi, n \in I\!N \textit{ arbitraires }\ \} \bigcup \\ & & \{\ h\ /\ h \textit{ peut être définie seulement par une des expressions suivantes :} \\ & & \qquad \bullet\ h = \hat{y},\ \textit{où le signal } y \textit{ a la propriété que } \exists\ h' \in \mathcal{T}_{\hat{x}_i}^{\preceq_{aff}}, \hat{y} = h', \\ & & \qquad \bullet\ h = [\hat{x_1}]_{(\varphi_1,n_1)}\ \textit{ ou } h = [\hat{x_1}]_{(\varphi_1,n_1)}\ \textit{op}\ [\hat{x_2}]_{(\varphi_2,n_2)},\ \textit{où les} \\ & & \qquad \textit{les signaux } x_1, x_2 \textit{ ont la propriété que } \exists\ h_1, h_2 \in \mathcal{T}_{\hat{x}_i}^{\preceq_{aff}}, \\ & & \qquad \hat{x_1} = h_1, \hat{x_2} = h_2,\ \textit{ et }\ \textit{op} \in \{\vee, \wedge, \backslash\}\ \}\end{array}$$

un ordre $\preceq_{aff}$ de la manière suivante :

(1) *Pour tout doublet $(\hat{y}, [\hat{y}]_{(\varphi,n)})$ appartenant à $\mathcal{T}_{\hat{x}_i}^{\preceq_{aff}}$ fixé, on a*

$$[\hat{y}]_{(\varphi,n)} \preceq_{aff} \hat{y}$$

(2) *Pour toute expression $E \in \mathcal{T}_{\hat{x}_i}^{\preceq_{aff}}$ fixé, construite sur $\{\hat{y_j}\} \subseteq \mathcal{T}_{\hat{x}_i}^{\preceq_{aff}}$ avec les opérateurs $\{\vee, \wedge, \backslash\}$, s'il existe $\hat{y} \in \mathcal{T}_{\hat{x}_i}^{\preceq_{aff}}$ telle que $\hat{y_j} \preceq_{aff} \hat{y}, \forall j$, alors :*

$$E(\ldots, \hat{y_j}, \ldots) \preceq_{aff} \hat{y}$$

(3) *Une horloge h peut appartenir à deux ensembles partiellement ordonnés $\mathcal{T}_{\hat{x}_i}^{\preceq_{aff}}$, $\mathcal{T}_{\hat{x}_j}^{\preceq_{aff}}$ différents : pour tout doublet $(\mathcal{T}_{\hat{x}_i}^{\preceq_{aff}}, \mathcal{T}_{\hat{x}_j}^{\preceq_{aff}})$ tel que $x_i \notin \mathcal{T}_{\hat{x}_j}^{\preceq_{aff}}$ et $x_j \notin \mathcal{T}_{\hat{x}_i}^{\preceq_{aff}}$, l'ensemble*

$$\{ \ h \ / \ (\exists) \ h_1 \in \mathcal{T}_{\hat{x}_i}^{\preceq_{aff}}, h_2 \in \mathcal{T}_{\hat{x}_j}^{\preceq_{aff}} \ \textit{tels que} \ h_1 \neq \hat{x}_i, h_2 \neq \hat{x}_j \ \textit{et} \ h = h_1 = h_2 \ \}$$

est non vide.

□

L'ordre partiel $\preceq_{aff}$ définit sur l'espace des expressions sur horloges affines une structure de $\vee$-semi-treillis qui, contrairement au cas booléen, n'est pas complet. En effet, pour un signal x donné, $\mathcal{T}_{\hat{x}}^{\preceq_{aff}}$ est limité supérieurement par la racine $\hat{x}$, mais $\hat{x}$ n'est pas toujours la plus petite limite supérieure de toutes les horloges de $\mathcal{T}_{\hat{x}}^{\preceq_{aff}}$ (i.e. $\hat{x}$ n'est pas la borne supérieure de $\mathcal{T}_{\hat{x}}^{\preceq_{aff}}$ et $\mathcal{T}_{\hat{x}}^{\preceq_{aff}}$ n'est pas un $\vee$-semi-treillis complet d'ordre $\preceq_{aff}$).

Afin de prouver l'existence d'une structure de treillis, il est facilement vérifiable, à partir des propriétés de l'ordre $\preceq_{aff}$, que toute paire d'horloges $h_1, h_2 \in \mathcal{T}_{\hat{x}}^{\preceq_{aff}}$ admet une borne supérieure par rapport à l'ordre $\preceq_{aff}$. Nous motivons brièvement cette affirmation. Si h_1 et h_2 sont des horloges affines définies par les expressions $h_1 = [h]_{(\varphi_1, n_1)}$ et $h_2 = [h]_{(\varphi_2, n_2)}$, alors il existe une relation affine de paramètres $((n_1, \varphi_1), (n_2, \varphi_2))$ entre h_1 et h_2 (voir section 7.4.2.2 pour la définition d'une horloge affine $[h]_{(\varphi,n)}$, avec $\varphi \geq 0$ et $n \geq 1$, et sections 7.1.1 et 7.2.1 pour la définition d'une relation affine). Soit $\mathbf{P}$ un processus arbitraire dans lequel $\preceq_{aff} = \preceq_{aff}^{\mathbf{P}}$ et $h_1, h_2 \in \mathcal{T}_{\hat{x}}^{\preceq_{aff}}$. On a $(h_1, h_2) \in \mathcal{R}_{((n_1,\varphi_1),(n_2,\varphi_2))}^{\mathbf{P}}$. Or, conformément aux résultats obtenus aux sections 7.1.3, 7.2.2 et 7.2.3, $\mathcal{R}_{((n_1,\varphi_1),(n_2,\varphi_2))}^{\mathbf{P}} = \mathcal{R}_{(n_1,\varphi_{12},n_2)}^{\mathbf{P}}$, avec $\varphi_{12} = \varphi_2 - \varphi_1$, et $\mathcal{R}_{(n_1,\varphi_{12},n_2)}^{\mathbf{P}}$ admet une forme canonique que l'on note $\mathcal{R}_{(n,\varphi,d)}^{\mathbf{P}}$. Le triplet (n, φ, d) est minimal, par rapport à l'ordre $\leq_t$, parmi les triplets (n', φ', d') avec la propriété $\mathcal{R}_{(n,\varphi,d)}^{\mathbf{P}} = \mathcal{R}_{(n',\varphi',d')}^{\mathbf{P}}$. Conformément à l'implémentation proposée pour une relation affine en section 7.4.2.2, il existe h_u telle que $h_1 = [h_u]_{(max(0,-\varphi),n)}$ et $h_2 = [h_u]_{(max(0,\varphi),d)}$. Il résulte $h_1 \preceq_{aff} h_u$ et $h_2 \preceq_{aff} h_u$ et h_u est la plus petite horloge, par rapport à l'ordre $\preceq_{aff}$, avec la propriété considérée. Effectivement, si on considère $h'_u \preceq_{aff} h_u$ telle que $h_1 \preceq_{aff} h'_u$ et $h_2 \preceq_{aff} h'_u$, on a $h_1, h_2, h_u, h'_u \in \mathcal{T}_{h_u}^{\preceq_{aff}}$ et on peut démontrer, à partir des propriétés de l'ordre $\preceq_{aff}$ dans $\mathcal{T}_{h_u}^{\preceq_{aff}}$, que $h_u = h'_u$.

Dans le cas où les horloges h_1 et h_2 sont des expressions sur horloges affines dans $\mathcal{T}_{\hat{x}}^{\preceq_{aff}}$, i.e. $h_1 = E_1(\ldots, [h]_{(\varphi_i,n_i)}, \ldots)$ et $h_2 = E_2(\ldots, [h]_{(\xi_i,d_i)}, \ldots)$, avec $\varphi_i, \xi_i \geq 0$ et $n_i, d_i \geq 1$, on applique itérativement l'algorithme de construction de la borne supérieure de deux horloges affines comme décrit ci-dessus et on obtient la borne supérieure de h_1 et h_2. Cet algorithme peut se généraliser au cas de deux expressions E_1 et E_2 arbitraires appartenant à un même ensemble $\mathcal{T}_{\hat{x}}^{\preceq_{aff}}$. Finalement, si $h_1 \in \mathcal{T}_{\hat{x}}^{\preceq_{aff}}$ et $h_2 \in \mathcal{T}_{\hat{y}}^{\preceq_{aff}}$, alors $h_1 \leq h_2$ implique $h_1 \preceq_{aff} h_2$.

Étant donnée l'appartenance d'une horloge h à deux treillis différents $\mathcal{T}_{\hat{x}}^{\preceq_{aff}}$ et $\mathcal{T}_{\hat{y}}^{\preceq_{aff}}$ – condition (3), Définition 21 – un semi-treillis $\mathcal{T}_{\hat{x}}^{\preceq_{aff}}$ n'est pas toujours complet. La différence essentielle entre les ordres $\preceq$ et $\preceq_{aff}$ provient justement de cette condition (3) qui permet d'unifier des horloges appartenant à des semi-treillis $\mathcal{T}_{\hat{x}}^{\preceq_{aff}}$ et $\mathcal{T}_{\hat{y}}^{\preceq_{aff}}$ différents. Cette unification est motivée par les règles de synchronisabilité présentées en section 7.3.2.

Pour une variable x arbitraire, le semi-treillis $\mathcal{T}_{\hat{x}}^{\preceq_{aff}}$ est représenté par un *arbre affine* d'horloges (voir Figure 10.6) : chaque arc de l'arbre modélise la relation d'inclusion $\preceq_{aff}$ et chaque père représente une limite supérieure de tous ses fils. Par construction, on attribue à un arbre affine des propriétés similaires aux arbres booléens : 1) A chaque nœud est associé un ensemble d'horloges équivalentes entre elles et un représentant est choisi parmi ces horloges; 2) Le placement des formules d'horloges dans un arbre affine est réalisé afin que tout parcours descendant gauche-droite calcule les opérandes $\hat{x}_i$ avant l'expression $E(\ldots, \hat{x}_i, \ldots)$ qui les utilise.

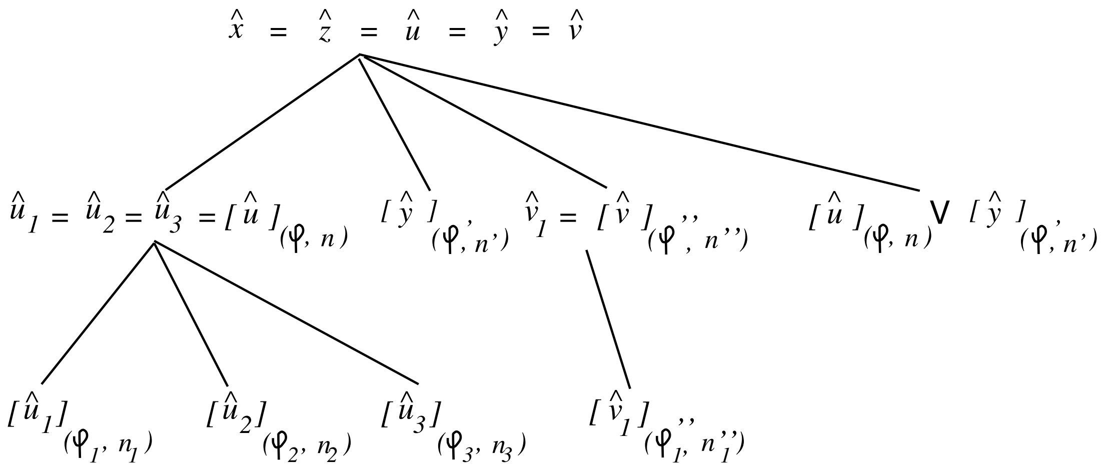

FIG. 10.6 – *Arbre affine.*

Le *graphe affine* est un graphe complètement connecté (i.e. il existe au moins un chemin entre chaque paire de nœuds du graphe) qui représente l'union de plusieurs semi-treillis $\mathcal{T}_{\hat{x}_i}^{\preceq_{aff}}$ qui partagent des éléments. Un graphe affine contient un ou plusieurs arbres affines. Nous avons représenté sur la Figure 10.7 un exemple de graphe affine.

Supposant que le système d'équations sur horloges associé à un programme après compilation contienne uniquement des horloges affines, nous proposons de résoudre ce système par triangularisation : les horloges du système triangularisé seront exprimées en fonction d'un vecteur de paramétrisation constitué d'horloges libres $\hat{x}$ ou d'éléments appartenant à des doublets $(\hat{x}, [x]_{(\varphi,n)})$. La mise en œuvre de la triangularisation dépends des propriétés des graphes affines ; elle se déroule en deux étapes principales : 1) orientation des équations associées à chaque racine d'un arbre affine élémentaire par désignation d'un représentant de la racine ; 2) orientation des équations (contraintes) externes qui définissent les représentants des racines des arbres affines élémentaires et, itérativement, les représentants des nœuds des *graphes affines* par : 2.1) fusion itérative d'arbres affines élémentaires et de *graphes affines* ; 2.2) preuve de *synchronisabilité* de formules d'horloges qui définissent la même horloge.

La deuxième étape constitue le cœur de la triangularisation et dépend directement des propriétés des graphes affines. Les éléments des arbres affines constituant ces graphes étant des expressions – unions, intersections et/ou différences – d'horloges en relations affines, toutes les propriétés des relations affines étudiées au chapitre 7 vont influencer l'algorithme de triangularisation.

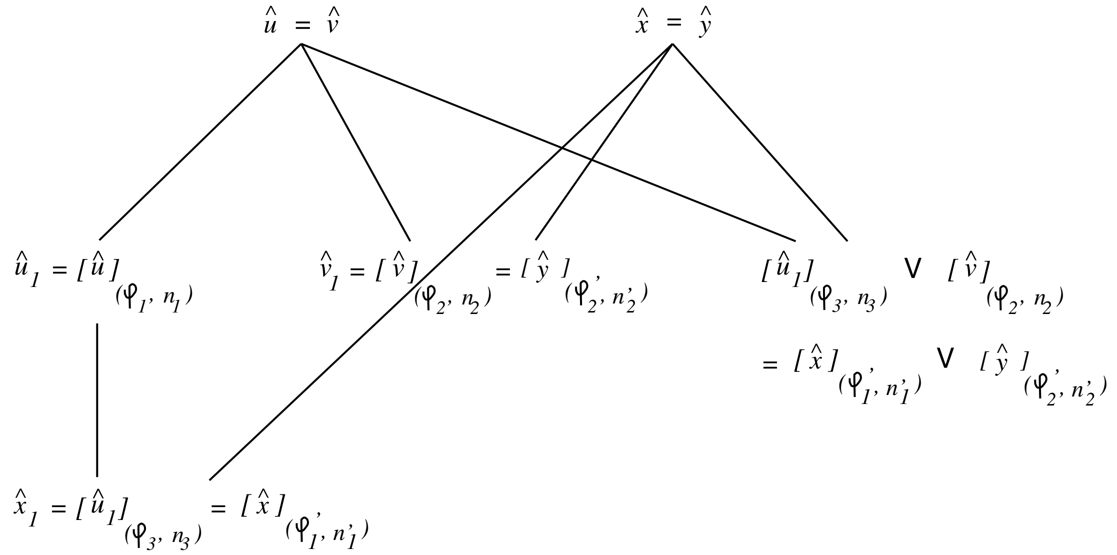

FIG. 10.7 – *Graphe affine.*

Nous avons divisé notre étude en deux parties :

1) En première instance, nous considérons que tout graphe affine se réduit à un arbre affine ; nous analysons dans ce cas l'influence des propriétés d'un arbre affine sur la *forme canonique arborescente*, la fusion d'arbres et la preuve d'*équivalence*[6] de formules d'horloges appartenant à un même arbre.

2) Nous étendons ensuite les concepts précédents au niveau des graphes affines ; nous étudions la fusion de graphes affines et la preuve de *synchronisabilité* de formules d'horloges d'un même graphe.

10.2.3 Arbres affines & Triangularisation

Dans cette section, nous considérons que deux treillis différents $\mathcal{T}_{\hat{x}}^{\preceq_{aff}}$ et $\mathcal{T}_{\hat{y}}^{\preceq_{aff}}$ ne partagent pas d'éléments. Cela veut dire que deux horloges appartenant à deux treillis différents ne peuvent pas être identifiées, donc le calcul affine utilise uniquement les règles de synchronisme déduites en section 7.3.1. Tout graphe affine se réduit à un arbre affine et la mise en œuvre de la triangularisation est réalisée par manipulation d'arbres affines. Essentiellement, il s'agit de : 1. Placement d'une formule d'horloges dans un arbre ; 2. Fusion d'arbres affines ; 3. Preuve d'équivalence de formules placées dans le même arbre.

Les arbres affines contiennent des horloges affines ou des expressions sur celles-ci. Étant donné un arbre affine T de racine r et une horloge h appartenant à T, on peut considérer que h se réduit à une horloge affine $[r]_{(\varphi,n)}$ ou une expression $E(\ldots,[r]_{(\varphi_i,n_i)},\ldots)$ sur horloges affines. Dans les deux cas précédents, on a $h \preceq_{aff} r$, conformément à la définition de l'ordre $\preceq_{aff}$ que nous avons introduit dans la section précédente. Si on note $\mathbf{P}$ le processus associé au programme SIGNAL en cours de compilation, $\preceq_{aff}$ représente la notation implicite pour l'ordre $\preceq_{aff}^{\mathbf{P}}$ défini dans la section 7.4.2.2 ; il existe en conséquence un ensemble de phases $\varphi_i \geq 0$ et périodes $d_i > 1$ telles que $r\ \mathcal{R}_t^{\mathbf{P}}\ h$ et

6. Dans le même arbre affine, la *synchronisabilité* est équivalente au *synchronisme* qui est une *équivalence*.

la relation $\mathcal{R}_t^{\mathbf{P}}$ est une expression – d'opérateurs union $\cup_r$, intersection $\cap_r$ et/ou différence $\setminus_r$ – sur des relations affines $\mathcal{R}^{\mathbf{P}}_{(1,\varphi_i,d_i)}$: $\mathcal{R}_t^{\mathbf{P}} = \mathcal{EXP}(\ldots, \mathcal{R}^{\mathbf{P}}_{(1,\varphi_i,d_i)}, \ldots)$. L'expression de la relation $\mathcal{R}_t^{\mathbf{P}}$ influence l'expression $E(\ldots, [r]_{(\varphi_i,n_i)}, \ldots)$ de définition de h. En conséquence, toutes les propriétés des relations $\mathcal{R}_t^{\mathbf{P}}$ étudiées en section 7.2 vont influencer le calcul d'horloges.

Nous étudions d'abord l'existence d'une forme canonique arborescente pour toute expression d'horloges appartenant à un arbre affine. Nous utilisons les résultats existants sur la forme normale relative d'une relation $\mathcal{R}_t^{\mathbf{P}}$ (voir sections 7.1.5.7 et 7.2.4.7). L'existence ou l'absence d'une forme canonique arborescente a des conséquences sur le placement d'une formule dans un arbre et sur les preuves d'équivalence de formules placées. Nous définissons ensuite le processus de fusion d'arbres affines qui est similaire, dans les points essentiels, au processus de fusion d'arbres booléens.

A partir de ces résultats, l'algorithme de triangularisation peut se résumer à une succession de pas comme suit :

1. Création de la forêt d'horloges initiale constituée d'un ensemble d'arbres affines élémentaires.

2. Fusion récursive des arbres de la forêt d'horloges tant qu'il existe des règles de réécriture applicables.

   2.1 Pour chaque arbre T' de la forêt d'horloges, on tente de réécrire la formule f de définition de sa racine r' ($r' = f$) afin que les opérandes se trouvent dans le même arbre T. En cas de succès, la racine r' peut être insérer dans T.

   2.2 Insertion de r' dans T (placement de f dans T) et fusion de T et T' pour obtenir T'' ; ces opérations sont décrites dans 10.2.3.1 et 10.2.3.2.

3. Test d'équivalence de formules d'horloges placées et, en cas de succès, unification de formules.

En pratique, les pas 2 et 3 ci-dessus ne sont pas strictement séquentiels. Le pas 2 correspond à l'orientation de l'équation $r' = f$ quand il n'existe pas d'autre définition de r'. Une fois la racine r' insérée dans T, toute autre équation $r' = f'$ doit être orientée en tenant compte du fait que r' est *placée*, c'est-à-dire elle possède déjà une définition f. Dans ce cas, l'orientation de $r' = f'$ nécessite la preuve de l'équivalence de f et f', ce qui fait l'objet du pas 3.

10.2.3.1 Forme canonique arborescente

La fusion des arbres affines T et T' peut être vue comme l'insertion de la racine r' de T', qui possède la formule de définition $formule(h_i)$, dans l'arbre T et l'union des sous-arbres A_1 et A_2 correspondant respectivement à $formule(h_i)$ et r' (en section 10.1.3 nous n'avons pas détaillé l'étape d'insertion de la racine r'). Les Figures 10.3 et 10.4 peuvent servir de support pour illustrer la fusion ; le principe de l'algorithme reste le même, seules l'insertion de r' et l'union de A_1 et A_2 doivent être adaptés en fonction des propriétés des arbres affines. On considère que le système d'équations sur horloges a été transformé de telle sorte que dans toute équation $r' = formule(h_i)$, $formule(h_i)$ contient au plus deux horloges : $formule(h_i) = h_1 \; op \; h_2$ ou $formule(h_i) = h_1$, avec $op \in \{\vee, \wedge, \setminus\}$ et les horloges h_1, h_2 placées dans l'arbre affine T de racine r.

L'insertion de r' dans T consiste en :

– Si $formule(h_i) \notin T$:

  – Placement de $formule(h_i)$ dans T par création d'un nœud $\eta \in T$ caractérisé par la formule d'horloges $formule(h_i)$;

- Insertion de r' dans l'ensemble de formules d'horloges associées à η (cet ensemble devient $\{r', formule(h_i)\}$) et r' est choisi comme représentant de η.
  Toutes les contraintes externes associées à r' restent valides.

- Si $formule(h_i) \in T$, soit η le nœud de T qui lui correspond et r le représentant de η. Les étapes sont :
  - Insertion de r' dans l'ensemble de formules d'horloges associées à η et r' est choisi comme représentant de η.
  - L'union des ensembles de contraintes externes sur r et r' devient le nouvel ensemble de contraintes externes associées à r'.

Il est intéressant d'étudier dans quelle mesure on peut définir un placement canonique de $formule(h_i) = h_1 \ op \ h_2$ dans T.

D'abord, h_1 et h_2 ne sont pas forcément des horloges affines, et cela parce qu'une formule $h_i \ op \ h_j$ sur horloges affines ne se réduit pas toujours à une horloge affine. Cette situation est la conséquence directe d'un résultat similaire constaté au niveau des relations affines : la formule $\mathcal{R}^{\mathbf{P}}_{(1,\varphi_i,d_i)} \ op_r \ \mathcal{R}^{\mathbf{P}}_{(1,\varphi_j,d_j)}$ ne se réduit pas toujours à une relation affine de type $\mathcal{R}^{\mathbf{P}}_{(1,\varphi,d)}$ (voir section 7.1.5.7). Si $h_i \ op \ h_j$ n'est pas une horloge affine, il est nécessaire que h_i et h_j soient calculées avant $h_i \ op \ h_j$ (contrainte de triangularité du système d'équations). En conséquence, le placement de l'horloge $h_i \ op \ h_j$ dans un arbre affine peut être réalisé comme pour les expressions booléennes d'horloges : $h_i \ op \ h_j$ sera placée sous le plus petit, par rapport à l'ordre $\preceq_{aff}$, ancêtre commun de h_i et h_j.

Afin d'éviter le placement différent de formules d'horloges équivalentes – qui peut entraîner une explosion combinatoire du nombre d'horloges d'un arbre affine et qui peut rejeter des contraintes de synchronisation sur des horloges équivalentes – il est intéressant d'étudier le problème d'une *forme canonique* des expressions sur horloges appartenant à un arbre affine. Nous sous-entendons par "formules équivalentes" deux formules synchrones (la relation de synchronisme est une équivalence). Les règles de synchronisme obtenues en section 7.3.1 deviennent ainsi des propriétés d'équivalence de formules d'horloges. Conformément à la règle no. 7, Théorème 10, on peut tester l'équivalence des horloges h et h' définies par les expressions suivantes

$$h = E_1(h_1, \ldots, h_p), \quad h_i = [r]_{(\varphi_i,d_i)}$$
$$h' = E_2(h'_1, \ldots, h'_q), \quad h'_j = [r]_{(\varphi'_j,d'_j)}$$

si E_1, E_2 peuvent être mises sous la forme d'unions d'horloges affines de mêmes périodes, i.e.

$$h = (\vee)_{\varphi \in I_1}[r]_{(\varphi,n)}$$
$$h' = (\vee)_{\varphi \in I_2}[r]_{(\varphi,n)}$$

Dans ce cas $h \hat{=} h'$ si et seulement si $I_1 = I_2$.

On recherche donc une forme canonique pour une expression sur horloges $h = E_1(h_1, \ldots, h_p)$, avec $h_i = [r]_{(\varphi_i,d_i)}$, sous la forme d'une union d'horloges affines. En section 7.2.4.7, Proposition 12, il a été démontré que toute relation $\mathcal{R}^{\mathbf{P}}_t = \mathcal{EXP}(\mathcal{R}^{\mathbf{P}}_{(1,\varphi_1,d_1)}, \ldots, \mathcal{R}^{\mathbf{P}}_{(1,\varphi_p,d_p)})$ peut être mise sous la forme d'une union de relations affines $\mathcal{R}^{\mathbf{P}}_t = (\cup_r)_{\varphi \in Exp(I_1,\ldots,I_p)} \mathcal{R}^{\mathbf{P}}_{(1,\varphi,n)}$, avec n un multiple commun de d_i, $\forall i = \overline{1,p}$, et I_i correspondant aux décompositions de $\mathcal{R}^{\mathbf{P}}_{(1,\varphi_i,d_i)}$, $\forall i = \overline{1,p}$: $\mathcal{R}^{\mathbf{P}}_{(1,\varphi_i,d_i)} = (\cup_r)_{\varphi \in I_i} \mathcal{R}^{\mathbf{P}}_{(1,\varphi,n)}$. A partir de cela, on obtient pour h une décomposition de la forme

$$h = (\vee)_{\varphi \in I}[r]_{(\varphi,n)}$$

si toute horloge $h_i = [r]_{(\varphi_i, d_i)}$ se décompose en $h_i = (\vee)_{\varphi \in I_i}[r]_{(\varphi, n)}$, avec $I_i = \{\varphi_i, \varphi_i + d_i, \ldots, \varphi_i + (\frac{n}{d_i} - 1)d_i\}$ et $I = Exp_1(I_1, \ldots, I_p)$ (Exp_1 est l'expression sur ensembles correspondant à l'expression E_1 sur horloges).

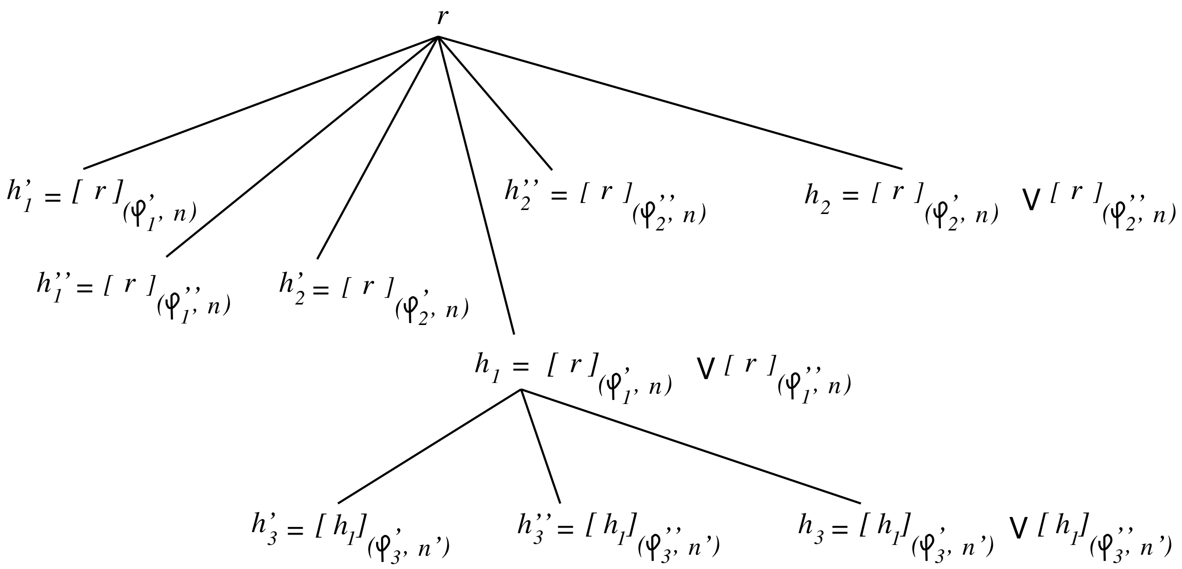

FIG. 10.8 – *Illustration de la FNR dans un arbre affine.*

La décomposition de $h = E_1(h_1, \ldots, h_p)$, avec $h_i = [r]_{(\varphi_i, d_i)}$, sous la forme $h = (\vee)_{\varphi \in I}[r]_{(\varphi, n)}$, n'est pas unique : les paramètres n et $I \subseteq \mathbb{N}$ dépendent d'une part de la forme des horloges h_i qui interviennent dans l'expression E_1, d'autre part du contexte d'utilisation de E_1. Des décompositions différentes peuvent être construites en fonction du contexte d'utilisation. S'il s'agit uniquement de comparer deux expressions d'horloges h et h', il suffit de choisir n comme le plus petit multiple commun (ppcm) de d_i, avec $i = \overline{1, p}$, et d'_j, avec $j = \overline{1, q}$. Par contre, n devrait changer de valeur pour permettre de comparer h et h'', respectivement h' et h'', avec h'' une expression sur horloges affines de phase φ''_k et périodes d''_k sur r :

$$h'' = E_3(h''_1, \ldots, h''_m), \quad h''_l = [r]_{(\varphi''_l, d''_l)}$$

Si on considère un ensemble $\mathcal{E}$ d'expressions sur des horloges affines de la forme $[r]_{(\varphi, d)}$, toute expression $E \in \mathcal{E}$ peut s'écrire sous une forme unique

$$E(\ldots, h_i, \ldots) = (\vee)_{\varphi \in I}[r]_{(\varphi, n)} \tag{10.1}$$

si *n est le ppcm de toutes les périodes d_i, pour toute horloge affine $[r]_{(\varphi_i, d_i)}$ appartenant à $\mathcal{E}$* (pour n fixé, I est ensuite uniquement déterminé en fonction de n et de l'expression E). Nous préférons dans ce cas parler d'une *forme normale relative* des expressions de $\mathcal{E}$, puisqu'elle est dépendante du contexte de définition, i.e. l'ensemble $\mathcal{E}$.

Sur la Figure 10.8, les horloges des sous-arbres $\mathcal{E}_1 = \{h'_1, h''_1, h_1, h'_2, h''_2, h_2\}$ et $\mathcal{E}_2 = \{h_1, h'_3, h''_3, h_3\}$ sont en forme normale relative, respectivement à $\mathcal{E}_1$ définie par n et à $\mathcal{E}_2$ définie par n' (pour l'horloge h, sa forme normale relative à $\mathcal{E}$ définie par n est notée $FNR^n_{\mathcal{E}}(h)$). Afin de construire la FNR de l'arbre entier, il est nécessaire de réécrire les expressions des horloges h'_3, h''_3 et h_3 pour qu'elles contiennent uniquement des horloges affines de la forme $[r]_{(\varphi_i, d_i)}$ et de calculer une valeur n'' qui va dépendre des expressions d'horloges de l'arbre entier et qui remplacera n et n'.

On peut ainsi constater que la mise en place d'une FNR dans l'arbre affine nécessite :

– Premièrement, une vue d'ensemble de toutes les horloges appartenant à l'arbre – vue dont on ne peut disposer qu'au moment où l'arbre est disponible dans sa totalité, donc en fin de construction – imposée par la nécessité de connaître n dès le début du placement des formules d'horloges dans l'arbre;

– Deuxièmement, la réécriture de chaque horloge comme une expression incluant uniquement des sous-échantillonnages affines sur la racine r de l'arbre.

Le deuxième aspect ci-dessus nécessite l'étude de techniques de réécriture basées sur les propriétés des expressions manipulées et la définition d'algorithmes d'implémentation appropriés. En ce qui concerne le premier problème, on peut envisager :

– Soit de modifier la FNR de toute horloge placée dans un arbre à l'ajout d'une nouvelle horloge affine. Ceci a comme inconvénient évident la nécessité de parcourir la liste de toutes les horloges déjà construites à l'ajout d'une nouvelle horloge.

– Soit de proposer une valeur pour n au début du calcul d'horloges ; il est ainsi possible de calculer la FNR de toute horloge au moment de son placement dans l'arbre affine, sans avoir à parcourir la liste de toutes les horloges déjà placées.

Dans les deux cas précédents, le calcul des ensembles finis $I \subseteq \mathbb{N}$ impose la manipulation d'ensembles discrets d'entiers et des opérations sur ces ensembles.

La définition d'une forme canonique dans un arbre affine est un problème complexe dû essentiellement à l'absence d'une décomposition unique d'une formule d'horloges sous la forme d'une union finie d'horloges affines de même période. Dans l'absence d'une forme canonique, toute formule h_1 *op* h_2 sera réécrite, quand c'est possible, sous la forme $[h]_{(\varphi,d)}$ (en utilisant les règles no. 4-5-6, section 7.3.1) et sera placée comme fille de h. Si h_1 *op* h_2 ne peut pas être réduite à une horloge affine, elle sera placée comme fille du plus petit ancêtre commun de h_1 et h_2, par rapport à l'ordre $\preceq_{aff}$.

10.2.3.2 Fusion d'arbres affines

Les Figures 10.3 et 10.4 de la section 10.1.3 présentant la fusion des arbres booléens T et T' pour produire T'' peuvent constituer également l'illustration du processus de fusion d'arbres affines ; la seule différence apparaît au niveau de l'union des arbres A_1 et A_2. Dans le cas des arbres booléens, A_1 et A_2 sont toujours disjoints et leur union et la même chose que leur juxtaposition (union disjointe). Dans le cas affine, A_1 et A_2 peuvent contenir des horloges égales et on peut opter pour l'union ou l'union disjointe de A_1 et A_2. Dans le premier cas, l'unification des horloges égales de A_1 et A_2 peut entraîner la réécriture de formules déjà placées dans A_2, ce qui revient à relancer le calcul d'horloges sur ces formules, après les avoir enlevées de l'arbre affine au préalable. Dans le deuxième cas, on peut être confrontés au problème de la duplication des horloges et, dans le pire des cas, d'une explosion du nombre d'horloges placées dans un arbre affine. Cette solution reste néanmoins la plus intéressante du point de vue de la vitesse de compilation.

10.2.3.3 Équivalence de formules placées

La fusion des arbres affines T et T' dans T'' (voir Figures 10.3 et 10.4) place la racine r' dans T'' comme représentant d'un nœud auquel est associé un ensemble d'horloges équivalentes (il s'agit de la formule f et d'un certain nombre d'horloges libres). Le représentant r' peut être ensuite contraint par un ensemble d'équations externes de la forme $r' = f'$. Conformément au principe de la triangularisation, une équation de type $r' = f'$ doit être orientée de droite à gauche. Dans ce cas, la formule f' ne peut pas constituer une définition pour r' puisque r' est déjà définie par f. Il est donc nécessaire de prouver l'équivalence de f et f'. Si f' peut être réécrite pour qu'elle appartienne T'', on peut appliquer les pas suivants pour vérifier l'équivalence de f et f' :

1. S'il existe une horloge $k \in T''$ telle que $f = [k]_{(\varphi,d)}$ et $f' = [k]_{(\varphi',d')}$, alors :

$$f = f' \Leftrightarrow \begin{cases} \varphi = \varphi \\ d = d' \end{cases}$$

2. Sinon :

   2.1 Si $f = E_1(h_1, \ldots, h_p)$, $f' = E_2(h'_1, \ldots, h'_p)$ et le père k de h_i, h'_j dans l'arbre T'' vérifie $h_i = [k]_{(\varphi_i,d_i)}$ et $h'_i = [k]_{(\varphi'_i,d'_i)}$, alors

$$h_i = h'_i,\ \forall i = \overline{1,p} \Rightarrow f = f'$$

   2.2 Sinon, si f et f' se réécrivent en :

$$\begin{array}{l} f = E_1([k]_{(\varphi_1,d_1)}, \ldots, [k]_{(\varphi_p,d_p)}) \\ f' = E_2([k]_{(\varphi'_1,d'_1)}, \ldots, [k]_{(\varphi'_q,d'_q)}) \end{array}$$

   alors

$$f = f' \quad \text{ssi} \quad FNR^n_{\mathcal{E}}(f) = FNR^n_{\mathcal{E}}(f')$$

   avec $n = ppcm(d_1, \ldots, d_p, d'_1, \ldots, d'_q)$ et $\mathcal{E} = \{[k]_{(\varphi_i,d_i)}, [k]_{(\varphi'_j,d'_j)}, f, f'\}$.

   2.3 Sinon, f et f' ne sont pas équivalentes.

Si les formules f et f' sont des horloges affines (pas 1.), on peut vérifier leur équivalence. Or, dans la majorité des cas f et f' ne sont pas des horloges affines et on ne dispose pas d'une forme canonique arborescente pour toute expression sur horloges affines. Dans ce cas, on peut soit développer leurs formes normales relatives et tester leur égalité (pas 2.2), soit comparer f et f' terme à terme sur leur expression de définition (pas 2.1), soit décider qu'on ne sait pas vérifier l'équivalence de f et f' et dans ce cas la réponse du compilateur sera "f et f' ne peuvent pas être démontrées équivalentes" (pas 2.3).

Dans le cas où la formule f' n'appartient pas à T'', f et f' ne peuvent pas être démontrées équivalentes (conformément à l'hypothèse, adopté en début de section 10.2.3, selon laquelle des $\vee$-semi-treillis affines différents ne partagent pas d'éléments).

10.2.4 Graphes affines & Triangularisation

Dans la section précédente, nous avons présenté le calcul affine d'horloges dans l'hypothèse où tout graphe affine se réduit à un arbre affine. Dans ce cas, le calcul affine était basé uniquement sur les règles de synchronisme déduites en section 7.3.1 ; les règles de synchronisabilité générales de la section 7.3.2 n'ont pas été utilisées. Dans ce contexte, l'algorithme de résolution d'un système d'équations sur expressions d'horloges affines consistait dans une triangularisation du système en tirant parti des propriétés des arbres affines. Nous avons pu constater l'influence des règles de synchronisme sur ces propriétés, essentiellement au niveau d'une forme canonique arborescente et de la fusion d'arbres affines.

Nous ajoutons les règles de synchronisabilité générales de la section 7.3.2 au système de règles de synchronisme mises à la disposition du calcul affine d'horloges et analysons leur influence sur la triangularisation d'un système d'équations sur expressions d'horloges affines. La triangularisation consiste maintenant d'une suite d'opérations sur des *graphes affines* et dépend directement de leurs propriétés.

Le graphe affine est un graphe acyclique complètement connecté dans lequel chaque nœud : 1) peut avoir 0, 1 ou plusieurs pères ; 2) représente au moins une expression sur horloges affines. Nous avons représenté en Figure 10.7 un exemple de graphe affine.

De manière standard, un arbre est un graphe acyclique dans lequel chaque nœud peut avoir au plus un père. Pour un arbre affine, on ajoute à chaque nœud un ensemble d'expressions sur horloges affines. Un graphe affine peut être vu comme une union finie d'arbres affines qui garde toutes les propriétés des arbres affines présenté précédemment.

Dans ces conditions, nous proposons un algorithme de triangularisation en trois étapes :

1. Construction des graphes affines élémentaires.

2. Fusion des graphes affines élémentaires et, itérativement, des graphes affines jusqu'à ce que les conditions de l'application de l'algorithme de fusion ne soient plus vérifiées.

3. Test de synchronisabilité de formules placées dans les graphes affines et, en cas de succès, unification des formules.

10.2.4.1 Création de graphes affines élémentaires

Tout *graphe affine élémentaire* se réduit à un arbre affine élémentaire (voir Figure 10.5, section 10.2.2, pour un exemple). A chaque racine d'un arbre affine élémentaire est associé un ensemble d'équations sur horloges qui appartiennent au système d'équations initial et qui sont orientées par le choix d'un représentant. Toutes les horloges différentes du représentant seront définies en fonction de ce dernier. Le représentant peut être contraint par un ensemble d'équations externes qui seront orientées à l'étape suivante (fusions de graphes affines) de l'algorithme de triangularisation. Les autres nœuds de l'arbre (différents de la racine) ne sont pas contraints par d'autres équations.

Si la première étape de l'algorithme de triangularisation (section 10.2.4) est quasiment indépendante, les étapes 2. et 3. peuvent être considérées comme s'exécutant en parallèle. Ces étapes correspondent à l'orientation des équations (contraintes) externes qui ont comme membre gauche les représentants des nœuds racines. Nous présentons les deux étapes de manière indépendante afin de dégager les principaux concepts.

10.2.4.2 Fusion de graphes affines

L'algorithme de fusion, lancé sur l'ensemble de graphes affines élémentaires construit à la première étape, est réitéré sur des graphes affines généraux tant qu'il existe de règles de réécriture applicables. Cet algorithme peut être résumé comme suit :

> Pour chaque graphe G' de l'ensemble de graphes affines existants à un pas de l'itération et pour chaque racine r' d'un arbre affine appartenant à G', si r' est contrainte par l'équation

externe $r' = f$ [7], alors :

- On tente de réécrire f pour que ses opérandes se trouvent dans le même arbre T d'un graphe affine G. En cas de succès, la racine r' peut être insérer dans G.
- On insère r' en G et on fusionne G et G' pour obtenir G''. Les racines non placées de G' se trouvent maintenant dans le graphe G'' ; l'algorithme de fusion est relancé sur les racines des arbres affines de G''.

L'insertion de r' en G implique l'insertion de r' dans tous les arbres $T_i \in G$ qui contiennent les opérandes de f – conformément à la procédure d'insertion d'une racine r' dans un arbre affine décrite en section 10.2.3.1 – et l'unification des nœuds $\eta_i \in T_i$ ainsi obtenus. Nous décrivons la procédure d'unification dans la section suivante.

Après insertion de r' dans G, la fusion des graphes G et G' est réalisée par union des sous-arbres affines A_1 et A_2 correspondant à r' dans G et à f dans G'. Nous avons décrit ce processus, et les problèmes qu'il implique, en section 10.2.3.2.

10.2.4.3 Synchronisabilité de formules

La fusion des graphes affines G et G' place la racine r' dans G'' comme représentant d'un nœud auquel est associé un ensemble d'horloges équivalentes (il s'agit de la formule f et d'un certain nombre d'horloges libres). Le représentant r' peut être contraint ensuite par un ensemble d'équations externes de la forme $r' = f'$ qui doivent être orientées de la même manière que $r' = f$. Nous parlons dans ce cas de l'orientation d'une équation associée à une formule placée.

La formule f' ne peut plus définir l'horloge représentant r', puisque celle-ci est déjà définie par f. Les horloges f et f' doivent être prouvées *synchronisables*. C'est maintenant que interviennent les règles de synchronisabilité générales démontrées au chapitre précédent [8]. Nous identifions trois cas :

1) La formule f' peut être réécrite pour que f et f' se trouvent dans le même arbre. Dans ce cas, le test de synchronisabilité de f et f' se réduit à la vérification de leur équivalence comme formules placées dans le même arbre, conformément au pas 1. et 2. de l'algorithme présenté en section 10.2.3.3.

2) La formule f' peut être réécrite pour que f et f' se trouvent dans le même graphe, mais dans des arbres différents. Dans ce cas, il est nécessaire de développer des techniques de recherche dans des graphes affines qui permettent l'application des règles de synchronisabilité générales proposées dans la deuxième partie de ce document. Ces techniques doivent permettre, entre autres : le calcul de tous les chemins existants entre deux formules d'horloges f et f' ; la caractérisation des relations existantes entre f, respectivement f' et un nœud sur un chemin entre f et f'. Dans le cas où ces relations sont affines ou des compositions de relations affines, nous disposons d'un ensemble minimal de règles de synchronisabilité suffisant pour traiter tous les cas des situations possibles.

7. Comme dans tous les cas précédents, f est une formule sur horloges affines de la forme h_1 *op* h_2 ou une horloge affine h. On considère que toutes les formules d'horloges ont été réécrites afin qu'elles contiennent au plus deux opérandes.

8. Tous les processus $\mathbf{Q}$ dans lesquels on applique les règles de synchronisabilité déduites en section 7.3 satisfont : $\mathbf{Q} \in \mathcal{P}_{A_2}$, $A_1 \subseteq A_2$ et $\Pi_{A_1}(\mathbf{Q}) \subseteq \mathbf{P}$, où $\mathbf{P} \in \mathcal{P}_{A_1}$ est un processus arbitraire dans lequel les règles de synchronisabilité de la section 7.3 ont été prouvées valides.

3) Les formules f et f' se trouvent dans des graphes différents. Dans ce cas, f et f' sont synchronisables si et seulement si elles sont affines, i.e. il existe k, k' telles que $f = [k]_{(\varphi,d)}$ et $f' = [k']_{(\varphi',d')}$ (conformément aux règles no. 8 et 8' présentées en section 7.3.2).

Dans le cas où le test de synchronisabilité de f et f' réussit, il est nécessaire de réaliser l'unification des nœuds η et η' correspondant à f et f'. Ceci implique en première instance l'union des classes de formules associées à η et η' avec le choix d'un représentant arbitraire parmi les représentants r et r' de η et η'. Ensuite, toutes les formules qui utilisent f' doivent être réécrites en remplaçant f' par f. Cette réécriture se déroule de manière différente d'après que f et f' se trouvent ou non dans le même arbre. Si les deux formules se trouvent dans le même arbre, la réécriture des formules qui utilisent f' correspond à relancer le calcul d'horloges sur celles-ci (enlèvement des formules de l'arbre et replacement par fusion d'arbres). Dans le cas où les formules f et f' appartiennent à des arbres différents, leur unification implique aussi l'appartenance de f' à l'arbre qui contient f, et cette appartenance doit être validée ensuite pour toutes les formules sur f' pour lesquelles elle devient effective.

10.2.5 Conclusion

L'idée du calcul affine d'horloges est non seulement de triangulariser le système d'équations initial afin de fournir à chaque horloge une définition unique, mais de synthétiser en même temps une horloge *maîtresse* unique qui cadence l'exécution du programme et à partir de laquelle toutes les autres horloges puissent être calculées pour la simulation fonctionnelle. Dans le cas où, à la fin du calcul d'horloges, toutes les horloges ont été englobées dans un graphe unique et toutes les contraintes de synchronisabilité ont été prouvées valides, on considère que le programme est correct du point de vue de ses contraintes de synchronisation et qu'il peut être exécuté (extension de la notion de "programme endochrone" au cas de plusieurs arbres appartenant au même graphe). Si le graphe unique se réduit à un arbre, la racine de l'arbre représente l'horloge maîtresse du programme. Autrement, la racine d'un arbre affine située "au niveau le plus haut" dans le graphe peut être considérée comme horloge maîtresse et un stratégie de génération de code peut être élaborée afin de calculer toutes les horloges du graphe à partir de celle-ci. Les horloges du graphe pourraient s'exprimer comme de sous-échantillonnages et sur-échantillonnages affines, et des expressions sur ceux-ci, à partir de l'horloge maîtresse, et des techniques de marquage d'horloges devraient être envisagées afin de ne calculer qu'une seule fois les horloges appartenant à plusieurs arbres du graphe.

Si le calcul d'horloges produit plusieurs graphes affines, le programme est exochrone et non complètement spécifié du point de vue des contraintes de synchronisation. Dans ce cas, des contraintes supplémentaires sont nécessaires en vue d'une simulation fonctionnelle.

10.3 Intégration des calculs affine & booléen sur horloges

Nous avons défini le calcul affine d'horloges comme la méthode de résolution d'un système d'équations $\mathcal{S}$ contenant uniquement des expressions sur des horloges affines. Dans ce contexte, l'idée la plus naturelle a été de vérifier dans quelle mesure les principes du calcul booléen d'horloges peuvent être repris dans le nouveau calcul affine. Nous avons proposer de triangulariser le système $\mathcal{S}$ en fonction d'une paramétrisation choisie parmi les horloges du système, et ceci pour deux raisons principales : 1) trouver l'ensemble des solutions du système est un problème complexe ; 2) l'espace des expressions sur horloges affines a une structure similaire à celui des expressions sur sous-échantillonnages par conditions booléennes. Concernant le dernier aspect, les propriétés des relations affines ont permis de définir sur l'ensemble des horloges affines un treillis d'ordre $\preceq_{aff}$. Nous avons introduit le graphe affine comme structure de représentation d'un système triangulaire d'après le modèle de l'arbre booléen existant. Nous avons analysé les principales propriétés des graphes affines – déduites directement de celles de l'ordre $\preceq_{aff}$ – et leur influence sur le processus de triangularisation. Dans le cas où tout graphe affine pourrait se réduire à un arbre affine, nous avons constaté des similarités évidentes avec le calcul booléen concernant la fusion d'arbres, les preuves d'équivalence de formules d'horloges et la nécessité d'une forme canonique arborescente. Autrement, dans le cas général, nous étions confrontés au problème de la manipulation de graphes plus complexes que les arbres. Nous avons décomposé le problème, en partant de l'observation que tout graphe est une union d'arbres et afin de retrouver les résultats disponibles au niveau des arbres affines, et avons esquissé les principes de base des mécanismes envisagés.

Dans cette section, nous proposons une méthode d'intégration des calculs affine et booléens sur laquelle nous nous sommes basé dans la mise en œuvre d'un prototype de test de nos résultats formels. D'abord, nous définissons la relation r_{mix} comme l'union des ordres partiels $\preceq$ et $\preceq_{aff}$ et introduisons le *graphe mixte* comme la structure de représentation de r_{mix}. Tout graphe mixte sera un graphe acyclique complètement connecté formé d'un ensemble d'arbres booléens et de graphes affines.

Nous définissons ensuite les principes de base du calcul *mixte* d'horloges qui représente la méthode de résolution d'un système d'équations impliquant des horloges sous-échantillonnages par conditions booléennes et des horloges affines. Finalement, nous concluons sur la mise en œuvre d'un prototype simple de test des principaux résultats précédents et sur les problèmes rencontrés.

10.3.1 Graphe mixte

Nous définissons la relation r_{mix} comme l'union des ordres partiels $\preceq$ et $\preceq_{aff}$:

Définition 22 *Pour toute $\cup_{i\in I}\mathcal{T}_{\hat{x}_i}^{\preceq_{aff}}$ union finie de $\vee$-semi-treillis d'ordre $\preceq_{aff}$ et toute $\cup_{j\in J}\mathcal{T}_{\hat{y}_j}^{\preceq}$ union finie de $\vee$-semi-treillis d'ordre $\preceq$, on définit sur l'ensemble d'horloges $\mathcal{G}^{r_{mix}}$ donné par*

$$\mathcal{G}^{r_{mix}} = (\ \cup_{i\in I}\mathcal{T}_{\hat{x}_i}^{\preceq_{aff}}\ ) \cup (\ \cup_{j\in J}\mathcal{T}_{\hat{y}_j}^{\preceq}\ )$$

la relation r_{mix} de la manière suivante :

$$h_1 r_{mix}\ h_2 \quad \Leftrightarrow \quad h_1 \preceq\ h_2 \ \textbf{ou}\ h_1 \preceq_{aff}\ h_2$$

□

Dans la définition ci-dessus, $\cup_{i\in I}\mathcal{T}_{\hat{x}_i}^{\preceq_{aff}}$ ou $\cup_{j\in J}\mathcal{T}_{\hat{y}_j}^{\preceq}$ peuvent être vides ; dans ce cas $\mathcal{G}^{r_{mix}}$ se réduit à une union de semi-treillis d'ordre $\preceq_{aff}$ (si $\cup_{j\in J}\mathcal{T}_{\hat{y}_j}^{\preceq} = \emptyset$) ou à une union de semi-treillis d'ordre $\preceq$ (si $\cup_{j\in J}\mathcal{T}_{\hat{y}_j}^{\preceq_{aff}} = \emptyset$).

La relation r_{mix} englobe également les propriétés de la structure de treillis d'ordre $\preceq$, qui contient l'ensemble des expressions construites sur des horloges sous-échantillonnages par conditions booléennes, et les propriétés du treillis d'ordre $\preceq_{aff}$, qui comprend les expressions sur horloges affines. Nous disposons ainsi d'un cadre unitaire pour la résolution d'un système mixte d'équations sur horloges. La méthode de résolution choisie consiste dans la triangularisation du système d'équations en utilisant les caractéristiques de la relation r_{mix}. Nous avons opté pour cette solution par plusieurs raisons ; parmi ceux-ci : 1) conformité avec la méthode de résolution implémentée dans le calcul booléen existant ; 2) facilité d'intégration des calculs affine et booléen étant données leurs propriétés similaires. Le système triangulaire final sera représenté sous la forme d'un ou de plusieurs *graphes mixtes* ; chaque graphe mixte implémente un ensemble $\mathcal{G}^{r_{mix}}$ et possède la propriété d'être complètement connecté.

Dans la section suivante, nous esquissons les principes du calcul *mixte* d'horloges qui consiste essentiellement de la fusion itérative de graphes mixtes et la preuve d'équivalence de formules placées dans le même graphe mixte. Les algorithmes existants dans le calcul booléen d'une part et ceux proposés pour le calcul affine d'autre part vont concourir à la mise en place du calcul mixte. En première instance nous exposons les principes, ensuite nous traitons de la mise en œuvre proposée.

10.3.2 Calcul mixte d'horloges

Le calcul mixte d'horloges représente la résolution d'un système mixte d'équations sur horloges, c'est-à-dire d'un système formé d'expressions sur horloges affines ou sous-échantillonnages par conditions booléennes. Résoudre le système revient à construire un système triangulaire équivalent, dans lequel chaque horloge est définie en fonction d'un sous-ensemble de l'ensemble total des horloges : horloges affines ou sous-échantillonnages par conditions booléennes. Cet ensemble d'horloges forment le vecteur de paramétrisation. Chaque équation du système est orientée de droite à gauche afin que l'expression de droite devienne la définition de l'horloge de gauche. Quand l'orientation n'est pas possible parce que l'horloge de la partie gauche possède déjà une définition, l'expression de droite doit être prouvée *synchronisable* avec la définition existante. Au cours du processus de triangularisation, nous utilisons les propriétés de la relation r_{mix} et nous manipulons des graphes mixtes qui sont des unions d'arbres booléens et de graphes affines. Comme dans le cas du calcul affine d'horloges (section 10.2), la triangularisation peut être résumée dans les trois étapes suivantes :

1. Construction des graphes mixtes élémentaires.

2. Fusion des graphes mixtes élémentaires et, itérativement, des graphes mixtes jusqu'à ce que les conditions de l'application de l'algorithme de fusion ne soient plus vérifiées.

3. Test de *synchronisabilité* de formules placées dans les graphes mixtes et, en cas de succès, unification des formules.

10.3.2.1 Construction de graphes mixtes élémentaires

Tout *graphe mixte élémentaire* se réduit à un arbre affine élémentaire (voir Figure 10.5, section 10.2.2) ou à un arbre de partition (voir Figure 10.1, section 10.1.1). A chaque racine d'un arbre affine élémentaire ou de partition est associé un ensemble d'équations sur horloges qui appartiennent au système d'équations initial et qui sont orientées par le choix d'un représentant. Toutes les horloges différentes du représentant seront définies en fonction de ce dernier. Le représentant peut être contraint par un ensemble d'équations externes qui seront orientées à l'étape suivante (fusions de graphes mixtes) de l'algorithme de triangularisation. Les autres nœuds de l'arbre (différents de la racine) ne sont pas contraints par d'autres équations.

10.3.2.2 Fusion de graphes mixtes

La construction du système triangulaire d'équations sur horloges poursuit avec l'orientation des équations (contraintes) externes qui ont comme parties gauches les représentants des racines des arbres affines élémentaires ou arbres de partition (que l'on a regroupé sous la dénomination de graphes mixtes élémentaires) construits en première étape. Ceci se réalise par un processus de fusion de graphes lancé initialement sur l'ensemble de graphes mixtes élémentaires et réitéré sur les graphes mixtes générés par la suite, tant qu'il existe des règles de réécriture applicables. L'algorithme de fusion peut être résumé comme suit : pour un graphe mixte G' construit à un pas de l'itération et pour une racine r' d'un arbre appartenant à G', l'équation $r' = f$ est orientée de droite à gauche, telle que f devienne la définition de r', en deux pas :

- Premièrement, on tente de réécrire f pour que ses opérandes[9] se trouvent dans le même arbre T appartenant au graphe G. En cas de succès, r' peut être insérer dans G et on passe à l'étape suivante. Autrement, r' ne peut pas être insérer dans un graphe G. Dans les deux cas, la formule f représente la définition de r'.

- S'il existe $T \in G$ tel que $f \in T$, alors r' est insérée en G et les graphes G et G' fusionnent pour produire G''.

Analysons maintenant l'algorithme présenté ci-dessus. On remarque d'abord que, afin de pouvoir insérer r' dans G, on tente de réécrire f pour que ses opérandes appartiennent au *même* arbre T de G. La question qu'on peut se poser naturellement est : peut-on insérer $f = h_1 \ op \ h_2$ dans G, si $h_1, h_2 \in G$, mais h_1, h_2 ne se trouvent pas dans le même arbre T de G? Et en quoi cette insertion serait-elle intéressante? Implicitement, les graphes qu'on manipule sont complètement connectés, c'est-à-dire il existe un chemin entre chaque paire de nœuds d'un graphe. Actuellement, l'insertion de f dans G quand h_1, h_2 ne se trouvent pas dans le même arbre ne peut pas être réalisée. Pour cela, il est besoin d'une structure plus complexe qu'un graphe mixte : un graphe "étendu" dans lequel on soit capables de modéliser des relations plus complexes que celle d'une inégalité de type $\preceq$ ou $\preceq_{aff}$, et des opérations sur ce type de relations. Parmi celles-ci, les relations affines de paramètres généraux (n, φ, d) et les opérations sur celles-ci, mais aussi des relations plus générales, comme celles existantes entre les horloges d'un arbre booléen. Cette extension pourrait être motivée et se dérouler en parallèle avec l'introduction de nouvelles règles de synchronisabilité basées sur les propriétés des opérations sur des relations affines *générales* de paramètres (n, φ, d), avec $n, d \in \mathbb{N}^*$ et $\varphi \in \mathbb{Z}$.

L'insertion de r' en G implique l'insertion de r' dans tous les arbres $T_i \in G$ qui contiennent les opérandes de f et l'unification des nœuds $\eta_i \in T_i$ ainsi obtenus. La procédure d'insertion de r' dans un arbre T_i suit le même schéma que celui de l'insertion d'une racine dans un arbre affine, section 10.2.3.1. La seule différence apparaît au niveau du placement de f dans T_i, qui est maintenant un arbre mixte. Dans l'absence d'un placement canonique – dû essentiellement à l'absence d'une forme canonique arborescente dans un arbre affine – la formule $f = h_1 \ op \ h_2$ sera placée afin de respecter la triangularité du système : f va être placée afin que tout parcours descendant gauche-droite calcul h_1 et h_2 avant f.

Nous décrivons l'unification des nœuds d'horloges dans la section suivante. Après insertion de r' dans G, la fusion des graphes G et G' implique l'union des sous-arbres mixtes A_1 et A_2 correspondant à r' dans G' et à f dans G. Nous avons choisi de remplacer l'union de A_1 et A_2 par leur union disjointe (voir section 10.1.3 pour la définition) ; nous évitons ainsi les problèmes liés à

9. La formule f contient au plus deux opérandes.

la réécriture de formules placées. Effectivement, l'union de A_1 et A_2 impliquerait l'unification des horloges équivalentes dans A_1 et A_2. Ce cas peut apparaître pour des horloges affines, ou expressions sur celles-ci, du fait de l'absence d'une forme canonique dans un arbre affine. L'unification de deux horloges nécessiterait la réécriture de toutes les formules placées construites sur les horloges unifiées. Afin de réaliser cette réécriture, les formules devraient être enlevées des arbres et le calcul d'horloges relancé sur celles-ci.

10.3.2.3 Synchronisabilité de formules

Après insertion de r' dans G et fusion de G et G', r' est définie par la formule f placée dans G et toute autre formule qui définit r' par une équation $r' = f'$ doit être prouvée synchronisable avec f. On obtient les cas suivants :

1) Les formules f et f' se trouvent dans le même arbre. Dans ce cas, le test de synchronisabilité de f et f' se réduit à la vérification de leur équivalence. Trois cas sont à considérer, en fonction de la forme des formules f et f' :

   1.1) f et f' sont des expressions sur des horloges affines ; dans ce cas, la preuve de leur équivalence peut suivre un schéma similaire à celui présenté en section 10.2.3.3.

   1.2) f et f' sont des expressions sur des sous-échantillonnages par conditions booléennes ; leur équivalence est la même chose que l'identité de leur forme canoniques booléennes.

   1.3) Dans tout autre cas, on construit les formules booléennes f_b et f'_b associées à f et f' et on vérifie l'identité de leurs formes canoniques. Ceci est possible étant donné que à toute horloge affine peut être associé un sous-échantillonnage par une condition booléenne particulière et à toute expression sur horloges affines une expressions booléenne. Nous avons basé la mise en œuvre du calcul mixte d'horloges sur cette constatation ; nous détaillons les aspects concernant cette mise en œuvre dans la section suivante.

2) Les formules f et f' se trouvent dans le même graphe mixte, mais elles appartiennent à des arbres différents. Dans ce cas, le test de synchronisabilité de f et f' nécessite la mise en œuvre de techniques de calculs sur les graphes mixtes similaires à celles exposées en section 10.2.4.3 pour les graphes affines. Ces techniques doivent permettre l'implémentation des règles de synchronisabilité générales que nous avons déduites en section 7.3.2. Par rapport au calcul affine d'horloges, elles doivent tenir compte du spécifique des graphes mixtes – qui englobent dans un même cadre deux structures différentes et complémentaires – et ne doivent pas permettre toutes les configurations possibles de graphes mixtes. En effet, dans certaines situations, un graphe mixte ne peut pas contenir deux arbres booléens parce que cela imposerait des contraintes (qu'on ne sait pas vérifier actuellement) sur les horloges de ces arbres (voir Figure 10.9 pour un exemple générique d'une telle configuration).

3) Les formules f et f' se trouvent dans des graphes différents. Dans ce cas, f et f' sont synchronisables si et seulement si une d'entre elles est affine, i.e. il existe k telle que $f = [k]_{(\varphi,d)}$ ou $f' = [k]_{(\varphi',d')}$ (conformément aux règles no. 8 et 8' présentées en section 7.3.2).

Dans le cas où les formules f et f' ont été prouvées synchronisables, l'étape suivante consiste dans l'unification des nœuds η et η' correspondant à f et f'. La formule f' étant remplacée par f, toutes les formules dans lesquelles f' apparaît doivent être réécrites conformément à cette substitution. Cette réécriture est plus complexe si les formules f et f' se trouvent dans le même arbre que si elles appartiennent à des arbres différents. Dans le premier cas, le calcul d'horloges doit être relancé

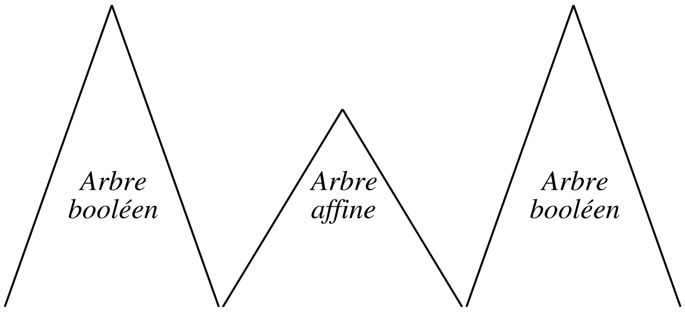

FIG. 10.9 – *Configuration non acceptable par le calcul mixte d'horloges.*

sur les formules à réécrire. Dans le cas second, il est nécessaire d'actualiser l'appartenance de ces formules aux arbres du graphes conformément au nouveaux pères de f' obtenus suite à l'unification de f et f'.

10.3.3 Mise en œuvre

Cette section décrit la mise en œuvre du calcul mixte d'horloges dont nous avons exposé les principes en section précédente. En première instance, il est apparu nécessaire d'étendre la structure d'arbre booléen – qui était la structure de représentation de base du calcul d'horloges existant – afin de représenter des graphes mixtes. Pour cela, nous avons étendu la représentation interne d'une expression d'horloges afin de pouvoir représenter tant des expressions sur horloges affines que des expressions sur des sous-échantillonnages par conditions booléennes. Cette extension est basée sur la représentation d'une horloge affine sous la forme d'un sous-échantillonnage par une condition booléenne particulière. Dans l'extension proposée, nous sommes ainsi en mesure de regrouper dans une seule structure les informations concernant les ordres partiels $\preceq$ et $\preceq_{aff}$. A partir de ces considérations, nous avons réalisé un prototype qui implémente un calcul mixte réduit basé uniquement sur la manipulation des arbres mixtes. Nous détaillons la réalisation de ce prototype et soulevons les principaux problèmes rencontrés. Nous discutons ensuite dans quelle mesure notre prototype est extensible vers la mise en œuvre d'un calcul mixte étendu par manipulation de graphes mixtes.

10.3.3.1 Représentation d'une horloge affine

Une horloge affine h_a est une horloge définie comme un sous-échantillonnage de phase φ positive et période d strictement positive sur une horloge arbitraire h ; cela s'écrit $h_a = [h]_{(\varphi,d)}$ ou encore

$$h_a \texttt{ := sample\{ } \varphi, d \texttt{ \}(} h \texttt{)}$$

en utilisant le processus affine `sample` introduit en section 7.4.2.1. La fonction réalisée par l'opérateur `sample` peut être décrite en SIGNAL en s'appuyant uniquement sur les quatre opérateurs élémentaires qui constituent le noyau du langage :

```
    ...
  ha := sample{ φ,d }( h )
    where
      process sample = { integer φ,d }
                       ( ? h ! ha )
              (|   C ^= h
               |   C := ( d  - 1) when (ZC = 0) default (ZC - 1)
               |   ZC := C $ 1 init φ
               |   ha := when (ZC = 0)
              |) / C, ZC
   ...
```

Dans le processus `sample` ci-dessus, nous avons exhibé un compteur `C` d'horloge égale à h qui permet de définir l'horloge h_a comme une horloge périodique sur h. Dans le diagramme de temps de la Figure 10.10 on peut voir la disposition des instants de présence de h_a par rapport à h; ces instants de présence coïncident avec les instants où le signal booléen ($\mathtt{ZC} = \mathtt{0}$) est présent et prend la valeur *true*. En conséquence, l'horloge h_a coïncide avec l'horloge des instants à *true* de ($\mathtt{ZC} = \mathtt{0}$), ce qui s'écrit $h_a = [ZC = 0]$, et on obtient ainsi une représentation de l'horloge affine h_a sous la forme d'une horloge sous-échantillonnage par la condition booléenne ($\mathtt{ZC} = \mathtt{0}$).

C	$\varphi-1$	$\varphi-2$	$\varphi-3$	...	0	$\boxed{d-1}$	$d-2$	$d-3$	...	0	$\boxed{d-1}$	$d-2$	$d-3$	...	0	$\boxed{d-1}$
ZC	φ	$\varphi-1$	$\varphi-2$	...	1	$\boxed{0}$	$d-1$	$d-2$	...	1	$\boxed{0}$	$d-1$	$d-2$	...	1	$\boxed{0}$
h	t	t	t	...	t	$\boxed{t}$	t	t	...	t	$\boxed{t}$	t	t	...	t	$\boxed{t}$
h_a	$\perp$	$\perp$	$\perp$	...	$\perp$	$\boxed{t}$	$\perp$	$\perp$	...	$\perp$	$\boxed{t}$	$\perp$	$\perp$	...	$\perp$	$\boxed{t}$

FIG. 10.10 – *Diagramme de temps pour le processus* `sample`.

Au cours du calcul d'horloges, une horloge affine peut être obtenue suite à un certain nombre d'opérations sur des horloges affines. On a vu, par exemple, en section 10.2.3.1, que l'horloge $[h]_{(\varphi,d)}$ peut se décomposer en

$$[h]_{(\varphi,d)} = (\vee)_{\varphi_i \in I_a}[h]_{(\varphi_i,n)}$$

avec n est un multiple de d et $I_a = \{\varphi, \varphi+d, \ldots, \varphi+(\frac{n}{d}-1)d\}$. En conséquence, il est possible que, au cours du calcul d'horloges, l'expression $(\vee)_{\varphi_i \in I_a}[h]_{(\varphi_i,n)}$ se réduise à $[h]_{(\varphi,d)}$. Dans ce cas, l'expression $(\vee)_{i=1}^{\frac{n}{d}-1}[ZC_i = 0]$ sera associée à $[h]_{(\varphi,d)}$ (en considérant que chaque horloge affine $[h]_{(\varphi_i,n)}$ est représentée par l'expression booléenne $[ZC_i = 0]$).

A part l'expression booléenne qui la définit, chaque horloge d'un arbre affine doit disposer d'une information concernant son caractère affine. Il s'agit d'un doublet de paramètres (φ, d) qui contient la phase φ positive et la période d strictement positive, si l'horloge est affine ; autrement, le doublet (φ, d) est non défini.

Le doublet de paramètres (φ, d) est étendu pour tout type d'horloge manipulée au cours du calcul d'horloges ; dans le cas des horloges sous-échantillonnages par conditions booléennes ou des expressions sur celles-ci, (φ, d) est non défini.

Avec les ingrédients précédents, nous avons étendu la structure d'arbre booléen à celle d'*arbre mixte*. Figure 10.11 illustre cette extension ; l'arbre mixte que nous avons représenté contient deux horloges affines et trois expressions sur des horloges qui sont des sous-échantillonnages par conditions booléennes.

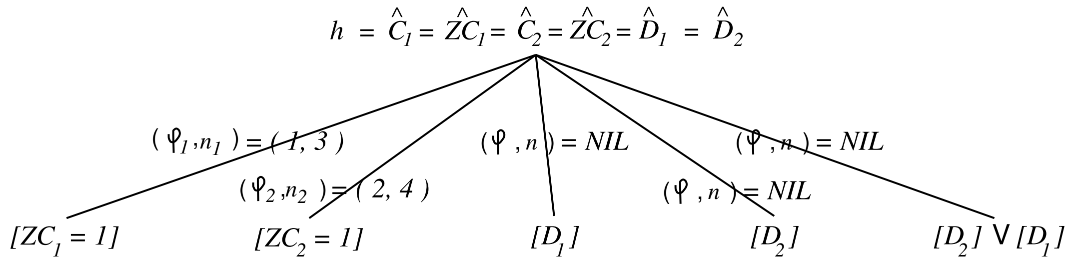

FIG. 10.11 – *Exemple d'arbre mixte.*

Nous avons implémenté un calcul d'horloges basé uniquement sur la manipulation d'arbres mixtes ; nous détaillons les principaux problèmes rencontrés et les solutions adoptées en section suivante. Ensuite, la structure d'arbre mixte doit être étendue à celle de graphe mixte. Nous discutons les problèmes qui apparaissent et les solutions envisageables en section 10.3.3.3.

10.3.3.2 Calcul mixte réduit & Arbre mixte

Le calcul réduit suit le même schéma que celui du calcul général présenté en section 10.3.2, avec l'observation que tout graphe mixte se réduit à un arbre mixte. La première étape consiste dans la création des arbres mixtes élémentaires, qui sont soit des arbres de partition, soit des arbres affines élémentaires. La deuxième étape est une fusion lancée sur l'ensemble des arbres mixtes élémentaires et réitérée sur les arbres mixtes créés par la suite, jusqu'à ce que les conditions de l'applicabilité de l'algorithme de fusion ne soient plus vérifiées. La fusion des arbres mixtes T et T' réalise l'orientation de l'équation $r' = f$, avec r' le représentant de la racine de l'arbre T' et f une formule appartenant à l'arbre T. Ceci implique l'insertion de r' dans T, qui consiste essentiellement du placement de f dans T, et la fusion des sous-arbres A_1 et A_2 correspondant respectivement à r' et f (la déduction de l'algorithme de fusion pour les arbres mixtes est immédiate à partir de la variante pour les graphes mixtes présentée en section 10.3.2.2).

Dans l'absence d'une forme canonique dans l'arbre mixte, le placement de f est réalisé dans le but de respecter la triangularité du système final : si $f = h_1 \ op\ h_2$, f est placée de telle manière que tout parcours descendent gauche-droite calcule h_1 et h_2 avant de calculer f. Le placement non canonique peut introduire deux horloges équivalentes placées différemment. Ceci influence le processus de fusion des sous-arbres A_1 et A_2 comme décrit précédemment, ainsi que les preuves d'équivalence de formules placées.

Premièrement, la fusion de A_1 et A_2 doit réaliser l'unification de toutes les horloges équivalentes placées également dans A_1 et A_2. Comme décrit en section 10.3.2.2, ceci implique la réécriture de toutes les formules placées construites sur deux horloges h_1 et h_2 unifiées de cette manière. Le calcul d'horloges que nous avons implémenté ne réalise pas cette réécriture ; ce processus aurait ralenti considérablement la compilation d'un programme SIGNAL et aurait été à l'encontre des principes de simplicité et d'efficacité sur lesquels est basé le compilateur de SIGNAL.

Pour les mêmes raisons nous n'avons pas effectué l'unification des formules d'un même arbre contraintes à être synchrones et prouvées synchronisables, i.e. équivalentes pour le cas particulier des arbres (conformément à la section 10.3.2.3). De plus, cette unification aurait posé des problèmes concernant l'unicité de la forme canonique *booléenne* d'une expression d'horloges.

On a vu en section 10.3.3.1 que à chaque expression sur horloges affines (expression affine) est associée une expression sur des horloges sous-échantillonnages par conditions booléennes de type $[ZC_i = 0]$ (expression booléenne). Au cours du calcul d'horloges, la forme canonique arborescente booléenne est calculée pour chaque horloge. Au cas où le test de l'identité des formes canoniques booléennes de f et f' échoue, leur équivalence est vérifiée à partir des informations concernant leur caractère affine. Si $f = h_1 \ op\ h_2$ et $f' = h'_1 \ op\ h'_2$, avec h_1, h_2, h'_1, h'_2 horloges affines, dans le cas où f et f' se réduisent chacune à une horloge affine, i.e. $f = [h]_{(\varphi,d)}$ et $f' = [h]_{(\varphi',d')}$, alors f et f' sont équivalentes si et seulement si $\varphi = \varphi'$ et $d = d'$. L'unification de f et f' induirait un problème concernant le choix de la forme canonique booléenne qui sera associée à f (et f').

A la fin du calcul d'horloges, si toutes les horloges du programme SIGNAL en cours de compilation ont été englobées dans un arbre mixte unique et toutes les équivalences ont été prouvées, le programme est correct du point de vue de ses synchronisations et il peut être exécuté. La racine de l'arbre mixte sera l'horloge qui cadencera son exécution. En ce qui concerne les horloges, la génération de code peut être résumée par un parcours descendant gauche-droite de l'arbre mixte. Chaque horloge est calculée en fonction de son expression de définition ; les horloges affines auxquelles sont attribuées des expressions booléennes de la forme $[ZC_i = 0]$ seront calculées par des compteurs modulo n initialisés à φ, avec (φ, n) les paramètres affines de l'horloge.

10.3.3.3 Calcul mixte étendu & Graphe mixte

L'implémentation de la structure de graphe mixte à partir de celle d'arbre mixte que nous avons proposée poserait plusieurs types de problèmes. Il y a d'abord l'appartenance d'une horloge à plusieurs arbres, donc la nécessité de conserver pour chaque horloge la liste de ses pères. En sens inverse, chaque père doit garder une liste de ses fils ; cela permettrait des recherches dans les graphes mixtes selon certains critères établis (recherches orientées) parmi lesquels ceux utilisés dans les preuves d'équivalence de formules d'horloges. Actuellement, il existe une structure d'arbre avec les arcs orientés vers la racine et cela permet uniquement l'accès de chaque horloge vers son père.

Pour chaque nœud d'un graphe mixte, il faut différencier au plus un père particulier dans la liste de ses pères : il s'agit du père "booléen", celui qui fixe l'appartenance du nœud à l'arbre booléen (le cas échéant). En ce qui concerne les pères "affines", une gestion des paramètres affines (φ, n) correspondant à chaque père, le cas échéant, doit être envisagée.

Finalement, la génération de code exécutable pour la simulation doit être adaptée à la nouvelle structure de graphe (mixte). On peut envisager, par exemple, que tout programme pour lequel le calcul d'horloges a produit un graphe unique est endochrone et la racine de l'arbre booléen situé au plus haut niveau dans le graphe (ou la racine de l'arbre affine avec la même propriété, si aucun arbre booléen ne satisfait cette condition) cadence l'exécution du programme. Toutes les autres horloges peuvent être calculées à partir de cette racine ; dans l'exemple de la Figure 10.12, l'horloge h se calcule à partir de h_1 par sur-échantillonnage et h_2, $[D_3]$, $[\neg D_3]$ peuvent être exprimées ensuite en fonction de h. Il reste à résoudre le problème du calcul multiple des horloges appartenant à plusieurs arbres ; chaque horloge déjà calculée devrait être marquée afin de ne la calculer qu'une seule fois.

Pour conclure, on remarque la nécessité de la mise en place d'une structure de données adaptée à la représentation de graphes bi-directionnels (les arcs des graphes seront orientés dans les deux sens afin de faciliter l'accès dans les deux directions) et d'un ensemble d'outils pour la manipulation de ces graphes, d'après le modèle du type abstrait « arbre » existant actuellement.

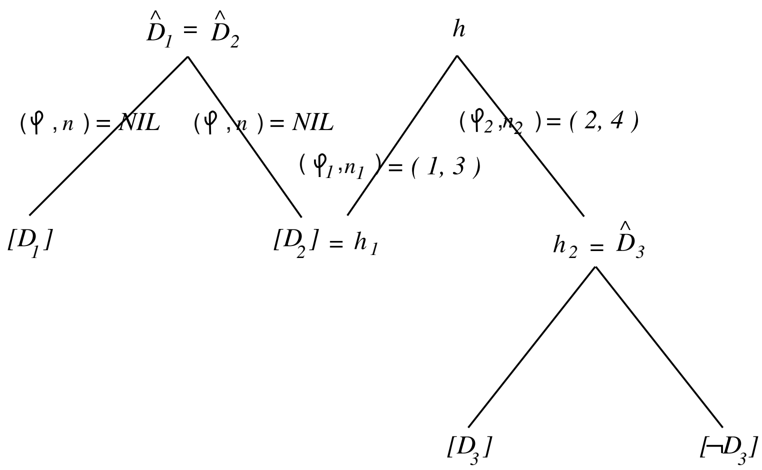

FIG. 10.12 – *Illustration de la génération de code exécutable dans un graphe mixte.*

L'idée des opérations sur des « classes » de graphes apparaît naturellement à partir de la notion de graphe mixte. Une classe de graphes peut être associée à un espace particulier d'horloges : la classe des graphes booléens (qui est une extension possible et envisageable de la classe d'arbres booléens existante), la classe des graphes affines, etc. A part une définition rigoureuse des propriétés de chaque classe – comme on dispose actuellement pour les arbres booléens et on a défini pour les graphes affines – une interface de communication doit être définie entre chaque classe. Cette interface devra traiter les aspects implicites et explicites liés à la manipulation d'horloges appartenant à des graphes de classes différentes (opérations sur des formules d'horloges dont les opérandes se trouvent dans des classes différentes, équivalence d'horloges appartenant à des classes différentes, etc).

Dans cet esprit, le graphe mixte peut être considéré comme l'union des classes des graphes affines et booléens. En première instance, nous avons défini l'interfaçage entre les deux classes en ce qui concerne les arbres en précisant : 1) la représentation d'une horloge affine sous la forme d'une expression booléenne caractérisé par un doublet (φ, n) ; 2) l'extension des paramètres (φ, n) pour les horloges booléennes ; 3) la spécification des traitements à effectuer à chaque pas du calcul d'horloges. Cette interface peut être étendue aux graphes mixtes. Il suffit pour cela d'étendre les points 1) et 2) pour définir un interfaçage au niveau des structures de données manipulées qui soit cohérent avec le type graphe bi-directionnel et de spécifier les traitements (point 3) supplémentaires. Parmi les problèmes à résoudre :

- Assurer l'unicité de l'expression booléenne de toute horloge, même dans le cas de son appartenance à plusieurs arbres ;

- Associer à chaque horloge une liste de doublets (φ, n) en directe correspondance avec la liste de ses pères affines ;

- Définir un algorithme d'unification cohérent avec les propriétés du graphe mixte ;

- Intégrer une procédure de génération de code adéquate ;

- etc...

10.4 Conclusion

Dans ce chapitre nous avons abordé les problèmes liés à la mise en œuvre du calcul affine d'horloges et à son intégration avec le calcul booléen existant. En première instance, nous avons analysé dans quelle mesure les méthodes et outils disponibles pour le calcul booléen sont extensibles au nouveau calcul affine, considéré comme indépendant du calcul existant. L'aspect le plus important à remarquer dans ce contexte était la nécessité d'étendre la structure de représentation d'un système triangulaire sur horloges – qui était l'arbre – à celle de *graphe*. Nous avons exposé brièvement les principaux problèmes liés à la manipulation de cette structure, qui tenaient essentiellement de la mise en œuvre de l'ensemble de règles de synchronisabilité que nous avons proposées dans le contexte des relations affines.

Ensuite, nous avons décrit les principes de base de la mise en œuvre du calcul affine, qui a généré un premier prototype d'intégration des calculs affine et booléen. Le calcul *mixte* d'horloges que nous avons défini est basé sur la manipulation de *graphes mixtes* qui sont des graphes acycliques complètement connectés contenant des arbres booléens et des graphes affines. En tenant compte des propriétés de leurs composants, nous avons défini l'ensemble des opérations sur graphes mixtes utilisées au cours du calcul mixte d'horloges. Dans le prototype réalisé, la structure d'arbre booléen a été étendue pour permettre la description d'*arbres mixtes* et tous les opérations mises en œuvre ensuite s'effectuent uniquement sur arbres mixtes. A plus long terme, l'extension proposée peut constituer le point de départ vers l'implémentation de la structure de graphe mixte ou de *classes de graphes* d'après le modèle du graphe mixte.

Chapitre 11

Applications du calcul affine d'horloges

Le calcul affine d'horloges augmente le domaine d'applicabilité du langage SIGNAL par l'introduction de techniques de manipulation d'objets nouveaux, qui renforcent le caractère flots de données du langage. Nous avons en vue essentiellement la possibilité d'effectuer des transformations affines sur des variables d'horloges et de calculer les synchronisations sur les horloges transformées. Le nouveau calcul sur les synchronisations s'ajoute aux méthodes de preuve formelle existantes dans le compilateur de SIGNAL et s'inscrit parmi les efforts réalisés pour augmenter la puissance de spécification et validation de SIGNAL. Pour exemplifier cette idée, nous introduisons en section 11.1 un exemple simple pour illustrer le calcul d'horloges affines et son utilisation dans la spécification d'applications temps-réel flots de données.

De manière plus générale, le calcul affine d'horloges constitue l'étape préliminaire pour l'intégration dans le langage SIGNAL des techniques d'optimisation de code rencontrées dans les paralléliseurs automatiques. Ainsi, les transformations affines d'horloges sont apparues en SIGNAL comme la conséquence des raffinements de programmes réalisés dans le contexte de la conception conjointe matérielle/logicielle (codesign) avec les langages SIGNAL et ALPHA. L'idée était de pouvoir bénéficier en SIGNAL des techniques d'implémentation optimale de programmes existantes en ALPHA et qui ont été inspirées par les méthodes de parallélisation automatiques de programmes. Dans ce sens, l'application de codage d'images vidéo à 34 Mbit/s [Sch92] a constitué le support de notre travail sur lequel nous avons pu tester les principaux concepts et résultats concernant l'interfaçage SIGNAL - ALPHA et le calcul affine d'horloges. Nous décrivons cette implémentation en section 11.2.

11.1 Exemples

Nous avons représenté sur la Figure 11.1 un programme SIGNAL contenant plusieurs appels au processus affine `sample`. Le système d'équations sur horloges affines associé au processus décrit par `PROGRAMME` est illustré en partie droite de la figure. Les équations 1), 2) et 3) définissent les horloges $\hat{s_1}$, $\hat{s_2}$ et $\hat{s_3}$. En ce qui concerne l'horloge $\hat{s}$, l'équation 4) peut être orientée de droite à gauche afin de définir l'horloge $\hat{s}$ comme égale à $\hat{s_1}$. La dernière équation ne peut plus être orientée de manière similaire puisque $\hat{s}$ possède déjà une définition. En conséquence, $\hat{s_3}$ doit être prouvée égale à $\hat{s_1}$. Conformément au treillis d'ordre $\preceq_{aff}$ représenté sur la Figure 11.2, $\hat{s_1}$ et $\hat{s_3}$ sont équivalentes si et seulement si $n_2\varphi_3 + \varphi_2 = \varphi_1$ et $n_1 = n_2n_3$ (voir les propriétés de l'opérateur de composition de relations dans un arbre affine présentées au chapitre précédent). Les valeurs des paramètres

n_1, n_2, n_3 et $\varphi_1, \varphi_2, \varphi_3$ sont analysées au cours du calcul d'horloges afin de vérifier si le programme est correct du point de vue de ses synchronisations.

```
process PROGRAMME = { integer n1, n2, n3,
                      phi1, phi2, phi3 }
  ( ? integer e;  ! integer s )
   (| s1 := sample { phi1, n1 }(e)
    | s2 := sample { phi2, n2 }(e)
    | s3 := sample { phi3, n3 }(s2)
    | s := s1 + s3  |) / s1, s2, s3
  where integer s1, s2, s3;
  end
end
```

Équations sur horloges

1) $\hat{s_1} = [\hat{e}]_{(\varphi_1, n_1)}$
2) $\hat{s_2} = [\hat{e}]_{(\varphi_2, n_2)}$
3) $\hat{s_3} = [\hat{s_2}]_{(\varphi_3, n_3)}$
4) $\hat{s} = \hat{s_1}$
5) $\hat{s} = \hat{s_3}$

FIG. 11.1 – *Le processus* PROGRAMME *et le système d'équations sur horloges affines associé.*

En section 2.3.2, nous avons présenté un exemple de modélisation d'architectures VLSI synchrones et périodiques avec SIGNAL. Le modèle VSDF d'un tel système a été décrit et simulé en SIGNAL : à chaque nœud VSDF ont été associé une partie comportementale, décrite en C, une partie de gestion temporelle des communications et une partie de vérification. Concernant la partie vérification, un ensemble de variables ERROR caractérisant les états d'erreur du système ont été introduites dans le but de : 1) spécifier les synchronisations sur les entrées/sorties des composants VSDF qui communiquent entre eux ; 2) permettre l'accès aux outils de preuve formelle [Dut92] [ALMR95] afin de vérifier les contraintes de synchronisation. Une variable d'erreur à vrai sur une trajectoire du système dans l'espace des états possibles impliquait des transferts de données incorrects.

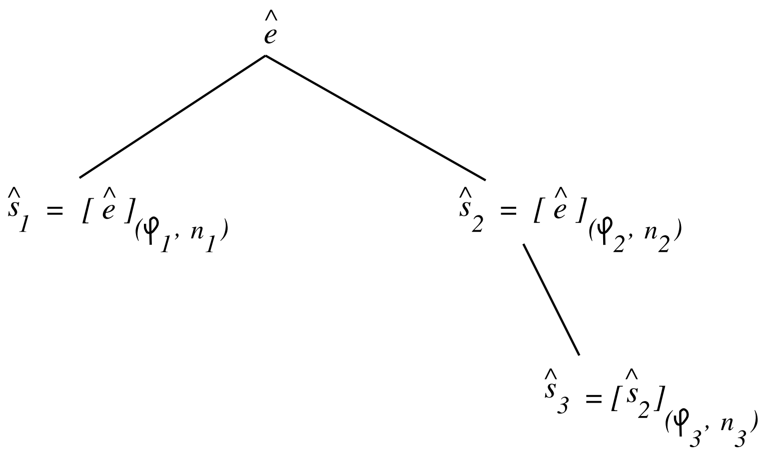

FIG. 11.2 – *Le treillis d'horloges affines associé à* PROGRAMME.

La vérification formelle de l'architecture VLSI a nécessité la réécriture de la bibliothèque de gestion temporelle des communications afin de n'utiliser que des opérations booléennes. Ensuite, une vérification statique a été réalisée sur une partie de l'application ; la validation de l'application entière s'est montré coûteuse en temps de calcul et espace mémoire, suite à l'explosion du nombre de variables booléennes.

Or, les horloges des entrées et sorties des composants VSDF sont des sous-échantillonnages affines sur l'horloge globale du système, ou des expressions sur des sous-échantillonnages affines. La vérification des synchronisations sur ces horloges peut être réalisée en utilisant le calcul affine d'horloges. En effet, nous avons implémenté une partie préliminaire qui pourrait être étendue par un calcul de forme normale relative afin de permettre la vérification de la synchronisabilité de deux expressions arbitraires sur des horloges affines (voir sections 10.2.3.1 et 10.2.3.3).

11.2 L'application de codage vidéo à 34 Mbit/s

Dans le cadre du codesign avec SIGNAL et ALPHA, nous avons testé notre prototype du calcul affine d'horloges sur l'application de codage d'images vidéo à 34 Mbit/s [Sch92]. Cette application est constituée d'un ensemble de traitements numériques sur des séquences d'images de même dimension ; chaque image est divisée en un nombre fixe de macro-blocs de luminance et de chrominance et les traitements sont appliqués itérativement sur chaque macro-bloc. Il s'agit principalement d'algorithmes de codage intra et inter-image impliquant des opérations comme la transformation en cosinus discret (DCT) ou l'estimation de mouvement. Cette application peut être vue comme étant constituée d'une partie importante de contrôle, qui peut être spécifiée avec SIGNAL, reliant des opérations coûteuses en temps de calcul (DCT), qui peuvent être décrites facilement avec des équations de récurrence affine. Nous avons identifié une partie de l'application dans laquelle l'algorithme de transformation en cosinus discret (DCT), initialement programmé en C, a été décrit en ALPHA et intégré dans le programme SIGNAL décrivant le contrôle. Nous avons considéré les deux niveaux d'interfaçage décrits en section 4.2.1 : au niveau fonctionnel, la DCT était une opération instantanée dans la logique temporelle de SIGNAL ; au niveau architectural, nous avons pris en considération le temps d'exécution de la DCT conformément aux optimisations proposées par l'environnement de ALPHA et avons dû raffiner la spécification SIGNAL initiale. De cette manière, nous avons effectué des transformations affines sur certaines variables d'horloges et avons tester leurs re-synchronisations en utilisant le calcul affine d'horloges.

L'application support est significative, premièrement, parce qu'elle nous a permis de valider les principaux concepts que nous nous sommes proposés d'aborder dans notre étude de codesign avec SIGNAL et ALPHA, i.e. la cospécification, la cosimulation et la synthèse automatique d'architectures parallèles spécialisées, et deuxièmement, parce qu'elle contient un ensemble d'opérations, critiques du point de vue du temps d'exécution/ressources, que l'on rencontre dans des standards de compression d'images comme MPEG.

Conclusion & Perspectives

Le travail exposé dans cette thèse est une contribution au développement de l'environnement de programmation d'applications temps-réel critiques existant autour du langage SIGNAL. Dans ce cadre, nous avons visé trois buts essentiels : 1) l'intégration en SIGNAL de certaines techniques d'optimisation de code rencontrées dans les compilateurs paralléliseurs ; 2) augmenter les capacités de spécification et validation de SIGNAL par un calcul formel plus puissant à l'intérieur de son compilateur ; 3) développer un prototype de test des principaux résultats concernant les points précédents en utilisant une application significative du traitement de l'image.

Intégration des langages SIGNAL et ALPHA

Nos travaux ont débuté par une étude sur l'intégration des langages SIGNAL et ALPHA et leur adaptabilité à la *conception mixte matériel/logiciel (codesign)* d'applications temps-réel critiques de traitement du signal ou de l'image. Notre démarche était motivée par le caractère complémentaire des deux langages. Ainsi, leur utilisation conjointe a fourni des solutions fiables à certains problèmes du domaine du codesign, parmi lesquels la cospécification, la cosimulation et la synthèse matérielle et logicielle d'implémentations optimales pour des algorithmes coûteux en temps de calcul. Dans ce contexte, il restait à résoudre les problèmes liés à l'*interfaçage* des deux langages. Nous avons défini deux niveaux d'interfaçage correspondant aux vues fonctionnelle et architecturale d'un système et avons mis en place les mécanismes nécessaires pour le passage d'un niveau à l'autre conformément aux divers raffinements qu'une spécification peut subir. Outre la gestion des communications entre les programmes SIGNAL et ALPHA, nous avons été confrontés au problème du *calcul des nouvelles synchronisations* dans les spécifications SIGNAL transformées. Le développement de ce calcul formel a constitué le deuxième objectif que nous nous sommes proposé et que nous décrivons dans la section suivante.

Une fois l'interfaçage défini, nous disposions des ingrédients nécessaires pour pouvoir bénéficier en SIGNAL des techniques d'implémentation optimale de programmes existantes en ALPHA et qui ont été inspirées par les méthodes de parallélisation automatique de programmes. En même temps, l'environnement de SIGNAL fournit des outils formels pour la preuve de propriétés de programmes. Plus particulièrement, dans le contexte de codesign proposé il est possible d'effectuer la simulation fonctionnelle et/ou la vérification formelle de spécifications obtenues par raffinements temporels (*retiming*) successifs dans le but de décrire et analyser diverses implémentations possibles pour un système donné. Nous n'avons pas étudié les problèmes du partitionnement automatique d'un système en matériel et logiciel, ni de l'interfaçage entre les parties matérielle et logicielle.

Calcul formel sur les transformations affines d'horloges

Les transformations de spécifications SIGNAL à divers niveaux d'abstraction ont induit des *transformations affines* sur les horloges et la *re-synchronisation* des horloges transformées. Dans ce contexte, nous nous sommes proposé d'aborder les aspects et problèmes concernant la mise en place du *calcul formel* nécessaire au traitement des nouvelles synchronisations. Nous avons proposé le *modèle sémantique des traces* de SIGNAL, introduit dans [LG91], comme contexte formel pour la dérivation de nos principaux résultats. Ce contexte mathématique s'est montré approprié pour l'étude des propriétés concernant les *relations affines* existantes entre chaque paire d'opérandes d'une transformation affine. Notre premier but a été de spécifier soigneusement et complètement le modèle mathématique en précisant les briques de base pour le développement de nos résultats ultérieurs. Ensuite, avec les outils mathématiques que nous nous sommes créés, nous avons défini un ensemble de propriétés sur les relations affines et sur les opérations qu'elles peuvent subir. Ces opérations ont été induites par les opérations d'union, d'intersection et de différence sur horloges. Nous avons traité intégralement le cas particulier des relations affines de paramètres $(1, \varphi, d)$, avec $\varphi \geq 0$ et $d \geq 1$, et avons exposé les principaux problèmes soulevés par le cas général (n, φ, d), avec $\varphi \geq 0$ et $n, d \geq 1$.

A partir de ces résultats, nous avons défini un système minimal de *règles de synchronisabilité* d'horloges qui s'ajoutent à celles déjà disponibles dans le calcul des synchronisations *(calcul d'horloges)* du langage SIGNAL. Notre travail est original en deux aspects : 1) il introduit le concept de synchronisabilité par l'intermédiaire des règles de synchronisabilité qui existaient de manière implicite derrière le calcul d'horloges ; 2) il propose une extension de l'ensemble des règles existantes dans le but d'augmenter la puissance de preuve formelle du calcul d'horloges. Le système de règles de synchronisabilité que nous avons proposé s'est montré suffisant pour les besoins de validation et simulation dans le contexte de codesign de départ. Il est modulaire et de cette façon *extensible* en fonction de nouvelles propriétés des opérations d'union, intersection ou différence sur des relations affines générales. Il représente une contribution importante aux techniques de spécification et vérification formelle intégrées dans l'environnement du langage SIGNAL. Dans ce sens, il peut être considéré comme une alternative à certaines méthodes de vérification disponibles dans l'outil de preuve SIGALI[1]. Nous avons contourné les problèmes soulevés par l'exploration de l'espace des états d'un programme en proposant un calcul simple et élégant qui tire parti des propriétés sur les *nombres entiers*.

Mise en œuvre

Finalement, nous avons utilisé les règles de synchronisabilité proposées pour définir un calcul *affine* d'horloges d'après le modèle du calcul *booléen* existant. Nous avons proposé de résoudre tout système d'équations impliquant des expressions sur horloges affines[2] par une *triangularisation* conforme aux propriétés de l'espace des horloges affines. Nous avons introduit le *graphe affine* comme la structure de base pour la représentation d'un système triangulaire et avons étudié l'influence de ses propriétés sur le processus de triangularisation.

Cette étude préliminaire a servi ensuite à la réalisation d'un prototype d'intégration des calculs affine et booléen. Quoique réduite, notre implémentation nous a permis de valider les résultats concernant le calcul formel des synchronisations sur horloges affines et les concepts de base de

1. SIGALI est un système formel de preuve de propriétés statiques et dynamiques de programmes SIGNAL basé sur le formalisme des systèmes dynamiques polynômiaux.

2. Sous-échantillonnages affines.

notre approche de codesign sur une application significative du domaine du traitement de l'image, le codage vidéo à 34 Mbit/s.

Perspectives

Pour l'avenir, notre travail peut constituer le point de départ vers l'intégration en SIGNAL des techniques de manipulation de structures de données régulières. Nous avons en vue les calculs formels sur les *polyèdres convexes* et *Z-polyèdres*, comme ils apparaissent dans les méthodes d'implémentation optimale de programmes rencontrées dans l'environnement du langage ALPHA. A l'appui de cette idée on peut considérer les similitudes constatées entre, d'une part, les résultats formels sur les relations affines et le calcul affine d'horloges, et d'autre part le calcul sur Z-polyèdres. En même temps, nous avons remplacé une éventuelle étude de *synchronisabilité* sur des variables d'horloges par une étude un peu plus générale sur l'*identifiabilité* de variables de mêmes domaines de valeurs et d'horloges différentes et indépendantes des valeurs. Dans le cas des variables de domaines différents, l'identifiabilité doit prendre en considération ces domaines. Un calcul de domaines convexes uni-dimensionnels peut être envisagé d'après le modèle existant en ALPHA ; il impliquera essentiellement des opérations sur polyèdres – intersection, union, test d'égalité – et sera extensible aux variables multi-dimensionnelles pour lesquelles la bibliothèque polyédrale de ALPHA est disponible.

Ensuite, le calcul sur relations affines peut constituer un bon point de départ pour l'étude des horloges *vectorielles* en SIGNAL. Actuellement, toute structure multi-dimensionnelle est supposée être synchrone, c'est-à-dire tous les éléments de la structure doivent avoir la même horloge. Or, dans certains cas il peut être intéressant d'avoir des tableaux d'éléments asynchrones. Dans ce cas, l'étude que nous avons réalisée devrait être étendue au cas multi-dimensionnel et intégrée avec les résultats disponibles dans le monde des Z-polyèdres.

Bibliographie

[ABK+97] Allemand M., Bodin F., Kountouris A., Le Guernic P., Le Lann J.C., Seznec A. et Wolinski C. A synchronous approach for hardware design. Rapport technique 1131, IRISA, Rennes, France, 1997.

[Ack82] Ackerman W.B. Data flow languages. Dans *Selected reprints on Dataflow and Reduction Architectures*, Thakkar S.S., éditeur, chapitre 4, pp. 179–188. The computer society of IEEE, 1982.

[Agi95] Agi I. GLU for Multidimensional Signal Processing. Dans *Proceedings of the International Symposium on LUCID and Intensional Programming (ISLIP), Sydney, Australie*, Mai 1995.

[ALMR95] Amagbegnon T., Le Guernic P., Marchand H. et Rutten E. The SIGNAL dataflow methodology applied to a production cell. Rapport technique 917, INRIA, France, Mars 1995.

[AM95] Aubry P. et Machard S. Distribution synchrone des programmes SIGNAL. Dans *Actes des 2es Journées Jeunes Chercheurs en Systèmes Répartis, Rennes, France*, Octobre 1995.

[Ama95] Amagbegnon T. *Forme canonique arborescente des horloges de SIGNAL*. Thèse de doctorat, Université de Rennes 1, France, Décembre 1995.

[Anc91] Ancourt C. *Génération automatique de codes de transfert pour multiporcessurs à mémoire locale*. Thèse de doctorat, Université de Paris VI, France, 1991.

[Aub97] Aubry P. *Mises en œuvre distribuées de programmes synchrones*. Thèse de doctorat, Université de Rennes 1, France, Octobre 1997.

[AW77] Ashcroft E.A. et Wadge W.W. LUCID, A Nonprocedural Language with Iteration. *Communications of the ACM*, 20(7), Juillet 1977.

[Ban93] Banerjee U. *Loop Transformations for Restructuring Compilers - The Foundations*. Kluwer Academic Publishers, 1993.

[BB91a] Benveniste A. et Berry G. Special section on another look at real-time programming. *Proceedings of the IEEE*, 79(9):1268–1336, Septembre 1991.

[BB91b] Benveniste A. et Berry G. The Synchronous Approach to Reactive and Real-Time Systems. *Proceedings of the IEEE*, 79(9), Septembre 1991.

[BD91] Boussinot F. et De Simone R. The Esterel Language. *Proceedings of the IEEE*, 79(9):1293–1304, Septembre 1991.

[BDRR93] Boulet P., Darte A., Risset T. et Robert Y. (Pen)-ultimate tiling? Rapport technique 36, Ecole Normale Supérieure de Lyon, France, Novembre 1993.

[Bel93] Belhadj M. Vers la synthèse automatique de circuits à partir de programmes SIGNAL. Rapport technique 2039, INRIA, France, Septembre 1993.

[Bel94a] Belhadj M. *Conception d'architectures en utilisant SIGNAL et VHDL*. Thèse de doctorat, Université de Rennes 1, France, 1994.

[Bel94b] Belhadj M. VHDL & SIGNAL : A Cooperative Approach. Dans *Proceedings of the International Conference on Simulation and Hardware Description Languages, Tempe, Arizona, USA*, pp. 76–81. Society for Computer Simulation, 1994.

[Ber89] Berry G. Real-Time Programming : Special Purpose or General Purpose Languages. Dans *Information Processing*, Ritter G.X., éditeur, vol. 89. Elsevier Science Publishers B.V. (North Holland), 1989.

[Bes92] Besnard L. *Compilation de SIGNAL : horloges, dépendances, environnement*. Thèse de doctorat, Université de Rennes 1, France, Septembre 1992.

[BG92] Berry G. et Gonthier G. The ESTEREL synchronous programming language : design, semantics, implementation. *Science of Computer Programming*, pp. 87–152, 1992.

[BGL+98] Beauvais J.-R., Gautier T., Le Guernic P., Houdebine R. et Rutten E. A Translation of Statecharts into Signal. Dans *Proceedings of the International Conference on Application of Concurrency to System Design, Japon*, Mars 1998.

[BL87] Benveniste A. et Le Guernic P. A denotational theory of synchronous communicating systems. Rapport technique 360, INRIA, France, Mai 1987.

[BLL88] Benveniste A., Le Goff B. et Le Guernic P. Hybrid dynamical systems theory and the language SIGNAL. Rapport technique 838, INRIA, France, Avril 1988.

[BLM97] Bournai P., Le Borgne M. et Marchand H. Environnement de conception d'automatismes discrets basé sur le langage SIGNAL. Rapport technique 3254, INRIA, France, Septembre 1997.

[BQR+90] Benaini A., Quinton P., Robert Y., Saouter Y. et Tourancheau B. Synthesis of a new systolic architecture for the algebraic path problem. *Science of Computer Programming*, 15:135–158, 1990.

[CFG+92] Charot F., Frison P., Gautrin E., Lavenier D, Quinton P., Raimbault F. et Wagner Ch. From equations to hardware. Towards the systematic mapping of algorithms onto parallel architectures. Dans *Proceedings of the Second International Conference on Parallel Image Analysis, Ube, Japon*, Décembre 1992.

[Ché91] Chéron B. *Transformations syntaxiques de programmes SIGNAL*. Thèse de doctorat, Université de Rennes 1, France, Septembre 1991.

[CLM91] Clarke E.M., Long D.E. et McMillan K.L. A Language for Compositional Specification and Verification of Finite State Hardware Controllers. *Proceedings of the IEEE*, 79(9):1283–1291, Septembre 1991.

[CON97a] CONSORTIUM SACRES. Deliverable report i2.a: The SACRES Design Methodology for Safety Critical Systems. Esprit project EP 20897: Sacres, Avril 1997.

[CON97b] CONSORTIUM SACRES. The Semantics Foundations of SACRES. Esprit project EP 20897: Sacres, Avril 1997.

[CPHP87] Caspi P., Pilaud D., Halbwachs N. et Plaice J. LUSTRE : a declarative language for programming synchronous systems. Dans *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, Janvier 1987.

[CWB94] Chou P., Walkup E.A. et Borriello G. Scheduling for Reactive Real-Time Systems. *IEEE Micro*, 14(4):37–47, Août 1994.

[De 94] De Micheli G. Computer-Aided Hardware-Software Codesign. *IEEE Micro*, 14(4):8–9, Aôut 1994.

[DPR94] Dion M., Philippe J.-L. et Robert Y. Parallelizing compilers: what can be achieved? Rapport technique, LIP-Lyon, France, Mars 1994.

[DR93] Darte A. et Robert Y. A graph-theoretic approach to the alignment problem. Rapport technique 20, Ecole Normale Supérieure de Lyon, France, Juillet 1993.

[DR95] Darte A. et Robert Y. Affine-by-Statement Scheduling of Uniform and Affine Loop Nests over Parametric Domains. *Journal of Parallel and Distributed Computing*, 29:43–59, 1995.

[DR96] Dion M. et Robert Y. Mapping affine loop nests. *Parallel Computing*, 22.10:1373–1397, 1996.

[DRR96] Dion M., Randriamaro C. et Robert Y. Compiling affine nested loops: how to optimize the residual communications after the alignment phase? *Journal on Parallel and Distributed Computing*, 38.2:176–187, 1996.

[Dut92] Dutertre B. *Spécification et preuve de systèmes dynamiques*. Thèse de doctorat, Université de Rennes 1, France, Décembre 1992.

[EHB93] Ernst R., Henkel J. et Benner T. Hardware-Software Cosynthesis for Microcontrollers. *IEEE Design & Test of Computers*, 10(4):64–75, Décembre 1993.

[ETS90] ETSI. Specification of component TV codecs 32-45 mbit/s. Rapport technique, European Telecommunication Standards Institute, Décembre 1990.

[Fea91] Feautrier P. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming*, 20(1):23–53, Février 1991.

[Fea92] Feautrier P. Some efficient solutions to the affine scheduling problem, part i: One dimensional time, and part ii: Multidimensional time. *International Journal of Parallel Programming*, 21(5-6), 1992.

[Fea96] Feautrier P. Automatic Parallelization in the Polytope Model. Dans *Proceedings of the Spring School on Data Parallelism, Les Ménuires, France*, pp. 79–100, 1996.

[FJ93] Faustini A.A. et Jagannathan R. Multidimensional problem solving in LUCID. Rapport technique, Department of Computer Science, Arizona State University & Computer Science Lab, SRI International, Californie, 1993.

[GCD94] Gupta R.K., Coelho C.N. et De Micheli G. Program Implementation Schemes for Hardware-Software Systems. *Computer*, pp. 48–55, Janvier 1994.

[GD92] Gupta R.K. et De Micheli G. System-Level Synthesis using Re-programmable Components. Dans *Proceedings of the Third European Conference on Design Automation*, pp. 2–7. IEEE CS Press, 1992.

[GD93] Gupta R.K. et De Micheli G. Hardware-Software Cosynthesis for Digital Systems. *IEEE Design & Test of Computers*, 10(3):29–41, Septembre 1993.

[Geh84] Gehani N. *ADA Concurrent Programming.* Prentice Hall, 1984.

[GLD94] Gautier T., Le Guernic P. et Dupont F. SIGNAL v4 : manuel de référence. Rapport technique 832, IRISA, Rennes, France, Juin 1994.

[GLM94] Gautier T., Le Guernic P. et Maffeïs O. For a new real-time methodology. Rapport technique 870, IRISA, Rennes, France, Octobre 1994.

[GLQ+97] Gautier T., Le Guernic P., Quinton P., Rajopadhye S., Risset T. et Smarandache I. Le projet CAIRN : vers la conception d'architectures à partir de SIGNAL et ALPHA. *Collection Technique et Scientifique des Télécommunications*, 1997.

[Har87] Harel D. STATECHARTS : a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, Juin 1987.

[HCRP91] Halbwachs N., Caspi P., Raymond P. et Pilaud D. The Synchronous Data Flow Programming Language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, Septembre 1991.

[HN96] Harel D. et Naamad A. The Statemate Semantics of STATECHARTS. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[HP85] Harel D. et Pnueli A. On the development of reactive systems. Dans *Logics and Models of Concurrent Systems*, Apt K.R., éditeur, vol. 13 de *NATO Asi Series, Computer and Systems Sciences*, pp. 477–498. Springer Verlag, New York, 1985.

[HPSS87] Harel D., Pnueli A., Schmidt J.P. et Sherman R. On the Formal Semantics of STATECHARTS. Dans *Proceedings of the Symposium on Logic in Computer Science*, pp. 54–64, 1987.

[Inc90] Inc. i-Logix. Languages of statemate. 22 Third Avenue, Burlington, MA 01803, USA, 1990.

[INM87] INMOS. *OCCAM2 Reference Manual.* Prentice Hall, 1987.

[JDA96] Jagannathan R., Dodd C. et Agi I. GLU: A High-Level System for Granular Data-Parallel Programming. Dans *Concurrency: Practice and Experience*. John Wiley and Sons, 1996.

[KL93] Kalavade A. et Lee E.A. A Hardware-Software Codesign Methodology for DSP Applications. *IEEE Design & Test of Computers*, 10(3):16–28, Septembre 1993.

[KL96] Kountouris A. et Le Guernic P. Profiling of SIGNAL programs and its application in the timing evaluation of design implementations. Dans *Hardware-Software Cosynthesis for Reconfigurable Systems*. Hewlett Packard Laboratories, Bristol, Colloquium organised by Professional Group C2 (Hardware and systems engineering), Février 1996.

[KMR93] Kerihuel A., McConnell R. et Raimbault F. Tools for correct DSP synchronization. Rapport technique 717, INRIA, France, Avril 1993.

[KMR94] Kerihuel A., McConnell R. et Rajopadhye S. VSDF : Synchronous data flow for VLSI. Rapport technique 843, INRIA, France, Juin 1994.

[KMW67] Karp R.M., Miller R.E. et Winograd S. The organization of computations for uniform recurrence equations. *Journal of the Association for Computing Machinery*, 14(3):563–590, Juillet 1967.

[KW96] Kountouris A. et Wolinski C. A real-time HW/SW co-design approach based on the SIGNAL language and its environment. Rapport technique, IRISA, Rennes, France, Octobre 1996.

[KW97] Kountouris A. et Wolinski C. A Method for the Generation of HDL Code at the RTL-level from a High-Level Formal Specification Language. Dans *Proceedings of MWSCAS, Sacramento, USA*. IEEE Computer Society Press, Août 1997.

[LBBG86] Le Guernic P., Benveniste A., Bournai P. et Gautier T. SIGNAL - a Data Flow-Oriented Language for Signal Processing. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 34(2), Avril 1986.

[Le 97] Le Moenner P. *Contribution à la réalisation d'une chaîne complète pour la synthèse de circuits réguliers.* Thèse de doctorat, Université de Rennes 1, France, Novembre 1997.

[Lee91a] Lee E.A. Consistency in Dataflow Graphs. Dans *Proceedings of the International Conference on Application Specific Array Processors, Synthesis and Verification*, pp. 355–369, 1991.

[Lee91b] Lee E.A. Consistency in Dataflow Graphs. *IEEE Transactions on Parallel and Distributed Systems*, 2(2), Avril 1991.

[LG91] Le Guernic P. et Gautier T. Data-Flow to von Neumann: the SIGNAL Approach. Dans *Advanced Topics in Data-Flow Computing*, Gaudiot J.-L. et Bic L., éditeurs, pp. 413–438. Prentice Hall, 1991.

[LGLL91] Le Guernic P., Gautier T., Le Borgne M. et Le Maire C. Programming Real-Time Applications with SIGNAL. *Proceedings of the IEEE*, 79(9):1321–1336, Septembre 1991.

[LHG+89] Lee E.A., Ho W., Goei E.E., Bier J.C. et Bhattacharyya S. Gabriel : A Design Environnement for DSP. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(11), Novembre 1989.

[Lip81] Lipson J.D. *Elements of Algebra and Algebraic Computing.* The Benjamin/Cummings Publishing Company, 1981.

[LLGL91] Le Guernic P., Le Borgne M., Gautier T. et Le Maire C. Programming real time applications with SIGNAL. Rapport technique 1446, INRIA, France, Juin 1991.

[LM87a] Lee E.A. et Messerschmitt D.G. Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing. *IEEE Transactions on Computers*, C-36(1):24–35, Janvier 1987.

[LM87b] Lee E.A. et Messerschmitt D.G. Synchronous Data Flow. *Proceedings of the IEEE*, 75(9):1235–1245, Septembre 1987.

[LM93] Lee E.A. et Messerschmitt D. An overview of the ptolemy project. Rapport technique, Department of Electrical Engineering and Computer Science, University of California Berkeley, Janvier 1993.

[LM97] Le Borgne M. et Marchand H. Partial order control and optimal control of discrete event systems modeled as polynomial dynamical systems over Galois fields. Rapport technique 3273, INRIA, France, Octobre 1997.

[LMQ90] Leverge H., Mauras Ch. et Quinton P. A language oriented approach to the design of systolic chips. Dans *Proceedings of the International Workshop on Algorithms and Parallel VLSI Architectures, Pont-à-Mousson, France*, Juin 1990.

[LSSS91] Lavarenne C., Seghrouchni O., Sorel Y. et Sorine M. The Syndex software environment for real-time distributed systems design and implementation. Dans *Proceedings of the ECC91 European Control Conference, Grenoble, France*, pp. 1684–1689, Juillet 1991.

[M.94] Von Der Beeck M. A Comparison of STATECHARTS Variants. Dans *Proceedings of the Third International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, Lübeck, Germany*, Langmaack H., Roever W.P. et Vytopil J., éditeurs, vol. 863 de *LNCS*, pp. 128–148. Springer Verlag, Septembre 1994.

[Maf93] Maffeïs O. *Ordonnancements de graphes de flots synchrones; application à la mise en œuvre de SIGNAL.* Thèse de doctorat, Université de Rennes 1, France, Janvier 1993.

[Mar91] Maraninchi F. The Argos language: Graphical Representation of Automata and Description of Reactive Systems. Dans *Proceedings of the IEEE Workshop on Visual Languages, Kobe, Japon*, Octobre 1991.

[Mar97] Marchand H. *Méthodes de synthèse d'automatismes décrits par des systèmes à événements discrets finis.* Thèse de doctorat, Université de Rennes 1, France, Octobre 1997.

[Mau89a] Mauras Ch. *ALPHA: un langage equationnel pour la conception et programmation d'architectures parallèles synchrones.* Thèse de doctorat, Université de Rennes 1, France, 1989.

[Mau89b] Mauras Ch. Définition de ALPHA : un langage pour la programmation systolique. Rapport technique 482, INRIA, France, Juin 1989.

[McC94] McConnell R. *Systèmes VLSI synchrones périodiques : modélisation et application.* Thèse de doctorat, Université de Rennes 1, France, Octobre 1994.

[MCL92] Maffeïs O., Chéron B. et Le Guernic P. Transformations du graphe des programmes SIGNAL. Rapport technique 1574, INRIA, France, Janvier 1992.

[Pur93] Puri A. Video coding using the MPEG-2 compression standard. Dans *Proceedings of the International Society for Optical Engineering - Visual Communications and Image Processing, Cambridge, MA, USA*, Haskell B.G. et Hang H.-M., éditeurs, vol. 2094, pp. 1701–1713, Novembre 1993.

[QR95] Quinton P. et Rajopadhye S. Deriving Imperative Code from Functional Programs. Dans *FPCA*, 1995.

[QRR96a] Quinton P., Rajopadhye S. et Risset T. Extension of the ALPHA language to recurrences on sparse periodic domains. Rapport technique 1017, IRISA, Rennes, France, Juillet 1996.

[QRR96b] Quinton P., Rajopadhye S. et Risset T. On manipulating $\mathbb{Z}$-polyhedra. Rapport technique 1016, IRISA, Rennes, France, Juillet 1996.

[QRW95a] Quinton P., Rajopadhye S. et Wilde D. Derivation of Data Parallel Code from a Functional Program. Dans *IPPS*, 1995.

[QRW95b] Quinton P., Rajopadhye S. et Wilde D. The Naive Execution of Affine Recurrence Equations. Dans *ASAP*, 1995.

[Qui84] Quinton P. Automatic synthesis of systolic arrays from recurrent uniform equations. Dans *Proceedings of the 11th Annual International Symposium on Computer Architecture, Ann Arbor, USA*, pp. 208–214, Juin 1984.

[QV89] Quinton P. et Van Dongen V. The mapping of linear recurrence equations on regular arrays. *Journal of VLSI Signal Processing*, 1(2):95–113, 1989.

[Raj86] Rajopadhye S.V. *Synthesis, Optimization and Verification of Systolic Architectures.* Thèse de doctorat, University of Utah, Salt Lake City, Utah 84112, Décembre 1986.

[Raj93] Rajopadhye S.V. Analysis of affine communication specification. Dans *Proceedings of the IEEE Symposium on Parallel and Distributed Processing, Dallas, Texas, USA*, Décembre 1993.

[Sam71] Samuel P. About euclidean rings. *Journal of Algebra*, 19:282–301, 1971.

[San80] Sanderson J.G. *A Relational Theory of Computing*, vol. 82 de *LNCS*. Springer Verlag, 1980.

[Sch92] Scharff X. Analyse et mise en œuvre de la chaîne de codage 34 mbits. Rapport technique, IRISA, Rennes, France, Septembre 1992.

[SJA93] Salinas M.H., Johnson B.W. et Aylor J.H. Implementation-Independent Model of an Instruction Set Architecture in VHDL. *IEEE Design & Test of Computers*, 10(3):42–54, Septembre 1993.

[Ski91] Skillicorn D. Stream Languages and Data-Flow. Dans *Advanced Topics in Data-Flow Computing*, Gaudiot J.-L. et Bic L., éditeurs, pp. 439–454. Prentice Hall, 1991.

[SL97a] Smarandache I. et Le Guernic P. Affine Transformations in SIGNAL and Their Applications in the Specification and Validation of Real-Time Systems. Dans *Transformation–Based Reactive Systems Development, Proceedings of the 4th International AMAST Workshop on Real–Time Systems and Concurrent and Distributed Software, Palma, Espagne*, vol. 1231 de *LNCS*. Springer Verlag, Mai 1997.

[SL97b] Smarandache I. et Le Guernic P. A canonical form for affine relations in SIGNAL. Rapport technique 3097, INRIA, France, Février 1997.

[SN97] Scholz P. et Nazareth D. Communication Concepts for STATECHARTS : A Semantic Foundation. Dans *Transformation–Based Reactive Systems Development, Proceedings of the 4th International AMAST Workshop on Real–Time Systems and Concurrent and Distributed Software, Palma, Espagne*, vol. 1231 de *LNCS*, pp. 126–140. Springer Verlag, Mai 1997.

[TAS93] Thomas D.E., Adams J.K. et Schmit H. A Model and Methodology for Hardware-Software Codesign. *IEEE Design & Test of Computers*, 10(3):6–15, Septembre 1993.

[Wil93] Wilde D. A library for doing polyhedral operations. Rapport technique 785, IRISA, Rennes, France, Décembre 1993.

[Wil94] Wilde D. The ALPHA language. Rapport technique 827, IRISA, Rennes, France, Mai 1994.

[WP94] Whiting P.G. et Pascoe R.S.V. A History of Data-Flow Languages. *IEEE Annals of the History of Computing*, 16(4), 1994.

[WS94] Wilde D.K. et Sié O. Regular array synthesis using ALPHA. Rapport technique 829, IRISA, Rennes, France, Mai 1994.

Annexe A

Preuve du Lemme 4, section 7.1.2.2

Preuve

Étant donnés les triplets (n, φ, d) et (n', φ', d'), les fonctions de densification $f_1, f_2, f'_1, f'_2 : \mathbb{N} \rightarrow \mathbb{N}$ sont définies par les expressions suivantes :

$$\begin{aligned} f_1 &= \lambda\ t.(nt + max(0, -\varphi)) \\ f_2 &= \lambda\ t.(dt + max(0, \varphi)) \\ f'_1 &= \lambda\ t.(n't + max(0, -\varphi')) \\ f'_2 &= \lambda\ t.(d't + max(0, \varphi')) \end{aligned}$$

Associés à ces fonctions, on considère les ensembles définis comme suit ($\forall\ t \in \mathbb{N}^*$) :

$$\begin{aligned} A_t &= \{s \in \mathbb{N}\ /\ f_1(s) \leq f_2(t-1)\}, \qquad & A'_t &= \{s \in \mathbb{N}\ /\ f'_1(s) \leq f'_2(t-1)\} \\ B_t &= \{s \in \mathbb{N}\ /\ f_2(s) \leq f_1(t-1)\}, \qquad & B'_t &= \{s \in \mathbb{N}\ /\ f'_2(s) \leq f'_1(t-1)\} \end{aligned}$$

Les ensembles A_t, A'_t, B_t, B'_t ont été choisis pour qu'ils vérifient, pour tout $t \in \mathbb{N}$,

$$\begin{aligned} \mid A_t \mid &= \mathcal{A}^F_y(x)_{(n,\varphi,d)}(t), \qquad & \mid A'_t \mid &= \mathcal{A}^F_y(x)_{(n',\varphi',d')}(t) \\ \mid B_t \mid &= \mathcal{A}^F_x(y)_{(n,\varphi,d)}(t), \qquad & \mid B'_t \mid &= \mathcal{A}^F_x(y)_{(n',\varphi',d')}(t) \end{aligned}$$

A partir des expressions des fonctions f_1, f_2 et f'_1, f'_2 on obtient :

$$\begin{aligned} A_t &= \{s \in \mathbb{N}\ /\ ns \leq d(t-1) + \varphi\} \qquad & A'_t &= \{s \in \mathbb{N}\ /\ n's \leq d'(t-1) + \varphi'\} \\ B_t &= \{s \in \mathbb{N}\ /\ ds + \varphi \leq n(t-1)\} \qquad & B'_t &= \{s \in \mathbb{N}\ /\ d's + \varphi' \leq n'(t-1)\} \end{aligned}$$

1. On démontre d'abord l'équivalence $b \Leftrightarrow c$.

Supposons $\mathcal{A}^F_y(x)_{(n,\varphi,d)} = \mathcal{A}^F_y(x)_{(n',\varphi',d')}$. Ceci implique :

$$d(t-1) + \varphi < 0 \Leftrightarrow d'(t-1) + \varphi' < 0,\ \ \forall t \in \mathbb{N}^*$$

$$\left\lfloor \frac{d(t-1)+\varphi}{n} \right\rfloor = \left\lfloor \frac{d'(t-1)+\varphi'}{n'} \right\rfloor,\ \ \forall t \in \mathbb{N}^*, d(t-1) + \varphi \geq 0$$

En utilisant les relations ci-dessus, on propose une preuve en 3 étapes :

1. Premièrement, on démontre que $A_t = A'_t, \forall\ t \in \mathbb{N}^*$.
2. A partir de 1. on démontre $B_t = B'_t, \forall\ t \in \mathbb{N}^*$.
3. Finalement, $B_t = B'_t, \forall\ t \in \mathbb{N}^* \ \Rightarrow\ \mathcal{A}_x(y)_{(n,\varphi,d)} = \mathcal{A}_x(y)_{(n',\varphi',d')}$.

1. Preuve de $A_t = A'_t, \forall\, t \in \mathbb{N}^*$.
   Soit $t \in \mathbb{N}^*$ un entier strictement positif.
   – Dans le cas $d(t-1)+\varphi < 0$ il résulte :

$$\left.\begin{array}{l} d'(t-1)+\varphi' < 0 \quad \Rightarrow \quad A'_t = \emptyset \\ \text{et} \\ A_t = \emptyset \end{array}\right\} \Rightarrow A_t = A'_t$$

   – Dans le cas $d(t-1)+\varphi \geq 0$ (équivalent à $d'(t-1)+\varphi' \geq 0$), il résulte $A_t \subseteq A'_t$ comme suit :

$$\begin{array}{lll} \forall\, s \in A_t & \Rightarrow & ns \leq d(t-1)+\varphi \\ & \Rightarrow & s \leq \dfrac{d(t-1)+\varphi}{n} \\ & \overset{s\in\mathbb{N}}{\Rightarrow} & 0 \leq s \leq \left[\dfrac{d(t-1)+\varphi}{n}\right] \\ & \Rightarrow & (s \in \mathbb{N}) \bigwedge (0 \leq s \leq \left[\dfrac{d'(t-1)+\varphi'}{n'}\right] \leq \dfrac{d'(t-1)+\varphi'}{n'}) \\ & \Rightarrow & (0 \leq s \leq \dfrac{d'(t-1)+\varphi'}{n'}) \bigwedge (s \in \mathbb{N}) \\ & \Rightarrow & s \in A'_t \end{array}$$

   $A'_t \subseteq A_t$ peut être démontré de manière similaire.
   En conclusion, $A_t = A'_t, \forall\, t \in \mathbb{N}^*$ (preuve étape 1.)
2. Preuve de $B_t = B'_t, \forall\, t \in \mathbb{N}^*$.
   On démontre premièrement l'équivalence suivante :

$$n(t-1)-\varphi < 0 \;\Leftrightarrow\; n'(t-1)-\varphi' < 0,\;\; \forall t \in \mathbb{N}^*$$

   Preuve par contradiction : supposons qu'il existe $s \in \mathbb{N}^*$ qui vérifie $n(s-1)-\varphi < 0$ and $n'(s-1)-\varphi' \geq 0$.

$$\begin{array}{lll} n(s-1)-\varphi < 0 & \Rightarrow & \forall\, t \in \mathbb{N}^*,\; n(s-1)-\varphi < d(t-1) \\ & \Rightarrow & \forall\, t \in \mathbb{N}^*,\; (s-1) \in A_t \\ & \overset{A_t=A'_t}{\Rightarrow} & \forall\, t \in \mathbb{N}^*,\; (s-1) \in A'_t \\ & \Rightarrow & \forall\, t \in \mathbb{N}^*,\; n'(s-1) \leq d'(t-1)+\varphi' \\ & \Rightarrow & \forall\, t \in \mathbb{N}^*,\; n'(s-1)-\varphi' \leq d'(t-1) \\ & \Rightarrow & n'(s-1)-\varphi' \leq 0 \end{array}$$

   $s \in \mathbb{N}^*$ satisfait $n'(s-1)-\varphi' \leq 0$ et $n'(s-1)-\varphi' \geq 0$. Il résulte $n'(s-1)-\varphi' = 0$.
   Comme $(x,y) \in \mathcal{R}^F_{(n,\varphi,d)} \cap \mathcal{R}^F_{(n',\varphi',d')}$, $(\exists)\; g, g' : \mathbb{N} \to \mathbb{N}$, croissantes sur $\mathbb{N}$ et strictement croissantes sur $f_1(\mathbb{N}) \cup f_2(\mathbb{N})$, respectivement $f'_1(\mathbb{N}) \cup f'_2(\mathbb{N})$, telles que

$$\begin{array}{l} F = (\;(g \circ f_1) \uparrow \Pi_{\{x\}}(F)\;) \cdot (\;(g \circ f_2) \uparrow \Pi_{\{y\}}(F)\;) \cdot \\ \cdot (\;(g' \circ f'_1) \uparrow \Pi_{\{x\}}(F)\;) \cdot (\;(g' \circ f'_2) \uparrow \Pi_{\{y\}}(F)\;) \cdot F \end{array}$$

   et

$$\begin{array}{l} \forall\, t,s \in \mathbb{N},\; f_1(t) = f_2(s) \;\Rightarrow\; x_{(F)}(t) = y_{(F)}(s) \\ \forall\, t,s \in \mathbb{N},\; f'_1(t) = f'_2(s) \;\Rightarrow\; x_{(F)}(t) = y_{(F)}(s) \end{array}$$

   L'unicité des fonctions de densification associées à x et y dans F implique : $(g \circ f_1)(\mathbb{N}) = (g' \circ f'_1)(\mathbb{N})$ et $(g \circ f_2)(\mathbb{N}) = (g' \circ f'_2)(\mathbb{N})$.

On a alors la suite d'équivalences :

$$\begin{aligned}\forall\, t,s \in \mathbb{N},\ nt = ds+\varphi &\Leftrightarrow f_1(t)=f_2(s)\\ &\Leftrightarrow (g\circ f_1)(t)=(g\circ f_2)(s)\\ &\Leftrightarrow (g'\circ f_1')(t)=(g'\circ f_2')(s)\\ &\Leftrightarrow f_1'(t)=f_2'(s)\\ &\Leftrightarrow \forall\, t,s\in\mathbb{N},\ n't=d's+\varphi'\end{aligned}$$

En conséquence, $s \in \mathbb{N}^*$ avec la propriété $n'(s-1)-\varphi'=0$ vérifie aussi $n(s-1)-\varphi=0$ ce qui est en contradiction avec la supposition initiale $n(s-1)-\varphi<0$.
En supposant que $\exists s\in\mathbb{N}^*,\ n(s-1)-\varphi<0$ et $n'(s-1)-\varphi'\geq 0$ on a obtenu un contradiction; on peut conclure :

$$n(t-1)-\varphi<0 \ \Leftrightarrow\ n'(t-1)-\varphi'<0,\quad \forall t\in\mathbb{N}^*$$

Soit $t\in\mathbb{N}^*$ un entier strictement positif.

– Dans le cas $n(t-1)-\varphi<0$ il résulte :

$$\left.\begin{array}{l} n'(t-1)-\varphi'<0 \quad\Rightarrow\quad B_t'=\emptyset\\ \text{et}\\ B_t=\emptyset\end{array}\right\}\Rightarrow B_t=B_t'$$

– Dans le cas $n(t-1)-\varphi\geq 0$ (équivalent à $n'(t-1)-\varphi'\geq 0$), il résulte $B_t\subseteq B_t'$ comme suit :

$$\begin{array}{rcl} \text{Soit } s\in B_t & \Rightarrow & ds\leq n(t-1)-\varphi \qquad\qquad (\text{A.1})\\ \text{Si } s\notin B_t' & \Rightarrow & d's+\varphi'>n'(t-1)\\ & \overset{t\in\mathbb{N}^*}{\Rightarrow} & d's+\varphi'>0 \ \text{ and }\ (t-1)\in A_{s+1}'\neq\emptyset\\ & \overset{A_{s+1}'=A_{s+1}}{\Rightarrow} & (t-1)\in A_{s+1}\\ & \Rightarrow & n(t-1)\leq ds+\varphi\\ & \overset{n'(t-1)\neq d's+\varphi'}{\Rightarrow} & nt<ds+\varphi \end{array}$$

qui est en contradiction avec l'hypothèse (A.1).
Il a été démontré que $\forall\ s\in\mathbb{N},\ s\in B_t\ \Rightarrow\ s\in B_t'$, i.e. $B_t\subseteq B_t'$. Un raisonnement similaire conduit à $B_t'\subseteq B_t$.

Il a été démontré que $B_t'=B_t,\forall t\in\mathbb{N}^*$. (preuve étape 2.)

3. A partir de 2. on obtient la suite d'implications :

$$\begin{aligned} B_t'=B_t,\forall t\in\mathbb{N}^* &\Rightarrow\ |\,B_t'\,| = |\,B_t\,|\,,\forall t\in\mathbb{N}^*\\ &\Rightarrow\ \mathcal{A}_x(y)_{(n,\varphi,d)}=\mathcal{A}_x(y)_{(n',\varphi',d')}\end{aligned}$$

L'implication

$$\mathcal{A}_y(x)_{(n,\varphi,d)}=\mathcal{A}_y(x)_{(n',\varphi',d')}\ \Rightarrow \mathcal{A}_x(y)_{(n,\varphi,d)}=\mathcal{A}_x(y)_{(n',\varphi',d')}$$

a été prouvée valide; un raisonnement similaire permet de prouver l'implication inverse. (preuve $b\Leftrightarrow c$)

2. A partir de $b\Leftrightarrow c$ on peut prouver les deux dernières équivalences. Nous exemplifions avec la première :

$$\left.\begin{array}{l}\left.\begin{array}{l} a\Rightarrow(b\bigwedge c)\\ c\Rightarrow b\end{array}\right\} \quad\Rightarrow\quad a\Rightarrow b\\ (b\Rightarrow c)\Rightarrow(b\Rightarrow b\bigwedge c)\quad\Rightarrow\quad b\Rightarrow a\end{array}\right\}\ \Rightarrow\ a\Leftrightarrow b$$

3. Cas $a\Leftrightarrow c$: preuve similaire au cas précédent.

□

Annexe B

Preuve de la Proposition 1, section 7.1.3

Preuve

1. Dans l'hypothèse $\mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')}$ on prouve que (a) et (b) sont valides en exclusion mutuelle.

   Pour tout $(x,y) \in \mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')}$ $(\exists)$ $g, g' : \mathbb{N} \to \mathbb{N}$, croissantes sur $\mathbb{N}$ et strictement croissantes sur $f_1(\mathbb{N}) \cup f_2(\mathbb{N})$, respectivement $f'_1(\mathbb{N}) \cup f'_2(\mathbb{N})$, telles que

   $$F = (\ (g \circ f_1) \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ (g \circ f_2) \uparrow \Pi_{\{y\}}(F)\ ) \cdot$$
   $$\cdot (\ (g' \circ f'_1) \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ (g' \circ f'_2) \uparrow \Pi_{\{y\}}(F)\ ) \cdot F$$

   et

   $$\forall\ t, s \in \mathbb{N},\ f_1(t) = f_2(s)\ \Rightarrow\ x_{(F)}(t) = y_{(F)}(s)$$
   $$\forall\ t, s \in \mathbb{N},\ f'_1(t) = f'_2(s)\ \Rightarrow\ x_{(F)}(t) = y_{(F)}(s)$$

   avec $f_1, f'_1, f_2, f'_2 : \mathbb{N} \to \mathbb{N}$, $f_1(t) = nt + \ max(0, -\varphi), f'_1(t) = n't + \ max(0, -\varphi')$ et $f_2(t) = dt + \ max(0, \varphi), f'_2(t) = d't + \ max(0, \varphi')$.

   Les triplets (n, φ, d) et (n', φ', d') doivent satisfaire les conditions nécessaires et suffisantes pour que l'égalité des fonctions de densification associées à x et y dans le flot F soit vérifiée: $(g \circ f_1)(\mathbb{N}) = (g' \circ f'_1)(\mathbb{N})$ et $(g \circ f_2)(\mathbb{N}) = (g' \circ f'_2)(\mathbb{N})$. A partir de ce résultat et en utilisant les définitions des fonctions d'occurrence relatives des signaux, on obtient l'égalité des fonctions d'occurrence associées à x et y dans le flot F :

   $$\mathcal{A}^F_x(y)_{(g \circ f_1, g \circ f_2)} = \mathcal{A}^F_x(y)_{(g' \circ f'_1, g' \circ f'_2)}$$
   $$\mathcal{A}^F_y(x)_{(g \circ f_1, g \circ f_2)} = \mathcal{A}^F_y(x)_{(g' \circ f'_1, g' \circ f'_2)}$$

   ce qui est équivalent à:

   $$\mathcal{A}^F_x(y)_{(n,\varphi,d)} = \mathcal{A}^F_x(y)_{(n',\varphi',d')}$$
   $$\mathcal{A}^F_y(x)_{(n,\varphi,d)} = \mathcal{A}^F_y(x)_{(n',\varphi',d')}$$

   D'après le lemme 3 et cas 3. du Lemme 4, les triplets (n, φ, d) et (n', φ', d') doivent satisfaire:

   $$\left[\frac{dt+\varphi}{n}\right] = \left[\frac{d't+\varphi'}{n'}\right], \forall\ t \in \mathbb{N},\ \ dt + \varphi \geq 0$$

   (l'égalité de $\mathcal{A}^F_y(x)_{(n,\varphi,d)}$ et $\mathcal{A}^F_y(x)_{(n',\varphi',d')}$ impose l'équivalence suivante: $dt + \varphi < 0 \Leftrightarrow d't + \varphi' < 0,\ \forall\ t \in \mathbb{N}$).

Soit $t' \in \mathbb{N}$ tel que $dt + \varphi > 0, \forall\, t \geq t'$.

Avec la relation précédente et les propriétés de la partie entière $[n] \leq n < [n] + 1$, on a les équivalences :

$$\begin{cases} \frac{dt+\varphi-n}{d't+\varphi'} < \frac{n}{n'}, & \forall\, t \geq t' \\ \frac{d't+\varphi'-n'}{dt+\varphi} < \frac{n'}{n}, & \forall\, t \geq t' \end{cases} \Leftrightarrow \begin{cases} \lim_{t\to+\infty} \frac{dt+\varphi-n}{d't+\varphi'} \leq \frac{n}{n'} \\ \lim_{t\to+\infty} \frac{d't+\varphi'-n'}{dt+\varphi} \leq \frac{n'}{n} \end{cases}$$
$$\Leftrightarrow \begin{cases} \frac{d}{d'} \leq \frac{n}{n'} \\ \frac{d'}{d} \leq \frac{n'}{n} \end{cases}$$
$$\Leftrightarrow nd' = n'd$$

D'autre part, si $pgcd(n,d)$ est un diviseur de φ, on obtient les équivalences :

$$\begin{array}{rcl} pgcd(n,d) \mid \varphi & \Leftrightarrow & \text{l'équation Diophantine } ns = dt + \varphi \text{ admet au moins} \\ & & \text{une solution } (s,t) \in \mathbb{N}^2 \\ & \Leftrightarrow & \exists\, (s,t) \in \mathbb{N}^2,\ f_1(s) = f_2(t) \\ & \Leftrightarrow & \exists\, (s,t) \in \mathbb{N}^2,\ (g \circ f_1)(s) = (g \circ f_2)(t) \\ & \Leftrightarrow & \exists\, (s,t) \in \mathbb{N}^2,\ (g' \circ f_1')(s) = (g' \circ f_2')(t) \\ & \Leftrightarrow & \exists\, (s,t) \in \mathbb{N}^2,\ f_1'(s) = f_2'(t) \\ & \Leftrightarrow & \text{la solution } (s,t) \in \mathbb{N}^2 \text{ vérifie aussi l'équation Diophantine} \\ & & n's = d't + \varphi' \\ & \Leftrightarrow & pgcd(n',d') \mid \varphi' \end{array}$$

Dans le cas $pgcd(n,d) \mid \varphi$, les équations Diophantine $ns = dt + \varphi$ et $n's = d't + \varphi'$ ont une solution commune $(s_0, t_0) \in \mathbb{N}^2$, i.e. $ns_0 = dt_0 + \varphi$ and $n's_0 = d't_0 + \varphi'$. La première équation implique $s_0 = \frac{dt_0+\varphi}{n}$; en remplaçant s_0 par son expression dans la deuxième équation, on obtient :

$$\frac{dt_0 + \varphi}{n} = \frac{d't_0 + \varphi'}{n'} \Rightarrow dn't_0 + \varphi n' = nd't_0 + \varphi' n \overset{nd'=dn'}{\Rightarrow} n\varphi' = n'\varphi$$

2. Supposons maintenant que (n, φ, d) et (n', φ', d') satisfont une et seulement une des conditions (a) et (b). Nous prouvons $\mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')}$ par inclusion réciproque.

   Soit $(x,y) \in \mathcal{R}^F_{(n,\varphi,d)}$. On prouve $(x,y) \in \mathcal{R}^F_{(n',\varphi',d')}$.

   Dans la suite, les fonctions f_1, f_2, f_1', f_2' seront définies par les expressions suivantes :

$$\begin{array}{l} f_1, f_1', f_2, f_2' : \mathbb{N} \to \mathbb{N}, \\ f_1(t) = nt + max(0, -\varphi), f_1'(t) = n't + max(0, -\varphi') \\ f_2(t) = dt + max(0, \varphi), f_2'(t) = d't + max(0, \varphi') \end{array}$$

   (a) On se situe dans le cas (a) de la Proposition 1, i.e. (n, φ, d) et (n', φ', d') satisfont les contraintes :

$$\begin{array}{l} (\ pgcd(n,d) \mid \varphi) \bigwedge (\ pgcd(n',d') \mid \varphi') \\ nd' = n'd \\ n\varphi' = n'\varphi \end{array}$$

   – Cas 1 : $\varphi = 0$. Il résulte $\varphi' = 0$.
     On note $\frac{n}{n'} = \frac{d}{d'} \overset{not}{=} k > 0$. Les expressions des fonctions f_1, f_2, f_2', f_2' conduisent à

$$\begin{array}{lllllll} f_1 & = & \lambda\, t.(nt) & = & \lambda\, t.(kn't) & = & kf_1' \\ f_2 & = & \lambda\, t.(dt) & = & \lambda\, t.(kd't) & = & kf_2' \end{array}$$

– Cas 2 : $\varphi \neq 0$; il résulte $\varphi' \neq 0$.
On note $\frac{n}{n'} = \frac{d}{d'} = \frac{\varphi}{\varphi'} \stackrel{not}{=} k > 0$. Les expressions des fonctions f_1, f_2, f_1', f_2' conduisent dans ce cas à

$$\begin{array}{lllllll} f_1 & = & \lambda\, t.(nt + max(0, -\varphi)) & = & \lambda\, t.(kn't + max(0, -k\varphi')) & = & kf_1' \\ f_2 & = & \lambda\, t.(dt + max(0, \varphi)) & = & \lambda\, t.(kd't + max(0, k\varphi')) & = & kf_2' \end{array}$$

En conclusion, dans tous les cas : $f_1 = kf_1'$ et $f_2 = kf_2'$.
Comme $x\ \mathcal{R}^F_{(n,\varphi,d)}\ y$, $(\exists)\ g : \mathbb{N} \to \mathbb{N}$, croissante sur $\mathbb{N}$ et strictement croissante sur $f_1(\mathbb{N}) \cup f_2(\mathbb{N})$, telle que x et y soient (f_1, f_2)-codensifiés (voir section 7.1.1) :

$$F = (\ (g \circ f_1) \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ (g \circ f_2) \uparrow \Pi_{\{y\}}(F)\ ) \cdot F \tag{B.1}$$

et

$$\forall\ t, s \in \mathbb{N},\ f_1(t) = f_2(s)\ \Rightarrow\ x_{(F)}(t) = y_{(F)}(s) \tag{B.2}$$

On choisit $g' : \mathbb{N} \to \mathbb{N}, g'(s) = g(ks), \forall s \in \mathbb{N}$. On va démontrer que g', f_1' et f_2' respectent les conditions pour que x et y soient (f_1', f_2')-codensifiés, ce qui va induire $x\ \updownarrow^F_{(n',\varphi',d')}\ y$.

1) Soit $t_1, t_2 \in \mathbb{N}, t_1 < t_2$.

$$\begin{array}{lll} t_1 < t_2 & \stackrel{k>0}{\Rightarrow} & kt_1 < kt_2 \\ & \Rightarrow & g(kt_1) \leq g(kt_2) \\ & \Rightarrow & g'(t_1) \leq g'(t_2) \end{array}$$

g' est croissante sur $\mathbb{N}$.

2) On prouve que g' est strictement croissante sur $f_1'(\mathbb{N}) \cup f_2'(\mathbb{N})$.
Tout d'abord, on a pour tout $i \in \{1, 2\}$:

$$\forall t \in \mathbb{N},\ g(f_i(t)) = g(kf_i'(t)) = g'(f_i'(t))$$

Il résulte $g' \circ f_1' = g \circ f_1, g' \circ f_2' = g \circ f_2$, et comme $g \circ f_1, g \circ f_2$ sont strictement croissantes, il en résulte $g' \circ f_1', g' \circ f_2'$ strictement croissantes, ce qui implique g' strictement croissante sur $f_1'(\mathbb{N})$ et strictement croissante sur $f_2'(\mathbb{N})$.

Il est nécessaire de prouver g' strictement croissante sur $f_1'(\mathbb{N}) \cup f_2'(\mathbb{N})$.
Soit $t_1 \in f_1'(\mathbb{N}), t_2 \in f_2'(\mathbb{N}), t_1 < t_2$. Il existe $s_1, s_2 \in \mathbb{N}$ tels que $t_1 = f_1'(s_1), t_2 = f_2'(s_2)$ et $f_1'(s_1) < f_2'(s_2)$.

$$\begin{array}{lll} f_1'(s_1) < f_2'(s_2) & \stackrel{k>0}{\Rightarrow} & kf_1'(s_1) < kf_2'(s_2) \\ & \Rightarrow & f_1(s_1) < f_2(s_2) \\ & \stackrel{g \text{ str. cr. sur } f_1(\mathbb{N}) \cup f_2(\mathbb{N})}{\Rightarrow} & (g \circ f_1)(s_1) < (g \circ f_2)(s_2) \\ & \Rightarrow & (g' \circ f_1')(s_1) < (g' \circ f_2')(s_2) \\ & \Rightarrow & g'(t_1) < g'(t_2) \end{array}$$

Soit $t_1 \in f_2'(\mathbb{N}), t_2 \in f_1'(\mathbb{N}), t_1 < t_2$. Il existe $s_1, s_2 \in \mathbb{N}$ tels que $t_1 = f_2'(s_1), t_2 = f_1'(s_2)$ et $f_2'(s_1) < f_1'(s_2)$. Il résulte comme précédemment que $g'(t_1) < g'(t_2)$.

En conclusion, g' est strictement croissante sur $f_1'(\mathbb{N}) \cup f_2'(\mathbb{N})$.

3) Par définition, F s'écrit :

$$F\ =\ (\ (g \circ f_1) \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ (g \circ f_2) \uparrow \Pi_{\{y\}}(F)\ ) \cdot F$$

En utilisant les égalités $g \circ f_1 = g' \circ f_1'$ et $g \circ f_2 = g' \circ f_2'$, F devient :

$$F \quad = \quad (\ (g' \circ f'_1) \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ (g' \circ f'_2) \uparrow \Pi_{\{y\}}(F)\ ) \cdot F$$

En utilisant B.2, on obtient pour f'_1, f'_2 la suite d'implications suivante :

$$\begin{array}{rcl} \forall\ t,s \in \mathbb{N},\ f'_1(t) = f'_2(s) & \Rightarrow & kf'_1(t) = kf'_2(s) \\ & \Rightarrow & f_1(t) = f_2(s) \\ & \Rightarrow & x_{(F)}(t) = y_{(F)}(s) \end{array}$$

On a exhibé $g' : \mathbb{N} \rightarrow \mathbb{N}$, croissante sur $\mathbb{N}$ et strictement croissante sur $f'_1(\mathbb{N}) \cup f'_2(\mathbb{N})$, telle que

$$F = (\ (g' \circ f'_1) \uparrow \Pi_{\{x\}}(F)\ ) \cdot (\ (g' \circ f'_2) \uparrow \Pi_{\{y\}}(F)\ ) \cdot F$$

et

$$\forall\ t,s \in \mathbb{N},\ f'_1(t) = f'_2(s) \ \Rightarrow\ x_{(F)}(t) = y_{(F)}(s)$$

avec $f'_1 = \lambda\ t.(n't + max(0, -\varphi'))$ et $f'_2 = \lambda\ t.(d't + max(0, -\varphi'))$.
Il résulte $(x, y) \in \mathcal{R}^F_{(n',\varphi',d')}$.

(b) On se situe dans le cas (b) de la Proposition 1, i.e. (n, φ, d) et (n', φ', d') satisfont les contraintes :

$$\begin{array}{l} (\ pgcd(n,d)\ \not|\ \varphi)\ \bigwedge\ (\ pgcd(n',d')\ \not|\ \varphi') \\ nd' = n'd \\ \left\lceil \frac{dt+\varphi}{n} \right\rceil \ = \ \left\lceil \frac{d't+\varphi'}{n'} \right\rceil\ , \forall t \in \mathbb{N}, dt + \varphi \geq 0 \end{array}$$

On note $\frac{n}{n'} = \frac{d}{d'} \overset{not}{=} k > 0$. Les expressions des fonctions f_1, f_2, f'_1, f'_2 conduisent dans ce cas à :

$$\begin{array}{rcl} f_1 & = & \lambda\ t.(nt + max(0, -\varphi)) \\ & = & \lambda\ t.(kn't + kmax(0, -\varphi') + max(0, -\varphi) - kmax(0, -\varphi')) \\ & = & kf'_1 + \lambda\ t.(max(0, -\varphi) - kmax(0, -\varphi')) \\ f_2 & = & \lambda\ t.(dt + max(0, \varphi)) \\ & = & \lambda\ t.(kd't + kmax(0, \varphi') + max(0, \varphi) - kmax(0, \varphi')) \\ & = & kf'_2 + \lambda\ t.(max(0, \varphi) - kmax(0, \varphi')) \end{array}$$

Comme $(\exists)$ $g : \mathbb{N} \rightarrow \mathbb{N}$, croissante sur $\mathbb{N}$ et strictement croissante sur $f_1(\mathbb{N}) \cup f_2(\mathbb{N})$, qui respecte les conditions (B.1), (B.2), on choisit $g' : \mathbb{N} \rightarrow \mathbb{N}$ croissante telle que

$$\begin{array}{ll} \forall s \in f'_1(\mathbb{N}), & g'(s) = g(ks + max(0, -\varphi) - kmax(0, -\varphi')) \\ \forall s \in f'_2(\mathbb{N}), & g'(s) = g(ks + max(0, \varphi) - kmax(0, \varphi')) \\ \forall s \in \mathbb{N} \setminus (f'_1(\mathbb{N}) \cup f'_2(\mathbb{N})), & g'(s) = w \in \mathbb{N} \text{ arbitraire} \end{array}$$

Le choix des valeurs w est fait tel que g' soit croissante sur $\mathbb{N}$ et $\forall t_1, t_2 \in \mathbb{N} \setminus (f'_1(\mathbb{N}) \cup f'_2(\mathbb{N})), t_1 = t_2 \Rightarrow g'(t_1) = g'(t_2)$. $pgcd(n', d')\ \not|\ \varphi' \Rightarrow f'_1(\mathbb{N}) \cap f'_2(\mathbb{N})$ est l'ensemble vide; $g'(s)$ est définie de manière unique sur $f'_1(\mathbb{N}) \cup f'_2(\mathbb{N})$.

1) g' est croissante sur $\mathbb{N}$ par construction.
2) La définition de g' implique $g' \circ f'_1 = g \circ f_1$, $g' \circ f'_2 = g \circ f_2$; il résulte $g' \circ f'_1$, $g' \circ f'_2$ strictement croissantes sur $\mathbb{N}$ et g' strictement croissante sur $f'_1(\mathbb{N})$ et $f'_2(\mathbb{N})$.
On va prouver g' strictement croissante sur $f'_1(\mathbb{N}) \cup f'_2(\mathbb{N})$.

Soit $t_1 \in f'_1(\mathbb{N}), t_2 \in f'_2(\mathbb{N}), t_1 < t_2$. Il existe $s_1, s_2 \in \mathbb{N}$ tels que $t_1 = f'_1(s_1), t_2 = f'_2(s_2)$ et $f'_1(s_1) < f'_2(s_2)$.

$$\begin{array}{rcl}
f'_1(s_1) < f'_2(s_2) & \Rightarrow & n's_1 < d's_2 + \varphi' \\
& \overset{s_1 \in \mathbb{N}}{\Rightarrow} & s_1 \leq \left[\frac{d's_2+\varphi'}{n'}\right] = \left[\frac{ds_2+\varphi}{n}\right] \leq \dfrac{ds_2 + \varphi}{n} \\
& \Rightarrow & ns_1 \leq ds_2 + \varphi \\
& \overset{pgcd(n,d) \not| \varphi}{\Rightarrow} & ns_1 < ds_2 + \varphi \\
& \Rightarrow & f_1(s_1) < f_2(s_2) \\
& \overset{\substack{g \text{ str. cr. sur} \\ f_1(\mathbb{N}) \cup f_2(\mathbb{N})}}{\Rightarrow} & (g \circ f_1)(s_1) < (g \circ f_2)(s_2) \\
& \Rightarrow & (g' \circ f'_1)(s_1) < (g' \circ f'_2)(s_2) \\
& \Rightarrow & g'(t_1) < g'(t_2)
\end{array}$$

Soit $t_1 \in f'_2(\mathbb{N}), t_2 \in f'_1(\mathbb{N}), t_1 < t_2$. Il existe $s_1, s_2 \in \mathbb{N}$ tels que $t_1 = f'_2(s_1), t_2 = f'_1(s_2)$ et $f'_2(s_1) < f'_1(s_2)$. Il résulte comme précédemment que $g'(t_1) < g'(t_2)$.

En conclusion, g' est strictement croissante sur $f'_1(\mathbb{N}) \cup f'_2(\mathbb{N})$.

On a exhibé $g' : \mathbb{N} \rightarrow \mathbb{N}$, croissante sur $\mathbb{N}$ et strictement croissante sur $f'_1(\mathbb{N}) \cup f'_2(\mathbb{N})$, qui satisfait les conditions de la définition d'une relation de (f'_1, f'_2)-codensification présentée dans la section 7.1 (comme $f'_1(t) \neq f'_2(s),\ \forall\, t, s \in \mathbb{N}$, on a évidemment $\forall\, t, s \in \mathbb{N},\ f'_1(t) = f'_2(s) \Rightarrow x_{(F)}(t) = y_{(F)}(s)$).
Il résulte $(x, y) \in \mathcal{R}^F_{(n',\varphi',d')}$.

On a démontré que, pour tout $(x, y) \in \mathcal{R}^F_{(n,\varphi,d)}$, on a aussi $(x, y) \in \mathcal{R}^F_{(n',\varphi',d')}$, donc $\mathcal{R}^F_{(n,\varphi,d)} \subseteq \mathcal{R}^F_{(n',\varphi',d')}$. Pour l'implication inverse, une succession de pas similaires permettra de construire la fonction g à partir de la fonction g'.
En conclusion, $\mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')}$.

$\square$

Annexe C

Preuve du Théorème 2, section 7.1.4

Preuve Nous démontrons que $(n_{FC}, \varphi_{FC}, d_{FC})$ est le plus petit triplet – par rapport à l'ordre partiel $\leq_t$ défini sur l'ensemble des triplets $(n, \varphi, d) \in \mathbb{N}^* \times \mathbb{Z} \times \mathbb{N}^*$ – équivalent à (n, φ, d).

a) a.1 (n, φ, d) et $(\frac{n}{G}, \frac{\varphi}{G}, \frac{d}{G})$ sont équivalents (ils satisfont les conditions (a) de la Proposition 1).

a.2 Soit le triplet (n', φ', d') équivalent à (n, φ, d) tel que $(n', \varphi', d') \neq (\frac{n}{G}, \frac{\varphi}{G}, \frac{d}{G})$. Il résulte (n', φ', d') équivalent à $(\frac{n}{G}, \frac{\varphi}{G}, \frac{d}{G})$ et ils satisfont les conditions (a) de la Proposition 1, c'est-à-dire $n'/d' = (\frac{n}{G})/(\frac{d}{G})$ et $d'(\frac{\varphi}{G}) = (\frac{d}{G})\varphi'$.

$\exists\, n_m, d_m, \varphi_m,\ n = Gn_m, d = Gd_m, \varphi = G\varphi_m$. Il est facilement vérifiable que n_m, d_m, φ_m sont premiers entre eux.

En conclusion $\frac{n'}{d'} = \frac{n_m}{d_m}$ et $d'\varphi_m = d_m\varphi'$.

Comme $(n', \varphi', d') \neq (\frac{n}{G}, \frac{\varphi}{G}, \frac{d}{G}) = (n_m, \varphi_m, d_m)$ et n_m, d_m, φ_m sont premiers entre eux, avec les résultats précédents $n' > \frac{n}{G}, \mid \varphi' \mid \ > \mid \frac{\varphi}{G} \mid$ et $d' > \frac{d}{G}$, i.e.[1] $(\frac{n}{G}, \frac{\varphi}{G}, \frac{d}{G}) <_t (n', \varphi', d')$.

En conclusion, $(\frac{n}{G}, \frac{\varphi}{G}, \frac{d}{G})$ est le plus petit élément de la classe d'équivalence de (n, φ, d), c'est-à-dire $(\frac{n}{G}, \frac{\varphi}{G}, \frac{d}{G})$ est la forme canonique de (n, φ, d).

b) On suppose $\varphi > 0$ (le cas $\varphi \leq 0$ peut être traité de manière similaire en utilisant $\left[\frac{nt-\varphi}{d}\right] = \left[\frac{n't-\varphi'}{d'}\right]$ $, \forall\ t \in \mathbb{N}$). Dans ce cas $dt + \varphi > 0, \forall\ t \in \mathbb{N}$, et les conditions (b) de la Proposition 1 sont équivalentes à :

$$\begin{array}{l} (\ pgcd(n,d) \not| \ \varphi) \\ \ pgcd(n',d') \not| \ \varphi') \\ nd' = dn' \\ \left[\frac{dt+\varphi}{n}\right] = \left[\frac{d't+\varphi'}{n'}\right], \forall\ t \in \mathbb{N} \end{array} \quad \Leftrightarrow \quad \begin{array}{l} (\ pgcd(n,d) \not| \ \varphi) \\ \ pgcd(n',d') \not| \ \varphi') \\ nd' = dn' \\ \left[\frac{dt+\varphi}{n}\right] = \left[\frac{d't+\varphi'}{n'}\right], \\ \forall\ t \in \{0,1,2,...,(n/G)-1\} \end{array}$$

A partir de la dernière relation du groupe en partie droite ci-dessus il résulte que l'on peut considérer seulement un ensemble fini de valeurs pour t.

On fixe quelques notations à utiliser dans la suite. On note $N^k_{n/G} = \{0, 1, 2, ..., k(n/G) - 1\}$ avec $k \in \mathbb{N}, k \geq 1$.

1. $(n_1, \varphi_1, d_1) <_t (n_2, \varphi_2, d_2)$ ssi $(n_1, \varphi_1, d_1) \leq_t (n_2, \varphi_2, d_2)$ et $\neg((n_1 = n_2) \wedge (\varphi_1 = \varphi_2) \wedge (d_1 = d_2))$.

$\forall\ t \in N^1_{n/G}$, on note $a_t = \frac{n}{G}\left[\frac{dt+\varphi}{n}\right] - \frac{d}{G}t$ et $A^k_t = \{ka_t + i, \forall\ i \in N^k_{n/G}\}$.

b.1 Dans la classe d'équivalence de (n, φ, d), on défini la relation minimale (n', φ', d') comme suit :

$$\exists\ m \in \mathbb{N}, m \geq 2,\ n' = m(n/G) \text{ et } d' = m(d/G); \tag{C.1}$$

$$m \text{ est minimal ;} \tag{C.2}$$

$$\varphi' \in \textstyle\bigcap_{t \in N^1_{n/G}} A^m_t \text{ et } m \not| \varphi'; \tag{C.3}$$

$$\varphi' \text{ est minimal.} \tag{C.4}$$

b.2 Il existe au moins une paire (m, φ') qui satisfait (C.1) et (C.3): (n, φ, d) appartient à sa classe d'équivalence, $G \not| \varphi$ et à partir de $\left[\frac{dt+\varphi}{n}\right] = \left[\frac{dt+\varphi}{n}\right]$, $\forall\ t \in N^1_{n/G}$ il résulte $\varphi \in \bigcap_{t \in N^1_{n/G}} A^G_t$. En conséquence $m = G$ et $\varphi' = \varphi$ satisfont les conditions (C.1) et (C.3).

b.3 Pour $m = 2$, il existe un unique φ' qui satisfait $\varphi' \in \bigcap_{t \in N^1_{n/G}} A^2_t$ et $2 \not| \varphi'$.

Preuve

– La condition b.2 implique $\varphi \in \bigcap_{t \in N^1_{n/G}} A^G_t$ et $G \not| \varphi$, c'est-à-dire

$$\textstyle\bigcap_{t \in N^1_{n/G}} \{x^t_i | x^t_i = Ga_t + i, \forall\ i \in N^G_{n/G} \text{ et } G \not| x^t_i\} \neq \emptyset$$

On obtient ainsi :

$$\exists\ i_0, ..., i_{n/G-1} \in N^G_{n/G},\quad G \not| i_k \quad \text{et } Ga_0 + i_0 = ... = Ga_{n/G-1} + i_{n/G-1}$$

équivalent à

$$\exists\ i_0, i_1, ..., i_{n/G-1} \in N^G_{n/G},\ \ G \not| i_k, \text{ satisfaisant le système Diophantine}$$

$$(S_1)\ \left\{\begin{array}{l} i_0 - i_1 = G(a_1 - a_0) \\ i_0 - i_2 = G(a_2 - a_0) \\ ... \\ i_0 - i_{n/G-1} = G(a_{n/G-1} - a_0) \end{array}\right.$$

Les solutions du système Diophantine (S_1) sont de la forme (voir [Ban93]):

$$(S'_1)\ \left\{\begin{array}{l} i_0 = s \\ i_1 = s - G(a_1 - a_0) \\ i_2 = s - G(a_2 - a_0) \\ ... \\ i_{n/G-1} = s - G(a_{n/G-1} - a_0) \end{array}\right.$$

avec$(s - G(a_k - a_0)) \in N^G_{n/G}$ et $G \not| (s - G(a_k - a_0)), \forall\ k \in N^1_{n/G}$.
Comme (S_1) admet au moins une solution, $\exists\ s = Gr + l$, avec $r \in N^1_{n/G}$ et $l \in \{1, 2, ..., G-1\}$, tel que s satisfait les système ci-dessus. De plus, $s - G(a_k - a_0) = G(r - a_k + a_0) + l$ et $s - G(a_k - a_0)$ doivent satisfaire $(s - G(a_k - a_0)) \in N^G_{n/G}$ et $G \not| (s - G(a_k - a_0)), \forall\ k \in N^1_{n/G}$.

Ces résultats impliquent que le système Diophantine (S_2) admet au moins une solution.

$$(S_2) \quad \begin{cases} r - p_k = a_k - a_0 \\ r, p_k \in N^1_{n/G}, \ \forall \ k \in \{1, 2, ..., n/G - 1\} \end{cases}$$

– Il existe un vecteur *unique* $(r, p_1, p_2, ..., p_{n/G-1})$ qui satisfait (S_2) (preuve par induction).
  De plus, les solutions de (S'_1) sont de la forme $s_l = Gr + l, l \in \{1, 2, ..., G-1\}$. Comme $\varphi \in \bigcap_{t \in N^1_{n/G}} A^G_t$, $\exists \ l \in \{1, 2, ..., G-1\}, \varphi = Ga_0 + s_l$. On obtient $\varphi = G(a_0 + r) + l$, i.e. $r = \left[\frac{\varphi}{G}\right] - a_0$.
– Comme antérieurement pour $\bigcap_{t \in N^1_{n/G}} A^G_t$, on a :

$$\begin{array}{lcl} \exists \ \varphi' \in \bigcap_{t \in N^1_{n/G}} A^2_t \neq \emptyset, & \Leftrightarrow & \text{Le système suivant admet} \\ 2 \nmid \varphi' & & \\ & & \text{au moins une solution} \end{array}$$

$$(S_3) \quad \begin{cases} i_0 = s' \\ i_1 = s' - 2(a_1 - a_0) \\ i_2 = s' - 2(a_2 - a_0) \\ \ldots \\ i_{n/G-1} = s' - 2(a_{n/G-1} - a_0) \\ \\ \text{avec} \\ s' - 2(a_k - a_0)) \in N^2_{n/G} \text{ et} \\ 2 \nmid (s' - 2(a_k - a_0)), \ \forall \ k \in N^1_{n/G} \end{cases}$$

Une solution potentielle de (S_3) sera de la forme $s' = 2r' + 1$, avec r' le premier élément du vecteur solution de (S_2) . Il résulte $r' = r$ et (S_3) admet une solution unique $s' = 2r + 1$.
– En conclusion, il existe un unique $\varphi' \in \bigcap_{t \in N^1_{n/G}} A^2_t, \varphi' = 2a_0 + s' = 2(a_0 + r) + 1 = 2\left[\frac{\varphi}{G}\right] + 1$.

(preuve b.3) □

b.4 $(n' = 2\frac{n}{G}, \varphi' = 2\left[\frac{\varphi}{G}\right] + 1, d' = 2\frac{n}{G})$ est la relation minimale équivalente à (n, φ, d).

Preuve $\forall (n'', \varphi'', d'')$ équivalent à (n, φ, d), $\exists \ p \in \mathbb{N}, p \geq 2, \quad n'' = p(n/G), d'' = p(d/G)$. Donc, $n'' \geq n', d'' \geq n'$. De plus, la preuve de la condition b.3 implique $\varphi'' = p(a_0 + r) + l_p$, avec $l_p \in \{1, 2, ..., p-1\}$. En conclusion, $\varphi'' \geq \varphi'$. (preuve b.4) □

En conclusion, $(2\frac{n}{G}, 2\left[\frac{\varphi}{G}\right] + 1, 2\frac{n}{G})$ est la forme canonique de (n, φ, d).

(preuve Théorème 2, section 7.1.4) □

Annexe D

Propriétés sur entiers

Nous présentons dans cette section les propriétés sur entiers qui sont utilisées dans le calcul sur relations affines présenté dans le chapitre 7.

Nous supposons les notations suivantes :

$\mathbb{N} = \{0, 1, 2, \ldots\}$ (l'ensemble des entiers positifs, 0 inclus).
$\mathbb{N}^* = \{1, 2, \ldots\}$ (l'ensemble des entiers strictement positifs).
$\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ (l'ensemble de tous les entiers).
$kA = \{kt \mid t \in A\}, \ \forall\, A \in \{\mathbb{N}, \mathbb{N}^*, \mathbb{Z}\}$(par exemple, $k\mathbb{N} = \{kt \mid t \in \mathbb{N}\}$).
$A + i = \{t + i \mid t \in A\}, \ \forall\, A \in \{\mathbb{N}, \mathbb{N}^*, \mathbb{Z}\}$ et $i \in \mathbb{Z}$.

Lemme 18

$$n\mathbb{N} + \varphi = \cup_{i=0}^{d-1}((d \cdot n)\mathbb{N} + \varphi + i \cdot n), \ \ \forall\, n, d \in \mathbb{N}^*, \varphi \in \mathbb{N}$$

Preuve $\forall\, t \in n\mathbb{N} + \varphi, \ \exists\, t' \in \mathbb{N}$ tel que $t = nt' + \varphi$.

$\forall\, t' \in \mathbb{N}, \ \exists\, t'' \in \mathbb{N}$ unique et $\exists\, i \in \{0, 1, 2, \ldots, d-1\}$ tels que $t' = dt'' + i$ (ce résultat, connu sous le nom de « propriété de division » des entiers naturels [Lip81] [Sam71], est vrai plus généralement pour $t', t'' \in \mathbb{Z}$). Il résulte : $\forall\, t \in n\mathbb{N} + \varphi, \ \exists\, t'' \in \mathbb{N}$ et $\exists\, i \in \{0, 1, 2, \ldots, d-1\}$ tels que $t = n(dt'' + i) + \varphi$, i.e. $t = (d \cdot n)t'' + \varphi + i \cdot n$. A partir de ce résultat, on a $n\mathbb{N} + \varphi \subseteq \cup_{i=0}^{d-1}((d \cdot n)\mathbb{N} + \varphi + i \cdot n)$.

Réciproquement, $\forall\, t \in \cup_{i=0}^{d-1}((d\cdot n)\mathbb{N} + \varphi + i\cdot n), \ \exists\, t'' \in \mathbb{N}$ et $\exists i \in \{0, 1, 2, \ldots, d-1\}$ tels que $t = (d\cdot n)t'' + \varphi + i \cdot n$. On a encore $t = n(dt'' + i) + \varphi$, et donc $t \in n\mathbb{N} + \varphi$. Il résulte $\cup_{i=0}^{d-1}((d \cdot n)\mathbb{N} + \varphi + i \cdot n) \subseteq n\mathbb{N} + \varphi$. □

Lemme 19 *Soit $I_1, I_2 \subseteq \mathbb{N}$ des sous-ensembles discrets de $\mathbb{N}$.*

$$\cup_{\varphi_1 \in I_1}(n\mathbb{N} + \varphi_1) = \cup_{\varphi_2 \in I_2}(n\mathbb{N} + \varphi_1) \ \ \Leftrightarrow \ \ I_1 = I_2, \ \ \forall\, n \in \mathbb{N}^*$$

Preuve "$\Leftarrow$" $I_1 = I_2$ implique évidemment $\cup_{\varphi_1 \in I_1}(n\mathbb{N} + \varphi_1) = \cup_{\varphi_2 \in I_2}(n\mathbb{N} + \varphi_1)$.

"$\Rightarrow$" Supposons $\cup_{\varphi_1 \in I_1}(n\mathbb{N} + \varphi_1) = \cup_{\varphi_2 \in I_2}(n\mathbb{N} + \varphi_1)$ et démontrons que $I_1 = I_2$.

Soit $\varphi_1^{min}, \varphi_2^{min}$ les plus petits éléments de I_1 et I_2. On a évidemment $\varphi_1^{min} = \varphi_2^{min}$ et aussi :

$$\begin{aligned} n\mathbb{N} + \varphi_1^{min} = n\mathbb{N} + \varphi_2^{min} \ &\Rightarrow \ (\ \cup_{\varphi_1 \in I_1}(n\mathbb{N} + \varphi_1)\ ) \setminus (\ n\mathbb{N} + \varphi_1^{min}\ ) = \\ &\ \ (\ \cup_{\varphi_2 \in I_2}(n\mathbb{N} + \varphi_2)\ ) \setminus (\ n\mathbb{N} + \varphi_2^{min}\ ) \\ &\Rightarrow \ \cup_{\varphi_1 \in I_1 \setminus \{\varphi_1^{min}\}}(n\mathbb{N} + \varphi_1) = \cup_{\varphi_2 \in I_2 \setminus \{\varphi_2^{min}\}}(n\mathbb{N} + \varphi_2) \end{aligned}$$

On note $I_1' = I_1 \setminus \{\varphi_1^{min}\}$ et $I_2' = I_2 \setminus \{\varphi_2^{min}\}$.

En considérant maintenant $\varphi_1^{min}, \varphi_2^{min}$ les plus petits éléments des I'_1 et respectivement I'_2 et en appliquant récursivement le même raisonnement, on obtient $\mid I_1 \mid \ = \mid I_2 \mid$ et $I_1 = I_2$.

□

Proposition 17 *(Intersection d'ensembles d'entiers)*
$\forall\ d_1, d_2 \in I\!N^*, \varphi_1, \varphi_2 \in I\!N,$

$$(d_1 I\!N + \varphi_1) \cap (d_2 I\!N + \varphi_2) = d I\!N + \varphi \quad \Leftrightarrow \quad \begin{cases} pgcd(d_1, d_2) \mid \varphi_2 - \varphi_1 \\ d = \frac{d_1 d_2}{g} = ppcm(d_1, d_2) \\ \varphi = d_1 s_{10} + \varphi_1 \end{cases}$$

avec $pgcd(d_1, d_2){=}g$ *le plus grand diviseur commun de* d_1 *et* d_2 *et* $s_{10} = s'_{10} - \frac{d_2}{g}\ [(s'_{10} g)/d_2]$ *avec* (s'_{10}, s'_{20}) *une solution particulière de* $d_1 s'_{10} - d_2 s'_{20} = \varphi_2 - \varphi_1$.

Preuve

$$(d_1 \mathbb{N} + \varphi_1) \cap (d_2 \mathbb{N} + \varphi_2) \neq \emptyset \quad \Leftrightarrow \quad \exists\ s_1, s_2 \in \mathbb{N}, d_1 s_1 + \varphi_1 = d_2 s_2 + \varphi_2$$

Afin que l'intersection ci-dessus soit non vide, les paramètres $d_1, d_2, \varphi_1, \varphi_2$ doivent satisfaire les conditions nécessaires et suffisantes pour que l'équation Diophantine $d_1 s_1 - d_2 s_2 = \varphi_2 - \varphi_1$ admette des solutions entières. Ceci est équivalent à $g \mid (\varphi_2 - \varphi_1)$.

Dans ce cas, les solutions s_1, s_2 de l'équation Diophantine $d_1 s_1 - d_2 s_2 = \varphi_2 - \varphi_1$ seront de la forme :

$$\begin{cases} s_1 = s_{10} - \frac{d_2}{g} t \\ s_2 = s_{20} - \frac{d_1}{g} t \end{cases}$$

avec $t \in \mathbb{Z}$ et (s_{10}, s_{20}) une solution particulière de l'équation $d_1 s_{10} - d_2 s_{20} = \varphi_2 - \varphi_1$. La solution (s_{10}, s_{20}) peut s'écrire :

$$\begin{cases} s_{10} = s'_{10} - \frac{d_2}{g} t' \\ s_{20} = s'_{20} - \frac{d_1}{g} t' \end{cases}$$

avec $t' \in \mathbb{Z}$ et (s'_{10}, s'_{20}) une autre solution de l'équation $d_1 s'_{10} - d_2 s'_{20} = \varphi_2 - \varphi_1$.

Pour s'_{10}, s'_{20} fixés, on choisit $t' = [(s'_{10} g)/d_2]$ ($[x]$ représente la partie entière de x). Les propriétés de la partie entière $[x] \leq x < [x] + 1, \forall\ x$ induisent :

$$(s'_{10} g)/d_2 - 1 < t' \leq (s'_{10} g)/d_2$$

On obtient la suite d'équivalences :

$$(s'_{10} g)/d_2 - 1 < t' \leq (s'_{10} g)/d_2 \quad \Leftrightarrow \quad 0 \leq (s'_{10} g)/d_2 - t' < 1 \quad \Leftrightarrow \quad 0 \leq (s_{10} g)/d_2 < 1$$

Il est nécessaire que s_1 soit élément de $\mathbb{N}$, i.e. $s_1 \geq 0$. Ceci est vrai pour $t \in \mathbb{Z},\ t \leq (s_{10} g)/d_2$. Comme $(s_{10} g)/d_2 < 1$, il résulte $t \in \mathbb{Z},\ t \leq 0$.

En conséquence, la solution entière s_1 de l'équation Diophantine $d_1 s_1 - d_2 s_2 = \varphi_2 - \varphi_1$ est de la forme :

$$s_1 = s_{10} + \frac{d_2}{g} t,\ \forall\ t \in \mathbb{N}$$
$$s_{10} = s'_{10} - \frac{d_2}{g}\ [(s'_{10} g)/d_2]$$

avec (s'_{10}, s'_{20}) une solution particulière de $d_1 s'_{10} - d_2 s'_{20} = \varphi_2 - \varphi_1$.

On a démontré que $(d_1\mathbb{N}+\varphi_1)\cap(d_2\mathbb{N}+\varphi_2)\neq\emptyset$ ssi $g\mid\varphi_2-\varphi_1$ et dans ce cas $(d_1\mathbb{N}+\varphi_1)\cap(d_2\mathbb{N}+\varphi_2)=\{t_A\mid t_A=\frac{d_1d_2}{g}t+d_1s_{10}+\varphi_1,\ \forall\, t\in\mathbb{N}\}$. Si on note $d=\frac{d_1d_2}{g}$ et $\varphi=d_1s_{10}+\varphi_1$, alors

$$(d_1\mathbb{N}+\varphi_1)\cap(d_2\mathbb{N}+\varphi_2)=d\mathbb{N}+\varphi \quad\Leftrightarrow\quad \left\{\begin{array}{l} pgcd(d_1,d_2)\mid\varphi_2-\varphi_1\\ d=\frac{d_1d_2}{g}=ppcm(d_1,d_2)\\ \varphi=d_1s_{10}+\varphi_1\end{array}\right.$$

□

Proposition 18 *(Union d'ensembles d'entiers)*
$\forall\ d_1,d_2\in\mathbb{N}^*,\varphi_1,\varphi_2\in\mathbb{N}$,

$$\left\{\begin{array}{l}(d_1\mathbb{N}+\varphi_1)\cup(d_2\mathbb{N}+\varphi_2)=d\mathbb{N}+\varphi\\ (d_1\mathbb{N}+\varphi_1)\cap(d\mathbb{N}+\varphi)=d_1\mathbb{N}+\varphi_1\\ (d_2\mathbb{N}+\varphi_2)\cap(d\mathbb{N}+\varphi)=d_2\mathbb{N}+\varphi_2\end{array}\right. \quad\Leftrightarrow$$

On se trouve dans une des situations suivantes :

$$\left\{\begin{array}{ll} d_2=k_2d_1,\ k_2\in\mathbb{N}^* & d=d_1\\ \varphi_2=\varphi_1+k_1'd_1,\ k_1'\in\mathbb{N} & \varphi=\varphi_1\end{array}\right.$$

$$\left\{\begin{array}{ll} d_1=d_2=2k,\ k\in\mathbb{N}^* & d=d_1\\ \varphi_2=\varphi_1+k & \varphi=\varphi_1\end{array}\right.$$

$$\left\{\begin{array}{l} d_1=k_1d_2,\ k_1\in\mathbb{N}^*\\ \varphi_2=\varphi_1+\left\{\begin{array}{ll} d_2, & si\ k_1>1\\ k_2'd_2, & si\ k_1=1\\ (k_2'\in\mathbb{N}^*) & \end{array}\right.\\ d=d_2\\ \varphi=\varphi_1\end{array}\right.$$

et les cas réciproques obtenus en permutant d_1 et d_2, φ_1 et φ_2 dans les équations antérieures.

Preuve Premièrement, on applique la proposition antérieure pour les intersections ci-dessus :

$$1)\ \ (d_1\mathbb{N}+\varphi_1)\cap(d\mathbb{N}+\varphi)=d_1\mathbb{N}+\varphi_1 \quad\Leftrightarrow\quad \left\{\begin{array}{l} pgcd(d_1,d)\mid\varphi-\varphi_1\\ d_1=\frac{d_1d}{g_1}=ppcm(d_1,d)\\ \varphi_1=d_1s_{10}+\varphi_1\end{array}\right.$$

avec $pgcd(d_1,d)=g_1$ le plus grand diviseur commun de d_1 et d et $s_{10}=s_{10}'-\frac{d}{g_1}\ [(s_{10}'g_1)/d]$, (s_{10}',s_0') une solution particulière de $d_1s_{10}'-ds_0'=\varphi-\varphi_1$.

$$1.1)\ \ d_1=\frac{d_1d}{g_1}\ \ \Rightarrow\ \ d=g_1\ \ \Rightarrow\ \ \exists\ k_1\in\mathbb{N}^*,\ d_1=k_1d$$

$$2)\ \ (d_2\mathbb{N}+\varphi_2)\cap(d\mathbb{N}+\varphi)=d_2\mathbb{N}+\varphi_2 \quad\Leftrightarrow\quad \left\{\begin{array}{l} pgcd(d_2,d)\mid\varphi-\varphi_2\\ d_2=\frac{d_2d}{g_2}=ppcm(d_2,d)\\ \varphi_2=d_2s_{20}+\varphi_2\end{array}\right.$$

avec $pgcd(d_2,d){=}g_2$ le plus grand diviseur commun de d_2 et d et $s_{20} = s'_{20} - \frac{d}{g_2}\ [(s'_{20}g_2)/d]$, où (s'_{20}, s''_0) une solution particulière de $d_2 s'_{20} - ds'_0 = \varphi - \varphi_2$.

$$2.2)\ d_2 = \frac{d_2 d}{g_2} \quad \Rightarrow \quad d = g_2 \quad \Rightarrow \quad \exists\ k_2 \in \mathbb{N}^*,\ d_2 = k_2 d$$

En remplaçant les nouvelles valeurs de d_1 et d_2 dans $(d_1\mathbb{N} + \varphi_1) \cup (d_2\mathbb{N} + \varphi_2) = d\mathbb{N} + \varphi$, on obtient :

$$(k_1 d\mathbb{N} + \varphi_1) \cup (k_2 d\mathbb{N} + \varphi_2) = d\mathbb{N} + \varphi \tag{D.1}$$

– Cas 1 : $\varphi_2 \geq \varphi_1$.

Dans ce cas, D.1 peut être transformée en

$$(k_1 d\mathbb{N}) \cup (k_2 d\mathbb{N} + \varphi_2 - \varphi_1) = d\mathbb{N} + \varphi - \varphi_1$$

Dans cette situation, on peut avoir seulement les deux cas suivants :

– Cas 1.1 : $\left\{\begin{array}{l} k_1 d = d(= d_1) \\ \varphi - \varphi_1 = 0 \\ \varphi_2 - \varphi_1 = k'_1 d \end{array}\right. \Rightarrow \left\{\begin{array}{l} d_2 = k_2 d_1,\ k_2 \in \mathbb{N}^* \\ \varphi_2 = \varphi_1 + k'_1 d_1,\ k'_1 \in \mathbb{N} \\ d = d_1 \\ \varphi = \varphi_1 \end{array}\right.$

Si $k_2 = 1$ et $d_1 = 2k,\ k \in \mathbb{N}^*$, on peut aussi avoir $\varphi_2 - \varphi_1 = \frac{d_1}{2}$ et on retrouve les conditions suivantes :

$$\left\{\begin{array}{l} d_1 = d_2 = 2k,\ k \in \mathbb{N}^* \\ \varphi - \varphi_1 = 0 \\ \varphi_2 - \varphi_1 = k \end{array}\right. \Rightarrow \left\{\begin{array}{l} d_1 = d_2 = 2k,\ k \in \mathbb{N}^* \\ \varphi_2 = \varphi_1 + k \\ d = d_1 \\ \varphi = \varphi_1 \end{array}\right.$$

– Cas 1.2 : $\left\{\begin{array}{l} k_2 d = d(= d_2) \\ \varphi - \varphi_1 = 0 \\ \varphi_2 - \varphi_1 = \left\{\begin{array}{ll} d, & \text{si } k_1 > 1 \\ k'_2 d, & \text{si } k_1 = 1 \\ (k'_2 \in \mathbb{N}^*) & \end{array}\right. \end{array}\right. \Rightarrow \left\{\begin{array}{l} d_1 = k_1 d_2,\ k_1 \in \mathbb{N}^* \\ \varphi_2 = \varphi_1 + \left\{\begin{array}{ll} d_2, & \text{si } k_1 > 1 \\ k'_2 d_2, & \text{si } k_1 = 1 \\ (k'_2 \in \mathbb{N}^*) & \end{array}\right. \\ d = d_2 \\ \varphi = \varphi_1 \end{array}\right.$

– Cas 2 : $\varphi_2 < \varphi_1$.

Dans ce cas, D.1 peut être transformée en

$$(k_1 d\mathbb{N} + \varphi_1 - \varphi_2) \cup (k_2 d\mathbb{N}) = d\mathbb{N} + \varphi - \varphi_2$$

Avec une analyse analogue, on obtient les réciproques des cas 1.1 et 1.2 précédents :

– Cas 2.1 : $\left\{\begin{array}{l} d_1 = k_1 d_2,\ k_1 \in \mathbb{N}^* \\ \varphi_1 = \varphi_2 + k'_2 d_2,\ k'_2 \in \mathbb{N} \\ d = d_2 \\ \varphi = \varphi_2 \end{array}\right.$ et $\left\{\begin{array}{l} d_1 = d_2 = 2k,\ k \in \mathbb{N}^* \\ \varphi_1 = \varphi_2 + k \\ d = d_2 \\ \varphi = \varphi_2 \end{array}\right.$

$$
\text{– Cas 2.2:} \left\{ \begin{array}{l} d_2 = k_2 d_1,\ k_2 \in \mathbb{N}^* \\ \varphi_1 = \varphi_2 + \left\{ \begin{array}{ll} d_1, & \text{si } k_2 > 1 \\ k'_1 d_1, & \text{si } k_2 = 1 \\ (k'_1 \in \mathbb{N}^*) & \end{array} \right. \\ d = d_1 \\ \varphi = \varphi_2 \end{array} \right.
$$

□

Proposition 19 *(Différence d'ensembles d'entiers)*
$\forall\ d_1, d_2 \in \mathbb{N}^*, \varphi_1, \varphi_2 \in \mathbb{N}$,

$$
(d_1\mathbb{N} + \varphi_1) \setminus (d_2\mathbb{N} + \varphi_2) = d\mathbb{N} + \varphi \quad \Leftrightarrow \quad \left\{ \begin{array}{l} d_2 = d = 2d_1 \\ \varphi_2 = \varphi_1 \\ \varphi = \varphi_1 + d_1 \end{array} \right. \quad ou \quad \left\{ \begin{array}{l} d_2 = d = 2d_1 \\ \varphi_2 = \varphi_1 + d_1 \\ \varphi = \varphi_1 \end{array} \right.
$$

Preuve

$$
\begin{array}{rcl} (d_1\mathbb{N} + \varphi_1) \setminus (d_2\mathbb{N} + \varphi_2) & = & (d_1\mathbb{N} + \varphi_1) \setminus ((d_1\mathbb{N} + \varphi_1) \cap (d_2\mathbb{N} + \varphi_2)) \\ & = & \left\{ \begin{array}{ll} d_1\mathbb{N} + \varphi_1, & \text{si } (d_1\mathbb{N} + \varphi_1) \cap (d_2\mathbb{N} + \varphi_2) = \emptyset \\ (d_1\mathbb{N} + \varphi_1) \setminus (\frac{d_1 d_2}{g}\mathbb{N} + d_1 s_{10} + \varphi_1), & \text{sinon} \end{array} \right. \end{array}
$$

avec $pgcd(d_1, d_2) = g$ le plus grand diviseur commun de d_1 et d_2 et $s_{10} = s'_{10} - \frac{d_2}{g}\ [(s'_{10}g)/d_2]$ avec (s'_{10}, s'_{20}) une solution particulière de $d_1 s'_{10} - d_2 s'_{20} = \varphi_2 - \varphi_1$.

L'intersection $(d_1\mathbb{N} + \varphi_1) \cap (d_2\mathbb{N} + \varphi_2)$ est vide ssi $pgcd(d_1, d_2) \not| \varphi_2 - \varphi_1$. Dans ce cas, $(d_1\mathbb{N} + \varphi_1) \setminus (d_2\mathbb{N} + \varphi_2) = d_1\mathbb{N} + \varphi_1$, donc $d = d_1$ et $\varphi = \varphi_1$.

Supposons maintenant $pgcd(d_1, d_2) \mid \varphi_2 - \varphi_1$.

$$
\begin{array}{rcl} (d_1\mathbb{N} + \varphi_1) \setminus (d_2\mathbb{N} + \varphi_2) & = & (d_1\mathbb{N} + \varphi_1) \setminus (\frac{d_1 d_2}{g}\mathbb{N} + d_1 s_{10} + \varphi_1) \\ & \overset{Lemme18}{=} & \cup_{i=0}^{(d_2/g)-1} ((\frac{d_2}{g} \cdot d_1)\mathbb{N} + \varphi_1 + i \cdot d_1) \setminus (\frac{d_1 d_2}{g}\mathbb{N} + d_1 s_{10} + \varphi_1) \end{array}
$$

Or, pour s_{10} on obtient :

$$
\left. \begin{array}{l} s_{10} = s'_{10} - \frac{d_2}{g}\ [(s'_{10}g)/d_2] \\ [(s'_{10}g)/d_2] \ \leq (s'_{10}g)/d_2 < \ [(s'_{10}g)/d_2]\ + 1 \end{array} \right\} \Rightarrow \ 0 \leq s_{10} < \frac{d_2}{g},\ s_{10} \in \mathbb{N}
$$

$$
\Rightarrow \ s_{10} \in \{0, 1, 2, \ldots, \frac{d_2}{g} - 1\}
$$

Dans ces conditions, une condition nécessaire pour que $(d_1\mathbb{N} + \varphi_1) \setminus (d_2\mathbb{N} + \varphi_2)$ puisse s'écrire sous la forme $d\mathbb{N} + \varphi$ est $\frac{d_2}{g} = 2$; il en résultera $s_{10} \in \{0, 1\}$.

Dans le cas $\frac{d_2}{g} = 2$, l'ensemble $d_1\mathbb{N} + \varphi_1$ se réécrit sous la forme d'une union comme suit : $d_1\mathbb{N} + \varphi_1 = (\ 2d_1\mathbb{N} + \varphi_1\ ) \cup (\ 2d_1\mathbb{N} + \varphi_1 + d_1\ )$.

$(d_1\mathbb{N} + \varphi_1) \setminus (d_2\mathbb{N} + \varphi_2)$ peut s'écrire sous la forme $d\mathbb{N} + \varphi$ si et seulement si

$$
d_2\mathbb{N} + \varphi_2 = 2d_1\mathbb{N} + \varphi_1 \ \Rightarrow \ \left\{ \begin{array}{l} d_2 = 2d_1 \\ \varphi_2 = \varphi_1 \end{array} \right. \ \Rightarrow \ \left\{ \begin{array}{l} d = 2d_1 \\ \varphi = \varphi_1 + d_1 \end{array} \right.
$$

ou

$$
d_2\mathbb{N} + \varphi_2 = 2d_1\mathbb{N} + \varphi_1 + d_1 \ \Rightarrow \ \left\{ \begin{array}{l} d_2 = 2d_1 \\ \varphi_2 = \varphi_1 + d_1 \end{array} \right. \ \Rightarrow \ \left\{ \begin{array}{l} d = 2d_1 \\ \varphi = \varphi_1 \end{array} \right.
$$

□

Résumé

Cette thèse s'inscrit dans le cadre du développement du langage SIGNAL dont les objectifs sont la spécification et la mise en œuvre d'applications temps-réel critiques. Nos travaux ont concerné l'extension du compilateur de SIGNAL qui est un outil formel pour la vérification des contraintes de synchronisation d'horloges en SIGNAL. De manière plus générale, notre étude se situe dans un contexte de conception conjointe de systèmes mixtes matériel/logiciel (codesign) pour laquelle l'environnement de programmation de SIGNAL s'est prouvé très adapté.

Dans ce contexte, nous avons constaté que les langages SIGNAL et ALPHA possèdent des propriétés complémentaires concernant la programmation de traitements numériques importants reliés par du contrôle. ALPHA est un langage fonctionnel pour la spécification et l'implémentation d'algorithmes réguliers, tandis que l'apport de SIGNAL se manifeste dans le domaine du contrôle. L'interfaçage des deux langages a relevé le problème du raffinement des spécifications mixtes et leur validation. Les transformations de spécifications ont induit des transformations affines sur les horloges SIGNAL et la nécessité de mettre en place un calcul formel sur les synchronisations des horloges transformées.

Le nouveau calcul formel mis en place est basé sur un ensemble de règles de synchronisabilité déduites à partir des propriétés des transformations affines. Nous avons retrouvé des conditions de synchronisabilité basées sur un calcul d'entiers qui nous ont permis d'étendre les capacités de preuve formelle du compilateur de SIGNAL. Nous avons validé le calcul de synchronisabilité proposé et les concepts liés à la conception conjointe avec SIGNAL et ALPHA à travers une application du domaine de codage d'images vidéo.

Mots clef : temps-réel, programmation synchrone, conception conjointe matériel/logiciel, transformation affine, calcul d'horloges, synchronisabilité.

Abstract

The work described in this document is part of the actions concerning the development of the language SIGNAL for the specification and implementation of critical real-time applications. We have been particularly concerned about the extension of the SIGNAL compiler which is a formal tool used for the verification of synchronization constraints on clocks in SIGNAL. More generally, our study is placed in the framework of the codesign of mixed hardware/software systems for which the SIGNAL environment has proved appropriate.

Our starting point has been the observation that the languages SIGNAL and ALPHA have complementary properties concerning the programming of important numerical treatments related by control parts. ALPHA is a functional language used for the specification and implementation of regular algorithms, while SIGNAL is appropriate for the programming of the control parts. Interfacing the two languages has introduced the problem of the refinement and validation of mixed specifications. The transformations performed on specifications have further introduced affine transformations on clocks in SIGNAL and the necessity of developping a formal calculus on the synchronization of the transformed clocks.

We have defined and implemented a new formal calculus based on a set of synchronizability rules induced by the properties of affine transformations. In this context, we have retrieved simple and elegant synchronizability rules based on integer calculus and we have thus augmented the proof capabilities of the SIGNAL compiler. We have validated the proposed synchronizability calculus and the concepts related to codesign using SIGNAL and ALPHA on a video image coding application.

Keywords: real-time, synchronous programming, codesign, affine transformation, clock calculus, synchronizability.