Expression et validation de contraintes temporelles pour la spécification des systèmes réactifs

David Delfieu

Année 1995

THÈSE

préparée au
LABORATOIRE D'ANALYSE ET D'ARCHITECTURE DES SYSTÈMES du CNRS

en vue de l'obtention du
DOCTORAT DE L'UNIVERSITÉ PAUL SABATIER
Spécialité : Informatique

par

David DELFIEU
Maître ès Informatique

Expression et validation de contraintes temporelles pour la spécification des systèmes réactifs

---

Soutenue le 6 janvier 1995 devant le jury :

MM.	**Bernard LECUSSAN**	Président
	Jean-Claude GENTINA	Rapporteur
	Christophe SIBERTIN-BLANC	Rapporteur
	Jean-Pierre THOMESSE	Rapporteur
	Daniel DOURS	Examinateur
	Alain JEFFROY	Examinateur
	Abd-El-Kader SAHRAOUI	Examinateur
	Robert VALETTE	Examinateur

Cette thèse a été préparée au
Laboratoire d'Analyse et d'Architecture des Systèmes du C.N.R.S.
7, avenue du Colonel Roche 31077 TOULOUSE CEDEX

Rapport LAAS *N°* : 95076

AVANT-PROPOS

Les travaux présentés dans ce mémoire ont été effectués au sein du groupe "Système de Production" (SP) du Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS) du Centre National de la Recherche Scientifique (CNRS). Je tiens à remercier Monsieur A. COSTES, directeur du LAAS pour m'avoir accueilli dans ce laboratoire, et messieurs Robert VALETTE et Jean-Claude HENNET directeurs successifs du groupe SP.

Je tiens à exprimer ma profonde gratitude à Monsieur Abd-El-Kader SAHRAOUI, Professeur à l'Institut Universitaire de Technologie de Toulouse-le Mirail, qui a accepté de diriger mes recherches. Malgré les contraintes temporelles dures auxquelles il était soumis, il a su assurer un encadrement propice au bon déroulement de ma thèse. Je lui suis fort reconnaissant pour ses conseils, ses remarques pertinentes, mais aussi pour ses "coups de colère" qui m'ont utilement forcé à me remettre en question.

Je suis extrêmement reconnaissant à Messieurs Jean-Claude GENTINA, Directeur de l'École Centrale de Lille, Christophe SIBERTIN-BLANC, Professeur à l'Université des Sciences Sociales de Toulouse, ainsi que Monsieur Jean-Pierre THOMESSE, Professeur des Universités (ENSEM) pour avoir accepté d'étudier mes travaux et d'en être les rapporteurs.

Je suis également reconnaissant à Messieurs Daniel DOURS, Professeur des Universités, Alain JEFFROY ingénieur à l'Aérospatiale, Bernard LECUSSAN, Professeur à l'École Nationale Supérieure de l'Aéronautique et de l'Espace qui m'ont honoré de leur présence dans le jury de thèse, et plus particulièrement à Monsieur Robert VALETTE, Directeur de recherche au CNRS, pour ses encouragements et son application à répondre à l'ensemble des questions que j'ai pu lui poser.

Je remercie très chaleureusement l'ensemble des membres du groupe SP pour leur gentillesse et leur disponibilité, et plus particulièrement Messieurs Eric Bernauer et Agnan de Bonneval, pour les échanges de point de vue que l'on a pu avoir, et pour la motivation mutuelle qui s'en est dégagée.

Mes remerciements s'adressent de plus, à l'ensemble du personnel du LAAS : les services Informatiques, Édition-Documentation, Direction-Gestion, et Réception qui participent à l'excellence de ce laboratoire.

Enfin, je tiens à adresser une pensée affectueuse et reconnaissante à ma famille et à celle qui partage ma vie, qui m'a supporté, dans tous les sens du terme, pendant ces années de thèse.

Table des Matières

INTRODUCTION

Le temps est une donnée précieuse, et peut donc être considéré à une ressource critique. C'est un paramètre qui est de plus en plus pris en compte au niveau des systèmes. La montée en puissance et la diminution du coût de l'électronique d'une part, les besoins croissants d'informations d'autre part, font que la gestion du temps devient un enjeu majeur dans tous les domaines.

Que cela soit, dans les systèmes de communications où les routages sont gérés de manière dynamique afin d'augmenter un flux général ou encore dans une cellule flexible d'assemblage manufacturier dont les ratios de production sont redéfinis dynamiquement en fonction d'aléas de fonctionnement, ces activités sont qualifiées de "temps réel". C'est à dire que la décision face à une situation réelle ne se fait plus par rapport à un plan de fonctionnement qui aurait été défini de manière statique, mais par rapport à une évaluation rapide de la situation présente et à une réponse immédiate.

L'aspect "temps réel" est un des thèmes centraux de notre groupe Système de Production du LAAS-CNRS à Toulouse, comme l'indique les dernières thèses soutenues. [de Bonneval 1993] traite de la surveillance en **temps réel** des défaillances du procédé, ou encore [Billaut 1993] dans le domaine de l'ordonnancement **temps réel**. Dans les travaux de A.E.K. Sahraoui ([Sahraoui 1994], [Sahraoui 1988], [Sahraoui 1989]) le problème de l'expression et de la validation des contraintes temporelles est une nouvelle orientation ([Sahraoui 1992]) qui fait l'objet de ce mémoire.

Le processus de réalisation des systèmes temps réel fait apparaître plusieurs étapes, qui vont de la spécification, en passant par la conception jusqu'à l'intégration dans l'environnement réel. Nous nous sommes intéressés à l'étape de spécification des systèmes temps réel.

Les méthodes de spécification actuelles, ont toutes été élaborées afin de formaliser et de démontrer des propriétés opérationnelles : interblocage, famine, ou des propriétés de terminaison, ... Ces méthodes ont été ensuite enrichies d'extentions temporelles. Notre contribution a consisté à faire une étude critique de ces méthodes, puis à proposer une approche alternative, dédiée à l'expression et l'analyse des contraintes temporelles. Nous montrons dans cette étude critique la complémentarité possible de certaines méthodes avec notre approche et relevons les idées et les concepts qui nous ont inspirés dans notre démarche.

Si l'on sait formaliser une contrainte fonctionnelle ou opérationelle et vérifier si telle exécution satisfait ou non cette contrainte, le problème est différent dès qu'il s'agit de contraintes de temps. L'approche que nous développons propose une classification des contraintes temporelles et une méthode d'analyse. En outre, pour résoudre le problème de la représentation du temps nous proposons de déplacer le problème de l'expression et de la validation des contraintes temporelles sur un terrain syntaxique. La représentation des contraintes de temps sous forme de séquences permet de construire un analyseur de traces d'exécutions temporisées. Ce mode de représentation du temps est indépendant de tout aspect matériel ou système, et permet dès

PARTIE A

La spécification des systèmes temps réel

l'étape de spécification de raisonner sur des contraintes temporelles.

Ce mémoire est organisé en cinq chapitres répartis sur deux parties. L'objet de la première partie est de présenter et de comparer différentes méthodes de spécification des systèmes temps réel. Cette partie est composée de trois chapitres. Le chapitre I expose les concepts et les définitions concernant les systèmes temps réel. Nous définissons la notion de système, puis abordons le thème central du temps. Après quelques définitions de notions liées aux systèmes temps réel telles que le déterminisme, le synchronisme, le parallèlisme, nous introduisons un format de traces comportant le temps de manière quantitative.

Le chapitre II a pour objet de présenter les différentes étapes du cycle de vie d'un système temps réel ainsi que les méthodes qui s'y rattachent. Il souligne l'importance de l'étape de spécification sur laquelle porte notre travail.

Le chapitre III présente les grands types de méthodes de spécification des systèmes temps réel. Deux grandes familles sont abordées : les approches systèmes de transition pour lesquelles il est possible de dériver un format commun de traces, et les approches logiques.

La deuxième partie présente l'aspect le plus original de notre contribution. Elle est composée de deux chapitres. Le chapitre IV présente le contexte général de notre approche. Après une classification des contraintes temporelles, nous proposons une méthode d'analyse structurée de ces contraintes. Une contrainte temporelle pouvant être constituée d'un ensemble complexe de contraintes temporelles plus simples liées par un ensemble d'opérateurs (conjonction, périodicité, ...), nous proposons de décomposer toute contrainte temporelle sous forme d'un ensemble de contraintes temporelles de base. Nous proposons pour ces contraintes de base, une représentation formelle qui nous permet de les recomposer, et ceci afin d'obtenir l'expression formelle de la contrainte initiale. Cette dernière forme peut être ensuite transformée en un analyseur syntaxique temporel.

Le chapitre V présente la définition d'un des opérateurs (liant les contraintes temporelles) les plus importants et le plus complexe, celui qui permet de composer (au sens conjonctif) un ensemble de contraintes. Après avoir défini cet opérateur pour des contraintes de base, on indique comment croiser une contrainte de base à un ensemble de contraintes.

CHAPITRE I

DÉFINITIONS ET TERMINOLOGIE

Nous présenterons dans ce premier chapitre quelques définitions concernant les systèmes temps réels, puis ferons une classification de ces systèmes. Enfin nous évoquerons quelques notions généralement associées à la problématique du temps réel, et ceci afin de définir la terminologie utilisée tout au long de ce mémoire.

1. Définitions générales

Pour définir les Systèmes Temps Réels (STR), on va s'attacher au sens de chacun des mots composant cette expression :"Système", "Temps" et "Réel".

1.1. Système

La notion de système peut recouvrir un grand nombre de domaines (économique, politique, production, ...), nous allons chercher à en donner une définition au sens "informatique".

Le Larousse [Larousse 1983] nous donne la définition suivante : "Un système est une combinaison d'éléments qui se coordonnent pour concourir à un résultat ou de manière à former un ensemble".

problèmes de commandes de vol (électronique ou mécanique pour les gouvernes), les problèmes de commande du moteur. Ce dernier cas, peut intégrer à la fois des notions d'aérodynamique, de mécanique ou de thermodynamique, mais aussi des propriétés de robustesse, ainsi que d'autres contraintes globales de fonctionnement ...

Cette frontière fait apparaître des problèmes appartenant à des domaines très différents, on aura donc une terminologie, axée sur la systémique, c'est à dire sur la coopération entre différents sous-systèmes. La réalisation du problème portera alors plus, sur l'assemblage de différentes entités, celles-ci pouvant faire l'objet d'un développement parallèle.

On remarque donc que la définition du système est un préalable essentiel à la conception d'une application. Elle définit un certain niveau d'abstraction avec lequel le système sera vu et réalisé. On peut donc donner notre propre définition de système (au sens informatique) :

DÉFINITION 1 "SYSTÈME"

Un système identifie un ensemble d'éléments et donc défini la frontière qui le sépare de son environnement. La définition de cette frontière permet d'identifier le problème de la conception du système à un domaine d'application et à un niveau d'abstraction du problème.

1.2. Le temps

La richesse des expressions comportant des références au temps, illustre la difficulté de définir rigoureusement cette entité. On parle "des effets du temps" comme si le temps était une entité active et destructrice sur toute chose, on trouve "le temps long" comme si, le temps se dilatait parfois, ... Nous allons de ce fait présenter le temps selon différents points de vue, philosophique, ou scientifique, et en dégager deux idées générales qui nous serviront ensuite de base dans notre analyse des contraintes temporelles.

Qu'est-ce que le temps ? Comment se mesure-t'il ?

Ces questions se posent tant au niveau philosophique, que scientifique. Au point de vue religieux le temps est une notion liée à la vie terrestre, il est mesuré par différentes étapes, qui sont l'occasion de cérémonies (Baptême, Confirmation, Eucharistie ...). La période terrestre achevée, l'âme rejoint un autre état, celui-là éternel. Le temps existe-t'il alors encore ? N'est-ce pas une notion contraire à celle de l'éternité ?

Au point de vue philosophique, on relève deux concepts différents :

- le temps est une idée qui résulte en nous de la comparaison de deux états successifs. Ainsi Laplace dira "Le temps est pour nous l'impression que nous laisse dans la mémoire, une suite d'événements dont nous sommes certains qu'ils ont été successifs" ou pour Leibniz "le temps est l'ordre des existences successives".
- le temps est une notion liée au mouvement. Ainsi pour Pascal dans [Pascal 1700] "l'essence du temps est la mesure du mouvement" ou pour Diderot dans [Diderot 1800] "le temps est le calcul du mouvement relatif à la postérité est à l'antériorité". D'Alembert [Alembert 1800] définissait la pesanteur par rapport au temps : "Sans connaître la cause de la pesanteur, nous apprenons par l'expérience que les espaces décrits par un corps qui tombe sont entre

La définition d'un système établit un ensemble, donc une frontière, qui place le système à l'intérieur et le reste à l'extérieur. La recherche de cette frontière est un préliminaire fondamental qui va permettre d'identifier les éléments essentiels. Dans le cas de systèmes informatiques, cela permettra de distinguer les éléments matériels des éléments logiciels et d'ébaucher une solution.

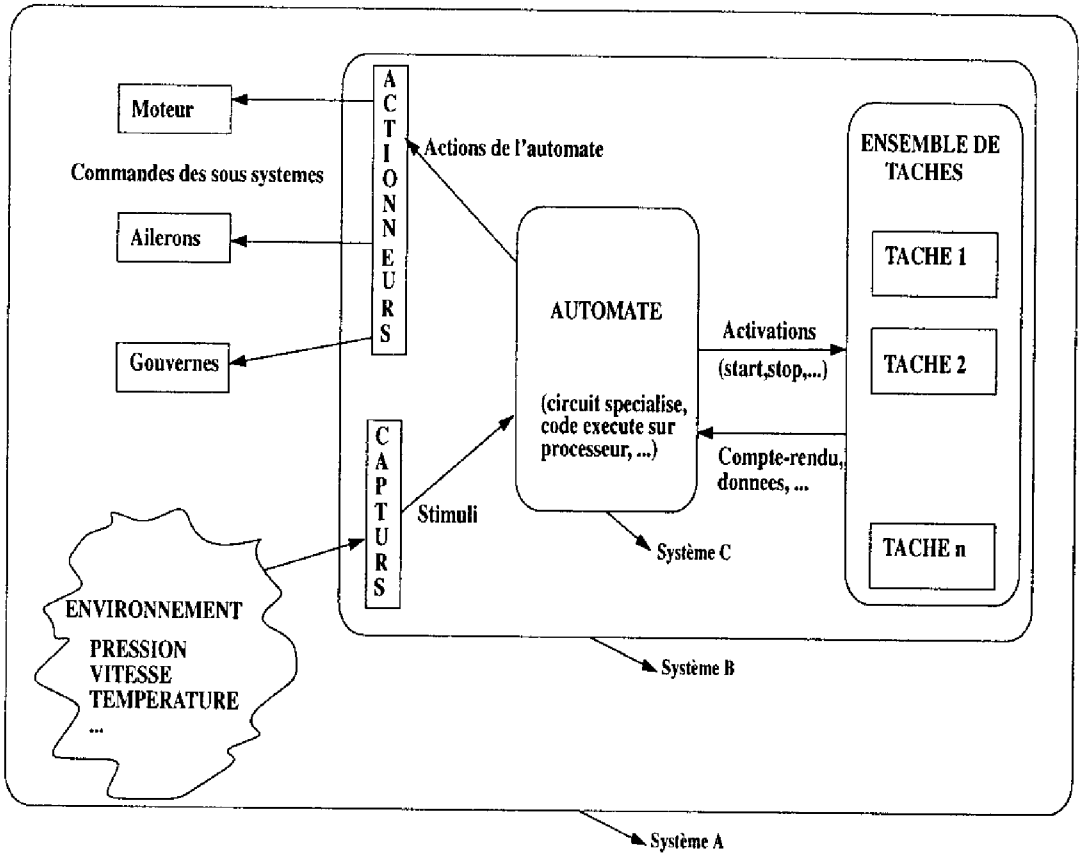


Figure I-1 : Définition d'un système temps réel

Dans la figure I-1 (représentant un système avionique), on observe trois frontières possibles. Pour la frontière C, le problème consiste à concevoir un automate de contrôle. Il s'agit alors de décrire tous les états du système en fonction des événements d'entrée et de produire des actions de sortie. La terminologie de description du système sera celle des machines à états, et les problèmes rencontrés pourront être l'explosion du nombre d'états, les problèmes d'interblocage, d'exclusion mutuelle, ...

La frontière B impose de concevoir un ensemble de tâches permettant de répondre aux fonctions du système ne pouvant s'exécuter de manière instantanée. En outre, il s'agira aussi de connecter l'automate à des entités physiques que sont les capteurs et les actionneurs. On aura d'une part, une terminologie correspondant à la gestion des tâches et d'autre part, à des concepts de protocoles d'échanges de données, avec les périphériques d'Entrées/Sorties (E/S).

Avec la frontière A, le problème est envisagé dans sa globalité. Donc il s'y greffera les

instants distincts du "passé"(avant t_1) sont ordonnés. Cela exprime que le passé est déterministe. L'absence de linéarité à droite, (qui est de fait, implicite dans cette formule) indique que le futur comporte des dates incomparables (au sens ou R n'est pas définie pour tout couple d'instants). Ceci peut être interprété comme l'indéterminisme du futur, où des destins peuvent suivre différentes voies. Enfin la densité du temps exprime, qu'entre deux dates, existent une infinité d'instants.

Ce modèle de temps est celui sur lequel est basé le calcul scientifique. Il permet le passage aux limites, le calcul différentiel, ... Il y a cependant d'autres définitions du temps plus simples mais suffisantes pour traiter des applications temps réel :

DÉFINITION 3 **"Le temps logique"**
C'est un cadre temporel (T,R) ou R est irréflexive, antisymétrique.

Les propriétés de ce dernier type de temps correspondent à celles de la fonction de précédence d'événements qui correspond à une représentation simple du temps.

De plus la notion de simultanéité d'événements à un sens pour cette définition du temps : dater des événements, c'est définir une correspondance entre un ensemble d'événements et un domaine temporel. La notion de simultanéité caractérise les propriétés de cette relation. Pour une vision discrète du temps, admettant donc le principe de simultanéité, cette relation comme l'indique la partie gauche de la figure I-2, représente une fonction, non injective : tout événement a une date unique, par contre deux événements peuvent avoir la même date. Par contre pour une vision continue du temps, cette fonction est une injection : tout événement a une et une seule date (elle n'est pas surjective), et la notion de simultanéité d'événements est inconcevable.

C'est la notion discrète du temps que nous adoptons par la suite, en effet la notion de simultanéité est nécessaire pour traduire la notion de parallélisme et le fait que tout système aussi performant soit-il, ne peut ordonner tous ses événements d'entrée. En outre, c'est une représentation simple qui nous permettra de modéliser l'écoulement du temps par un événement.

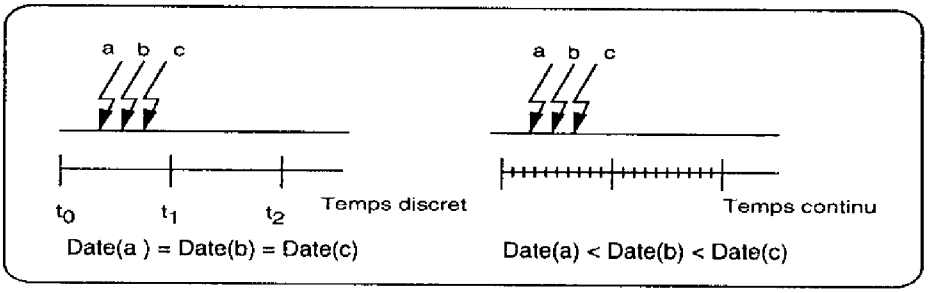


Figure I-2 : Temps théorique et temps logique

"réel" intervient pour exprimer que l'environnement est pris en compte de manière réaliste, c'est à dire que le système interagit de manière compatible avec la dynamique du procédé. Par exemple, un système de prévision météorologique doit prélever ses données et les traiter en un temps suffisamment court afin qu'elles ne soient pas périmées. Autre exemple, un robot autonome qui doit évaluer sa trajectoire, suffisamment rapidement, relativement à sa vitesse de déplacement.

eux comme le carré des temps".

On retrouvera ces deux notions dans les différents modèles de spécification des STR qui incluent le temps. La première idée présente une vue discrète du temps, son écoulement est matérialisé par une suite d'événements. Ceci implique aussi que tout événement peut être un marqueur temporel, dans la mesure où l'on est capable d'en ordonner les occurrences. On retrouvera ce concept dans les langages synchrones (présentés au chapitre 3).

La deuxième idée associe le temps au mouvement, dont il prend le caractère continu qui oppose cette notion à la précédente. On peut citer par exemple la rotation de l'aiguille autour de son cadran, ou l'écoulement de l'eau dans une clepsydre. Au point de vue physique (mécanique et astrophysique), c'est la deuxième idée du temps continu qui prévaut. En effet si la mesure physique du temps est discrète, il prend dans les équations un caractère continu, pour permettre par exemple le calcul différentiel, ou le passage aux limites. Il faut noter cependant, chez les thermodynamiciens, que le temps revêt une forme inhabituelle, celle d'une grandeur : l'*Entropie*. Cette notion est parfois associée à l'écoulement du temps. Cette quantité mesure le désordre moléculaire. En effet un des grands principes de la thermodynamique énonce que la variation de cette grandeur est toujours positive. Cette notion rejoint l'idée du vieillissement.

Avec une vue système à événements discrets des systèmes informatiques, il n'est évidemment pas question d'associer le temps à un mouvement continu, ni de considérer le temps comme un vieillissement ou un désordre "algorithmique" du système. Il est naturel de reprendre à notre compte le premier point de vue philosophique, qui l'interprète comme l'idée qu'une suite d'événements nous laisse en mémoire. Effectivement, les concepts de mémoire et de séquences d'événements sont directement transposables en termes informatiques (respectivement segment mémoire, interruptions, pile).

Alors que dans l'interprétation continue du temps, la notion de simultanéité d'événements est difficile à concevoir, dans l'interprétation discrète, deux événements simultanés sont confondus, et représentent deux facettes d'un même événement sous-jacent. La notion de simultanéité nous amène aussi à distinguer deux interprétations du temps : le temps continu qui est la représentation la plus générale que l'on peut faire du temps et un temps discret, qui est un concept plus simple et suffisante à nos besoins de formalisation.

Il est intéressant de représenter le temps de manière formelle, car l'expression des propriétés énoncées ci-dessus deviennent alors concrètes et non ambigües.

DÉFINITION 2 "TEMPS THÉORIQUE"

Le temps est caractérisé par un couple (T, R) où T est un ensemble d'instants non vide et R une relation sur T, appelée relation de précédence. R est une relation binaire s'appliquant sur les éléments de T, où $\forall t_1 \in T, \forall t_2 \in T, R(t_1, t_2)$ s'interprète comme "t_1 précède t_2", et R vérifie les propriétés suivantes :

- **Irréflexive** $(\forall t_1 \in T) \neg R(t_1, t_1)$
- **Antisymétrique** $(\forall t_1 \in T)(\forall t_2 \in T)((R(t_1, t_2) \wedge (R(t_2, t_1)) \Rightarrow (t_1 = t_2))$
- **Linéaire gauche** $(\forall t_1 \in T)((\forall t_2 \in T)(\forall t_3 \in T)(R(t_2, t_1) \wedge R(t_3, t_1))) \Rightarrow$
  $((t_2 = t_3) \vee R(t_2, t_3) \vee R(t_3, t_2))$
- **Dense** $(\forall t_1 \in T)(\forall t_2 \in T) R(t_1, t_2) \Rightarrow ((\exists t_3 \in T) R(t_1, t_3) \wedge R(t_3, t_2))$

Si les premières propriétés se comprennent aisément, la linéarité à gauche exprime que deux

2. Classification des STR

Harel et Pnueli [Harel 1985] ont introduit les termes transformationnels et réactifs. Ces termes distinguent deux grandes classes de systèmes, non forcément disjointes. Une représentation

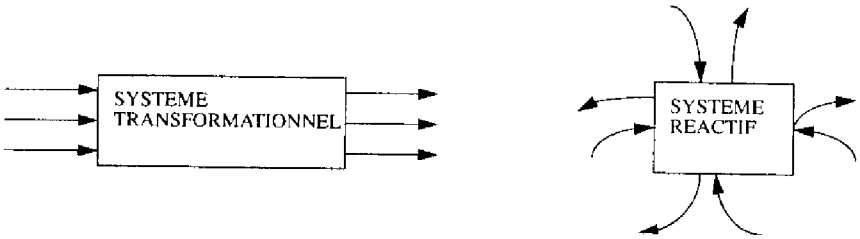


Figure I-3 : Représentation traditionnelle des systèmes Réactif et transformationnel

désormais classique des systèmes réactifs ([Harel 1985]) et transformationnels est présentée dans la figure I-3. Les STR transformationnels traitent des flux de données, le flux de sortie étant directement lié au flux d'entrée, tandis que les STR réactifs sont soumis à des stimuli, qui arrivent de façon sporadique, et dont la réaction doit être instantanée.

2.1. Les systèmes transformationnels

Le terme transformationnel qualifie les applications qui traitent essentiellement des données, et qui sont pauvres en contrôle. Leur rôle est en général de réaliser une fonction de transfert entre des données d'entrées et des données de sorties. Le temps intervient dans ces systèmes comme une contrainte de performance. C'est la performance du système qui rythme le flux d'entrées de données. On peut citer par exemple les systèmes de traitement d'images (compression, ...).

2.2. Les systèmes réactifs

Les systèmes réactifs sont en interaction permanente avec leur environnement. Ils réagissent à des stimuli (voir la définition ci-dessus), de manière quasi-instantanée pour ne pas perdre d'événements d'entrée. Ces systèmes se doivent d'être suffisamment rapides pour accepter toute une série de stimuli qui seraient arrivés dans un très court laps de temps. C'est l'environnement, contrairement aux systèmes transformationnels, qui rythme le fonctionnement du système. Notre définition des systèmes temps réel, nous montre que cette distinction est un peu abusive, car ceux-ci présentent les deux caractéristiques.

Par la suite, nous ne nous intéresserons pas aux aspects transformationnels des STR qui soulèvent des problèmes en terme de satisfaction de contraintes de performances, mais plutôt aux aspects réactifs et à leurs "contraintes temporelles comportementales".

2.3. Les domaines du temps réel

D'après la définition que nous avons donnée des STR, ceux-ci peuvent intervenir dans beaucoup de domaines. Historiquement, les systèmes les plus réactifs, ou les plus critiques en terme de

Le terme "réel" correspond à la notion d'asservissement que l'on retrouve en automatique mais appliquée à des systèmes complexes, qui font appel non seulement à des calculs de type réflexe, mais aussi à des tâches de fond. Ce terme rappelle aussi que les échanges entre le système et l'environnement sont permanents et se font sous différentes formes. Un STR interagit avec l'environnement et cet échange peut se dérouler soit de manière continue, soit discrète. Une donnée continue est une grandeur physique ou une mesure qui a une valeur définie à tout instant sur un intervalle de temps donné. Une donnée continue dans le temps est continuellement disponible (par exemple : la valeur d'un capteur de pression). Une donnée discrète est une donnée qui n'est définie qu'à des instants isolés (par exemple : le changement d'un niveau sur un capteur, top d'horloge, interruption, ...). Bien sûr dans le cadre des systèmes de commandes informatiques que nous étudions, toutes les données sont échantillonnées.

L'échange entre un système et son environnement, donne au système une vue approximative et fugace de l'état de l'environnement, qui lui permet d'évoluer. On a vu précédemment que le système devait réagir assez rapidement pour ne pas perdre d'événement. En réalité, il peut y avoir un phénomène de mémorisation des événements qui peut se faire dans un sous système, servant d'interface entre les capteurs et l'automate. Cependant, le système global doit donner l'impression de les traiter de manière immédiate.

1.3. Définition des STR

Les systèmes temps réels peuvent, comme on les a déjà schématisés par la figure I-1 peuvent être découpé en deux entités ([Berry 1989] et [Andre 1991]) : l'automate, qui a un caractère **réactif**, c'est à dire que son comportement est dicté par l'état de ses entrées (ensemble de stimuli [1]), et dont le temps de réaction lui permet d'éviter d'ignorer des entrées, et une partie que nous appelerons **transformationnelle** (ensemble de tâches), qui a pour but d'effectuer des calculs sur des données (transformations). Ces tâches peuvent s'exécuter dans une durée supérieure à l'intervalle séparant deux entrées de l'automate. Cette partie est commandée par des signaux d'activation ou d'interruption, qui seront alors considérés comme des événements d'entrée pour l'automate. Elles peuvent aussi interagir sur l'automate par modification de variables internes ou par émission de compte-rendus d'exécution. On peut maintenant donner une définition de l'appelation "Système Temps Réel"

DÉFINITION 4 "SYSTÈME TEMPS RÉEL"

C'est un ensemble constitué d'éléments matériels et logiciels, qui comprend généralement une partie réactive pour traiter les données discrètes et une partie transformationnelle pour les données continues. Ce système est en relation avec l'environnement de façon à en acquérir les informations et à interagir sur lui, selon des délais cohérents avec la dynamique du processus concerné.

[1] On appelle stimulus, l'occurrence d'un événement (éventuellement valué), en provenance de l'environnement vers le système. Cette occurrence est furtive c'est à dire que si elle n'est pas traitée ou mémorisée à l'instant de son ocurrence, elle est perdue.

Ce déterminisme peut permettre de raisonner de façon prédictive sur le système et en particulier, il est possible de prouver pour des programmes séquentiels déterministes, des propriétés d'invariance et de vivacité ([Audurau *et al.* 1990]).

3.2. Le parallélisme

On a déjà dit que le système devait être capable d'assurer le traitement des entrées. Pour cela il est parfois nécessaire de diviser le système en sous systèmes parallèles et communicants. Ce parallélisme peut entraîner lors de sa conception, notamment à cause des caractéristiques des langages ou des systèmes d'exploitations utilisés, des comportements non déterministes.

En effet considérons deux tâches parallèles (type Ada ou Modula 2) dont on observe le comportement par les événements d'entrées et de sorties. Si le programmeur spécifie l'ordre partiel dans lequel les instructions sont exécutées, il ne peut en revanche distinguer quelle a été la séquence d'entrées requise pour une sortie donnée.

Exemple :
Soient deux tâches T_1 et T_2 qui s'exécutent en parallèle :

T_1 : attend $e1$; || $T2$: attend $e2$;
emet $s1$; emet $s2$;

Si e_1 précède e_2 cela n'implique pas que l'on observera s_1 avant s_2. En effet la tâche T_1 peut être suspendue par le système d'exploitation entre l'occurrence de e_1 et l'observation de s_1.

3.3. Le synchronisme

Le problème évoqué précédemment, peut se retrouver dans d'autres langages que nous classons dans la terminologie des "langages Asynchrones". En effet dans ces langages, on peut retrouver le même problème évoqué dans l'exemple précédent où l'on rencontre des tâches dont la vitesse d'exécution relative est imprévisible. Sur un système monoprocesseur, c'est le "scheduler"[3] qui décide de l'ordre d'exécution des tâches. Sur une architecture multiprocesseur, les tâches auront chacune leur propre vitesse d'exécution, et on ne pourra pas ordonner de manière statique les événements observables qu'elles produiront.

Par opposition à ces principes, ont été développés une classe de langages dits "synchrones" qui permettent de produire du code déterministe, même en présence d'un opérateur de parallélisme [4]. Cette nouvelle classe est basée sur l'hypothèse "synchrone" ([Berry *et al.* 1987]).

DÉFINITION 6 "HYPOTHÈSE SYNCHRONE"

L'hypothèse synchrone amène à considérer des systèmes idéalisés qui réagissent aux entrées externes par une mise à jour instantanée de leur état interne et en produisant immédiatement leur sortie. Un programme synchrone se comporte comme si

[3]Nom usuel donné au gestionnaire de tâches, dans un système d'exploitation

[4]Cet opérateur a l'apparence de l'opérateur classique, mais lors de la compilation, il recombine ensuite les processus parallèles en un seul processus séquentiel

temps de réponse, se trouvaient surtout dans le domaine du spatial (on parle alors de systèmes embarqués[2]).

On les retrouve maintenant dans les protocoles de communication, dans l'industrie manufacturière avec les concepts de "flux tendu", mais aussi dans les appareils "grand public", comme par exemple, les systèmes de flash des appareils photographiques, qui mesurent en temps réel l'intensité lumineuse à pourvoir.

On trouve dans [Benveniste 1991] une intéressante classification des domaines selon leur ordre de complexité :

- Les applications qui ne comportent que des tâches purement séquentielles. On retrouve notamment ces applications dans le domaine des systèmes de production. Ces applications nécessitent un niveau de sécurité important, car un dysfonctionnement peut engendrer des défauts sur toute une chaîne de fabrication. L'aspect temps réel apparaît au niveau des contraintes de performances. La partie transformationnelle, est prépondérante dans le système global.
- Les systèmes de contrôle et de surveillance de procédé à grande complexité, comme les transports en commun, ou les commandes de vol d'un avion ... Ils présentent une part réactive importante. Ces systèmes, souvent distribués, doivent vérifier des contraintes de sécurité critiques (car elles peuvent mettre en danger des vies humaines).
- Les systèmes embarqués (non réparables) sont les plus complexes de cette classification. Ils peuvent être hautement distribués, et sont généralement fortement réactifs. Ils comportent aussi des systèmes heuristiques type "meilleur effort". On les trouve essentiellement dans le domaine militaire (système de suivi de cibles, navigation à basse altitude, ...). Ils nécessitent une robustesse à la défaillance maximale par rapport aux systèmes des deux premières catégories, et donc des efforts important pour démontrer des propriétés de sécurité ou de prototypage.

3. Notions au cœur des systèmes temps réel

Dans la problématique du temps réel apparaissent souvent les termes déterminisme, parallélisme, synchronisme ou asynchronisme. Que recouvrent donc ces notions ?

3.1. Le déterminisme

Comme on vient de le voir dans la classification précédente, les STR exigent tous un niveau de sécurité de fonctionnement élevé. Un des moyens pour assurer cette sécurité, est de garantir que le comportement des futures applications soit déterministe.

DÉFINITION 5 "DÉTERMINISME"

Un système est dit déterministe si ses exécutions sont reproductibles c'est à dire si, pour une même configuration d'entrée et pour un même état interne, il présente toujours la même sortie.

[2]Un système est dit embarqué s'il est intégré au dispositif physique avec lequel il interagit et dont il partage les contraintes : températures excessives, humidité, ...

Dans l'approche que nous proposons en deuxième partie, nous reprenons à notre compte ce principe, ainsi que l'idée que les contraintes de performances ne relèvent que de l'étape de conception.

3.4. Les contraintes temporelles

Les interactions d'un STR avec son environnement sont par nature temporellement contraints. Donnons quelques exemples de contraintes temporelles tirées du contexte d'un système téléphonique :

Exemple :
i) On doit attendre la tonalité de marche avant de pouvoir composer son numéro de téléphone.
ii) On doit composer chaque chiffre du numéro en "au plus" 3 secondes par rapport au chiffre précédent.
iii) Tous les chiffres doivent être composés dans un intervalle de temps compris entre 6 et 30 secondes.
iv) Toutes les unités décomptées (une unité de taxation est définie relativement au tarif en vigueur) de téléphone doivent être signalées par un message sonore.

Ces exemples nous montrent que les contraintes temporelles peuvent être des contraintes de précédence (i), des contraintes de type maximum (ii) c'est à dire exigeant l'occurrence d'un événement avant l'expiration d'un délai, des contraintes de durée (iii), ou des contraintes périodiques (iv). Le cas (iv) montrent que l'unité de temps employée peut être variable (multiple d'unité de temps), ou même exprimée en terme d'occurrence d'autre signaux que ceux de type horloge.

Considérons qu'une horloge produise des événements périodiques "h" marquant l'écoulement d'une unité de temps (idée introduite dans [Delfieu 1993]). En observant les événements venant de l'environnement s'entrelaçant avec les événements horloges, on peut faire les remarques suivantes :

- Dans tous les cas présentés, l'origine des temps n'est pas unique, mais relative par rapport à un événement. On peut illustrer (ii) par le schéma I-4, où une séquence observable, correspondant à la contrainte est présentée (en considérant que le numéro de téléphone ne comporte que trois chiffres).
- Une origine des temps est liée à l'occurrence d'un événement.
- Une date peut se définir, comme une origine des temps et une durée.
- Une durée peut se définir comme un nombre entier d'occurrences d'un événement horloge depuis un événement marquant l'origine des temps.

Les séquences temporisées

L'introduction d'un événement marquant l'écoulement du temps permet de représenter le temps dans des traces, que nous appelons des séquences temporisées. Une contrainte temporelle peut alors se définir par l'ensemble des traces temporisées qui la respecte comme dans l'exemple précédent ou s_1, s_2 et s_3 définissent la contrainte (ii).

Exemple :
Si l'on reprend l'exemple (ii), les séquences suivantes répondent toutes à la contrainte : (soit D,

ces actions internes étaient exécutées par une machine extrêmement rapide, dont les calculs ne prendraient pas de temps. Le programme est inactif entre les réactions, il attend simplement les entrées suivantes.

Les conséquences suivantes peuvent être énoncées :

- La réaction est synchrone avec l'événement déclencheur.
- Deux actions sont synchrones, si et seulement si, elles appartiennent à la même réaction.
- Une réaction ne peut être interrompue par une autre entrée.

L'hypothèse synchrone peut s'interpréter ou se comprendre comme un concept en opposition au concept d'interruption. Le style de programmation interruptif est la façon de traiter les événements par rupture de séquences, c'est à dire que chaque événement provoquera un déroutement avec exécution d'une fonction associée, qui peut elle-même être interrompue à son tour par une autre interruption de priorité supérieure ...

Le style synchrone nécessite que la réaction soit atomique (c.a.d. ininterruptible). De plus, cette hypothèse implique un temps de réaction suffisamment court, pour ne pas perdre une série d'événements qui seraient arrivés en rafale. [5] Cela pose une hypothèse sur la rapidité du système, tel qu'il sera réalisé physiquement. Cette hypothèse permet de reléguer les problèmes de performances à des niveaux plus aval du cycle de vie, pour se concentrer sur des problèmes de cohérence comportementale de la spécification. Cette hypothèse permet d'adopter un style de programmation particulièrement adapté à la résolution des systèmes complexes et critiques que sont les systèmes réactifs.

La programmation synchrone améliore considérablement la productivité dans le domaine de la création de logiciel puisqu'elle différencie ce qui relève de la commande ou du contrôle (partie réactive), des autres fonctions de calcul ou de traitement de l'information (partie transformationnelle), ce qui diminue la complexité globale.

En outre, les langages synchrones proposent des primitives idéalisées (parallélisme, interruption, instructions ne consommant pas de temps) qui permettent aux programmeurs de considérer que leur programme réagit de manière instantanée aux événements externes. L'hypothèse synchrone peut mieux se résumer par les propositions suivantes:

- L'ordonnancement des événements d'E/S détermine le comportement du programme et non pas l'inverse.
- Un événement ne peut être mémorisé ni ignoré.
- Le modèle établit que le traitement d'un événement est ininterruptible et la machine suffisamment rapide pour prendre en compte tous les événements d'entrée.

Cette hypothèse a été choisie dans de nombreux travaux, notamment pour la conception d'un module de contrôle de commande [Elkhattabi 1993] dans le cadre de projet CASPAIM [Bourey *et al.* 1989] (la partie du système de surveillance), dans la spécification de contrôle de cellules flexibles [Marty 1994], dans la conception et la réalisation de systèmes temps réel [Peraldi 1993].

[5] Théoriquement, les événements ne peuvent arriver strictement en même temps et on emploie cette expression pour exprimer que des événements sont arrivés à des dates si proches que le système ne peut les ordonner. Il les considère alors, comme étant arrivés simultanément

ces traces, comme dans l'exemple précédent où entre deux occurrences des événements 1, 2 ou 3, il n'y avait pas plus de trois occurrences de "h". Les remarques précédentes nous permettent d'énoncer **la définition** des contraintes temporelles que nous avons adoptée dans le cadre de notre contexte d'étude :

DÉFINITION 8 "EXPRESSION D'UNE CONTRAINTE TEMPORELLE"

Toute contrainte temporelle peut s'exprimer uniquement en terme d'occurrences d'événement :
- L'écoulement du temps est materialisé par l'occurrence d'un événement spécifique.
- Les durées sont materialisées par un nombre d'occurrences d'événements.
- Une contrainte temporelle s'exprime comme une propriétée syntaxique sur des séquences temporisées.

4. Conclusion

Dans ce chapitre, pour établir une définition des systèmes temps réel, nous sommes revenu sur la notion de système ainsi que sur le problème de la représentation du temps. De plus, nous avons exposé les concepts liés à la conception des systèmes temps réel. La présentation de ces concepts permet de cerner le domaine des systèmes temps réel et d'évoquer les problèmes qui s'y rapportent. La représentation du temps que nous avons adoptée, nous a permis de proposer un format d'expression de traces temporisées pour lequel nous allons montrer au chapitre III qu'il peut être produit par toute méthode de spécification (type système de transitions) incluant le temps. En outre ces traces constituent le point d'entrée de la méthode que nous proposerons au chapitre IV. Auparavant, nous montrons dans le chapitre suivant l'importance de la spécification dans le cycle de vie d'un système temps réel, car c'est à cette étape que se situe notre approche.

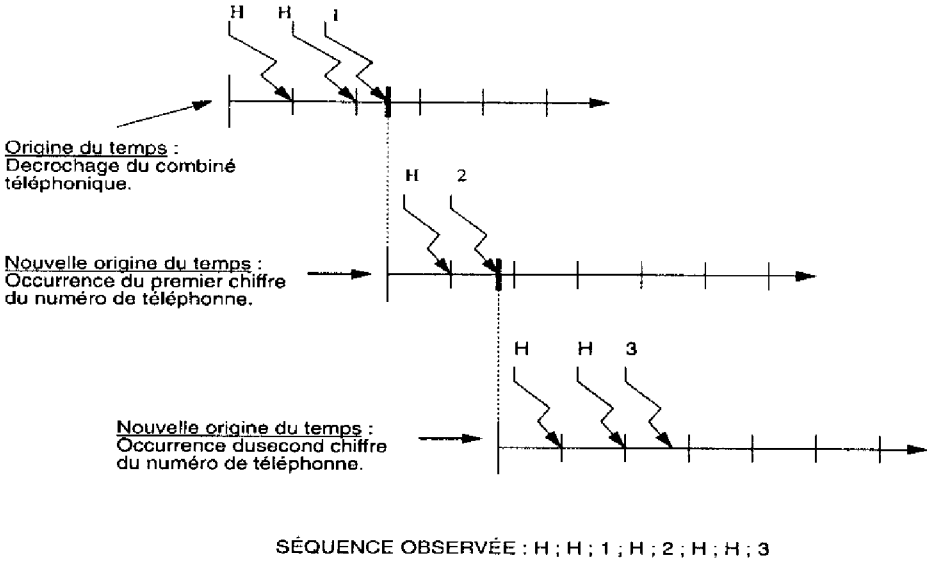


Figure I-4 : Origine des temps relatives

l'événement correspondant au décrochage du combiné téléphonique)
$s_1 = D; 1; H; 2; H; H; 3$
$s_2 = D; H; 1; H; H; 2; H; H; 3$
$s_2 = D; H; H; 1; 2; H; H; 3$

DÉFINITION 7 "CONTRAINTE TEMPORELLE"

Une séquence temporisée est une séquence composée des événements qui sont impliqués dans la contrainte et d'un événement marquant l'écoulement du temps. Une contrainte temporelle est un ensemble de séquences temporisées.

Une séquence d'événements peut être produite par toute méthode de spécification de type "système de transitions" et donc à priori, une séquence temporisée peut être aussi obtenue par une version temporelle ou temporisée d'un tel formalisme, comme nous le verrons au chapitre III. Ces traces peuvent donc constituer à la fois un lien et un format commun entre tous ces formalismes. Elles peuvent en outre, devenir le point d'entrée pour un outil de type "analyseur syntaxique de traces" comme nous le préciserons au chapitre IV.

Un ensemble de séquences temporisés peut être caractérisé par une propriété syntaxique sur

CHAPITRE II

LE CYCLE DE VIE DES SYSTÈMES TEMPS RÉEL

L'objet de ce chapitre est de présenter les différentes étapes de la réalisation d'un système. On évoquera pour chacune d'entre elles, les problèmes spécifiques soulevés par les aspects Temps Réel, et les méthodes qui s'y rapportent.

1. Le cycle de vie d'un logiciel

La mise en œuvre de tout système est un processus qui peut être décomposé en plusieurs étapes :

- L'expression du besoin : le client donne une description de ses besoins, en général de façon informelle, en précisant l'inventaire des besoins fonctionnels, des capacités de stockage, du matériel existant sur lequel le futur système devra se greffer, les contraintes temporelles que le système doit vérifier ...

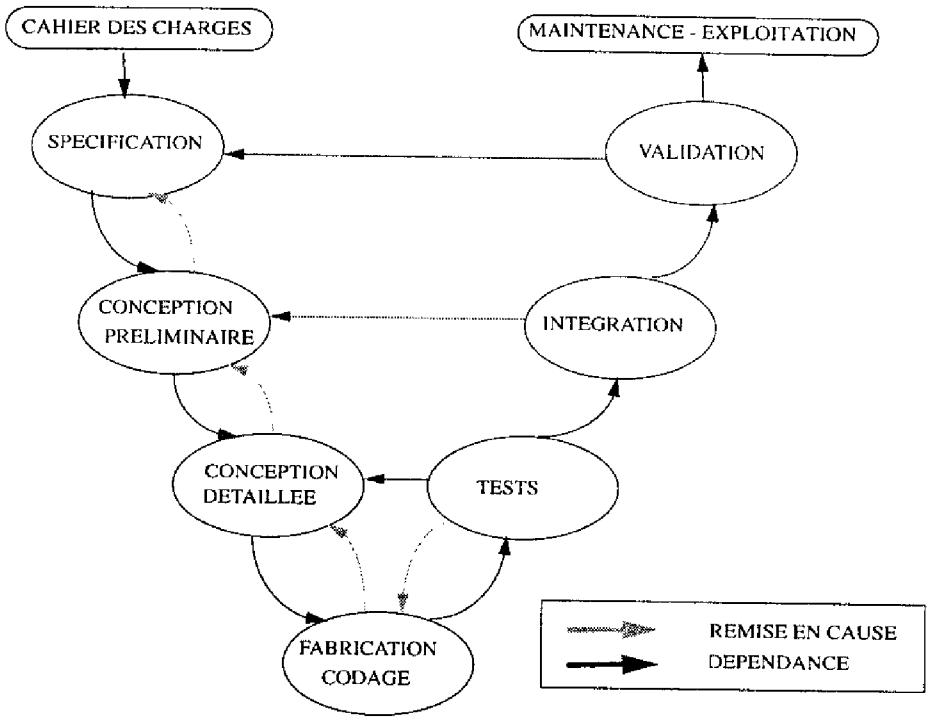


Figure II-1 : Cycle de vie en V

2.2. Correction ou conformité par rapport au cahier des charges

S'il y a un seul sujet d'accord entre les programmeurs, les ingénieurs et les clients, c'est la nécessité d'établir que tout programme réalisé soit **correct**. Seulement la notion de "correction" peut s'interpréter de plusieurs manières : un fonctionnement conforme au cahier des charges, sans erreurs, économe en temps CPU et en occupation mémoire, robuste, portable, convivial, évolutif, ...

Tous ces paramètres ne peuvent pas évidemment être pris en compte tous en même temps. La notion de correction mérite donc d'être examinée dans un certain contexte d'interprétation. On peut dégager deux domaines d'interprétation plus généraux que ceux précités, la correction syntaxique et la correction sémantique ([Turski 1987]).

La correction syntaxique exprime que le programme est correctement écrit par rapport aux règles de construction syntaxiques qui définissent le langage dans lequel il est écrit. Ce type de correction n'intervient pas au niveau de la spécification de programme, c'est un niveau de détail qui relève de l'étape de codage.

La correction sémantique exprime que le programme réalise ce que l'on attendait de lui. C'est pour déterminer ce sens général qu'intervient la spécification. La spécification est un processus d'abstraction, qui consiste à représenter un système, par ses aspects essentiels. On représente, au premier niveau, le système comme une boîte noire, où les seuls éléments décrits sont les flux de données et de contrôle, cette boîte pouvant être ensuite décrite de manière plus fine. La spécification s'attache donc à formaliser une représentation du sens général du programme. Elle

- L'étude de faisabilité examine l'expression des besoins par rapport à l'environnement global du système, notamment les aspects humains, normatifs et techniques. Cette étape se réalise comme la précédente, de façon relativement informelle.
- La spécification, définie par rapport aux deux étapes précédentes, est une représentation plus ou moins formelle du cahier des charges. Cette représentation doit exprimer les besoins en terme de fonctionnalités, de contrôle et de données : elle détermine le "quoi".
- La conception détermine le "comment"; elle se décompose généralement en deux étapes :
  1. la conception préliminaire qui détermine une architecture globale du système, permettant d'implanter le futur système dans un éventuel environnement existant.
  2. La conception détaillée définit les méthodes à utiliser pour la réalisation des aspects matériels et logiciels du système.
- L'étape de fabrication. Elle consiste à construire les éléments constitutifs du système. Pour le logiciel il s'agira de codage et pour le matériel, de la fabrication de prototype.
- Les tests ont pour but de contrôler la bonne exécution du logiciel dans l'environnement matériel construit (il peut s'agir aussi d'un prototype).
- L'intégration vise à implanter le logiciel et/ou le prototype matériel dans son environnement réel d'exécution.
- La validation consiste à prouver que le système final intégré dans son environnement réel, répond bien au cahier des charges initial.

On retrouve ces étapes sur la figure II-1. Cette figure indique que chaque étape a une entrée qui est le résultat de la précédente. Un autre aspect de cette figure indique que chaque étape de la branche remontante valide son vis à vis. Ainsi l'étape d'intégration vise à valider les choix d'architecture fait lors de la conception préliminaire, ... Le rebouclage d'un état sur son précédent indique que chaque étape peut remettre en cause la précédente. Chaque phase est terminée par une vérification et une validation (V & V) dont les objectifs sont d'éliminer le plus de problèmes possibles relevant de cette étape.

2. La spécification

2.1. Importance de la spécification

Considérons qu'une erreur ait été commise à cette étape et qu'elle n'ait été détectée que lors de la validation finale. Tout le travail accompli pourrait alors être remis en cause. C'est à dire, non seulement le système complet mais aussi les choix fondamentaux de conception rendant ainsi toutes modifications impossibles. Une remise en cause de la spécification après la conception préliminaire est donc très coûteuse. La spécification a donc une importance majeure dans le cycle de vie d'un programme. En outre elle va servir de guide dans toutes les étapes suivantes. Le concepteur et le programmeur pourront s'y référer, pour vérifier que leur travail correspond bien au cahier des charges. Elle offre ainsi un cadre d'interprétation de la correction de programme.

question 2 : Que signifie "valider les contraintes temporelles" lors de l'étape de spécification d'un système temps réel ?
Nous allons affiner ces questions par un ensemble d'interrogations dans la section suivante. Celui-ci constituera un ensemble de critères d'évaluation de méthodes de spécification. Cet ensemble nous servira pour l'étude que nous mèneront dans le chapitre III.

2.4.1. Critères d'évaluation et de comparaison des méthodes de spécification de STR

Nous examinons dans ce paragraphe les exigences que l'on peut avoir par rapport à une méthode de spécification, mais aussi les différentes caractéristiques qui singularisent les méthodes existantes.

Les STR sont soumis à un grand nombre de contraintes. Contraintes de temps, de robustesse, de tolérance aux pannes, ... mais c'est le temps qui constitue donc le paramètre fondamental du problème. Les contraintes temporelles devraient pouvoir être exprimées dans un formalisme adapté qui permettrait avant tout choix technologique de raisonner ou de montrer des propriétés, comme cela existe pour les contraintes fonctionnelles. On a vu au premier chapitre les différents points de vue que l'on pouvait avoir par rapport au temps, aussi les méthodes diffèrent selon la façon qu'elles ont de prendre en compte le temps.

Un autre concept qui différencie les méthodes, est celui de la vision synchrone ou asynchrone qu'elles ont du système. Ce concept influera sur la façon de concevoir le système.

Par définition la spécification, devrait être aussi détachée que possible, de tout concept, matériel ou logiciel. Aussi la méthode de spécification, devrait éviter de limiter la conception future à une architecture ou à des techniques particulières.

On peut aussi se poser le problème du choix entre une approche logique ou de type "système de transitions". Les approches logiques permettent de représenter de façon assez abstraite et générale les systèmes, tout en permettant de vérifier certaines propriétés. Par contre elles modélisent avec difficulté la dynamique du procédé ce que font de manière naturelle les approches "système de transitions".

Une méthode de spécification doit être aussi un outil de communication, sans sacrifier son pouvoir d'expression. Certaines méthodes sont enrichies de multiples opérateurs et concepts mais perdent leurs caractéristiques de base (pouvoir de vérification de certaines propriétés) voire même leurs principes fondateurs (système devenant non déterministe, ...).

Les remarques que nous venons d'énoncer, introduisent quant au choix d'une méthode les questions suivantes :

- Comment intégrer le temps au modèle ?
- Le modèle intègre-t-il la modélisation des données et du contrôle ?
- La méthode de spécification doit-elle tenir compte d'une architecture cible ou bien peut-elle imposer un style de conception ?
- Comment vérifier de manière formelle des propriétés de programmes ?
- Quelle approche choisir : logique ou système de transition ?
- Quelle hypothèse choisir : synchrone ou asynchrone ?

définit donc un cadre général pour pouvoir répondre au problème de la correction sémantique d'un programme.

2.3. De la spécification à la mise en œuvre

Les modèles[1] de spécifications sont souvent plus faciles à interpréter, que les programmes correspondants. De ce fait les concepts employés sont moins dépendants de l'architecture. On peut par exemple citer la représentation de l'encapsulation ou de port de communication dans Lotos, la notion de synchronisation de processus dans les réseaux de Petri ou la notion de diffusion d'événements dans les Statecharts. Ces concepts permettent de se dégager des détails informatiques ou matériels pour se tourner vers le vocabulaire du domaine d'application, améliorant pour le client et le concepteur la connaissance générale du sens que l'on donnera au système.

Une fois la spécification établie, il est encore aujourd'hui nécessaire de passer d'une représentation (souvent graphique) à une autre (souvent textuelle) à cause des standards de conception (compilateurs ou architecture cible). La programmation ne devient alors qu'un processus de translation d'une forme à une autre. Cette étape de codage est difficile et source d'erreurs, puisque l'on passe de modèles expressifs et clairs, à des langages dans lesquels on devra implémenter les concepts abstraits du modèle précédent. La pauvreté expressive du langage relativement au modèle de spécification augmente considérablement la complexité et les risques d'erreurs du code obtenu.

On peut se demander [Harel 1987] si à plus ou moins moyen terme, cette étape ne va pas perdre de son importance, comme en témoigne les nombreux ateliers de génie logiciel (CASE : Computer Aided Software Engineering), qui produisent automatiquement du code à partir de spécifications textuelles ou graphiques. On peut citer notamment Statemate [Harel *et al.* 1990], qui produit à partir de schémas représentant des automates, du code C, ADA et VHDL. Le code obtenu étant est peu lisible, il est hasardeux d'envisager de le maintenir ; toute modification sera donc faite au niveau de la spécification et sera suivie d'une nouvelle génération de code ce qui permettra d'assurer un cohérence globale au niveau du cycle de développement (fichiers de documentation, tests, ...).

2.4. Spécification d'un système temps réel

Pour les systèmes temps réel, l'étape de spécification est cruciale. En effet, s'apercevoir d'une erreur de spécification lors de l'étape de validation en environnement réel de fonctionnement peut s'avérer désastreux. Si l'on s'aperçoit qu'un satellite en orbite ne correspond pas à ce que l'on voulait au départ, c'est évidement trop tard pour intervenir. D'autre part, pour ces systèmes, la construction de prototypes matériels est très coûteuse, ce qui relève encore l'importance de cette étape sur laquelle porte notre travail.

On peut se poser essentiellement deux grandes questions par rapport aux modèles de spécification des STR, et à leur relation aux propriétés temporelles:

question 1 : Quelles sont les qualités nécessaires à une méthode de spécification des systèmes temps réel ?

[1] On appellera modèle, la représentation de la spécification du système obtenue à l'aide d'un formalisme quelconque

n'auront pas faussé le modèle en introduisant des phénomènes d'interblocage, de famine, ...)

3. Conception de systèmes temps réel

Le comportement global ayant été fixé lors de la spécification, la seule inconnue restante est de savoir si l'on va pouvoir satisfaire les contraintes de performances. Les systèmes temps réel ont pour caractéristiques de se positionner à *la limite de la technologie actuelle*, et imposent de choisir un compromis coût/performances.
Cela remet donc en cause la conception classique et par la même, les aspects algorithmiques traditionnels (programme séquentiel), les aspects architectures (Von Neumann) et les systèmes opératoires.

Les solutions utilisées pour résoudre ces problèmes sont les suivantes :

- La parallélisation, tant au niveau du code, que de l'architecture ou du système d'exploitation multitâches.
- Des nouvelles architectures (RISCs, CISCs, ...).
- Des circuits spécialisés
- Des exécutifs temps réel.
- ...

Nous présentons par la suite un aperçu des méthodes de conception des systèmes temps réel, avec notamment un exemple de méthode incluant parallélisation de code et définition d'architecture parallèle.

3.1. La parallélisation de code

La parallélisation de programmes a des répercussions dans le domaine de l'algorithmique, de l'architecture et des systèmes d'exploitation.

Au niveau algorithmique

Elle implique de repenser le calcul en terme de blocs les plus indépendants possibles. Il s'agit d'essayer d'éclater les algorithmes en plusieurs entités, qui pourront s'exécuter en parallèle tout en concourant au résultat final. Ces entités devront avoir un besoin de communiquer minimal, pour éviter que le surcoût induit ne pèse sur le temps gagné.

S'il arrive parfois que l'on redécouvre sous l'angle du parallélisme des algorithmes de calcul connus (mais jugés inefficaces pour le calcul séquentiel), la recherche de nouveaux algorithmes est difficile et il n'y a pas de méthodes permettant de trouver un algorithme présentant un parallélisme optimal, pour un problème donné.

Le parallélisme que nous venons d'évoquer, est un parallélisme dit "gros grain", car il définit des "grosses entités" parallèles. Un autre type de parallélisation "grain fin" est employée; elle se place au niveau des instructions. Parmi ce type d'approches on peut citer les architectures type DATA FLOW, la machine de Dennis (pour un survol de ces approches [Vegdahl 1984]).

- Faut-il privilégier le pouvoir expressif ou le pouvoir communicatif d'un modèle ?
- Faut-il privilégier le pouvoir expressif par rapport au pouvoir déductif ?

2.4.2. Valider les contraintes temporelles

La terminologie temps réel est souvent associée, pour ne pas dire confondue, avec le domaine de l'optimisation des performances de systèmes. Certes, les applications temps réel doivent souvent s'exécuter très rapidement, à des fréquences CPU qui sont parfois à la limite de la technologie actuelle, comme, par exemple, des calculs de trajectoires ou de controle de réactions chimiques, ... Pour pallier à cette limite, on utilise généralement la parallélisation ou la vectorisation sur des architectures spécifiques.

Cependant, toute la difficulté de réalisation ne réside pas que dans la conception détaillée, surtout quand la complexité du contrôle prédomine par rapport au calcul sur les données. Il s'agit alors de modéliser le comportement pour assurer sa cohérence et sa fiabilité. Dans ce type d'application on pourra alors avoir aussi des contraintes temporelles de synchronisation de processus, ou des contraintes assurant un comportement de manière périodique.

Il y a donc deux types de contraintes: les contraintes de **performance**, qui portent sur des temps d'exécutions, et les contraintes temporelles de type **comportemental**, qui porteront sur le contrôle. Les contraintes de performance indiquent en général une limite pour le temps d'exécution, le temps final obtenu, pouvant bien sûr être très largement inférieur à la limite donnée. Les contraintes temporelles comportementales, spécifient des précédences entre des événements. Il ne s'agit plus d'être aussi rapide que possible, mais d'assurer que tel ou tel événement se produira à telle date avec une précision correspondant à un intervalle, ou bien après tel ou tel délai, ...

Au niveau de la spécification, le respect des contraintes de performances ne peut être assurée que si l'on connaît les caractéristiques de la machine d'exécution, et d'autres détails d'implémentation, ce qui est incongru à cette étape. Si l'on considère à cette étape qu'aucune hypothèse ne peut être faite sur la machine d'exécution, sauf à considérer qu'elle aura au moins la puissance minimale assurant la faisabilité du système, alors la spécification pourra permettre de :

- Vérifier que toutes les contraintes temporelles sont compatibles entre elles.
- Vérifier que le modèle temporel correspond bien à ce qu'attend le client.
- Déduire des propriétés temporelles non contenues dans le cahier des charges qui pourront servir de guide lors de l'implémentation (contrainte de performance).

  Exemple :
  Considérons que l'on déduit du modèle, qu'entre "a" et "b" s'écoule toujours au moins 3 unités de temps. Si "a" est l'événement déclencheur d'une tâche "A" et "b", l'événement déclenchant le calcul de la tâche suivante, cette propriété nous permet alors de connaître le temps maximum associé au calcul de "A" et donc une puissance de calcul nécessaire.

- S'assurer que les propriétés fonctionnelles (c'est à dire celle qui ne relèvent pas "stricto sensu" du temps : famine, exclusion mutuelle, ...) sont toujours vérifiées, après l'ajout des contraintes temporelles (par exemple on pourra vérifier, que les contraintes temporelles

CHAPITRE III

ETUDE CRITIQUE DES MÉTHODES DE SPÉCIFICATION DES SYSTÈMES TEMPS RÉEL

On a présenté au chapitre précédent quelles pouvaient être les qualités que l'on pouvait attendre d'une méthode de spécification des STR. Il y a trois de ces critères qui nous semblent fondamentaux pour la spécification des STR :

- Doit-on préférer une approche type "système de transitions' ou de type "logique" ?
- Approche Synchrone ou Asynchrone ?
- Approche textuelle ou graphique ?

Le premier de ces points identifie deux grandes classes de méthodes très différentes. En effet, les systèmes de transitions se focalisent sur la description de la dynamique du procédé, tandis que les approches logiques insistent plus sur la description statique et abstraite du système. Le deuxième point caractérise les méthodes selon leur façon de prendre en compte le temps et les événements. Enfin, le dernier point différencie les méthodes selon qu'elles utilisent une forme textuelle, ou graphique. Cette distinction n'est pas inutile, car elle indique si le critère de communicabilité a été choisi ou non au départ, comme un fondement de la méthode. Dans

Ces approches considèrent toutes, que la séquence des opérations doit uniquement venir de la dépendance de données. Ainsi les instructions indépendantes s'exécutent en parallèle (version multiprocesseur de la machine de Dennis [Dennis 1979]). Si cette approche permet d'extraire tout le parallélisme d'un code source, elle nécessite en revanche une architecture très spécifique. Il faut en outre mémoriser les instructions qui sont dépendantes d'autres données non encore présentes, ce qui peut engendrer une saturation de la mémoire. De plus cette approche ne tire pas partie du parallélisme qui pourrait être établi dans la phase algorithmique.

Un exemple d'approche de parallélisation de code

L'approche C.S.T.R ([Dours *et al.* 1992] et [de Michiel 1994]), présente un compromis par rapport aux deux points de vue précédent. Elle présente, d'une part, la possibilité d'exprimer un parallélisme "gros grain" grâce à un langage de description parallèle et d'autre part, le compilateur développé pour ce langage permet d'extraire un parallélisme grain fin, compte tenu des contraintes temporelles de l'application et d'une architecture cible. En outre une architecture cible a été définie. Elle est de type MIMD, modulaire et reconfigurable, pour tenir compte de la diversité des applications temps réel traitées, et réalisable à partir de processeurs standard, et de cartes du marché.

Cette approche s'intéresse plus particulièrement à des applications de traitement du signal. Ces applications qui sont reliées à un ensemble de capteurs et d'actionneurs, doivent traiter des flux de données à haut débit et peuvent se décomposer en sous-systèmes de traitement. Ces applications ont un caractère transformationnel marqué, en raison des calculs importants qui doivent être effectués, mais aussi un caractère réactif, ces systèmes devant réagir aussi à des événements discrets. Les contraintes temporelles sont essentiellement des contraintes de performances, de type débit de traitement.

La première étape de la description du système passe par la description de l'application par un langage de type impératif [Magnaud 1990] à trois niveaux de hiérarchie. Le premier niveau décrit l'aspect réactif, c'est à dire tout ce qui définit l'interface du système avec son environnement. C'est à cette étape que sont décrites les contraintes temporelles portant sur les fréquences d'acquisition et de sortie des signaux.

Le second niveau permet de décrire la partie transformationnelle, cette partie d'exprimer le parallélisme fonctionnel de l'application sous formes de réseaux de modules [Delfieu 1990], qui permettent de définir les parties parallèles et séquentielles. Le dernier niveau décrit de manière séquentielle le corps des modules.

La deuxième étape est le processus de compilation. Il a pour objectif de déterminer le parallélisme minimal, compte tenu des contraintes temporelles exprimées à l'étape précédente, et des performances connues de l'architecture cible. Cette étape va calculer, en fonction des débits d'entrée et de sortie, les temps alloués au niveau de chaque module du réseau. Lorsqu'il s'avère que le temps de traitement alloué à un module est incompatible avec le temps de calcul de l'architecture cible, une décomposition en un nouveau réseau de modules comportant un niveau de parallélisme supplémentaire est effectuée, le processus étant réitéré, jusqu'a l'obtention du parallélisme juste suffisant pour répondre aux contraintes temporelles.

La dernière étape a pour but d'optimiser le réseau de module issu de la compilation. A chaque module correspond un processeur, il s'agit donc de réorganiser le réseau pour réduire le nombre de processeur.

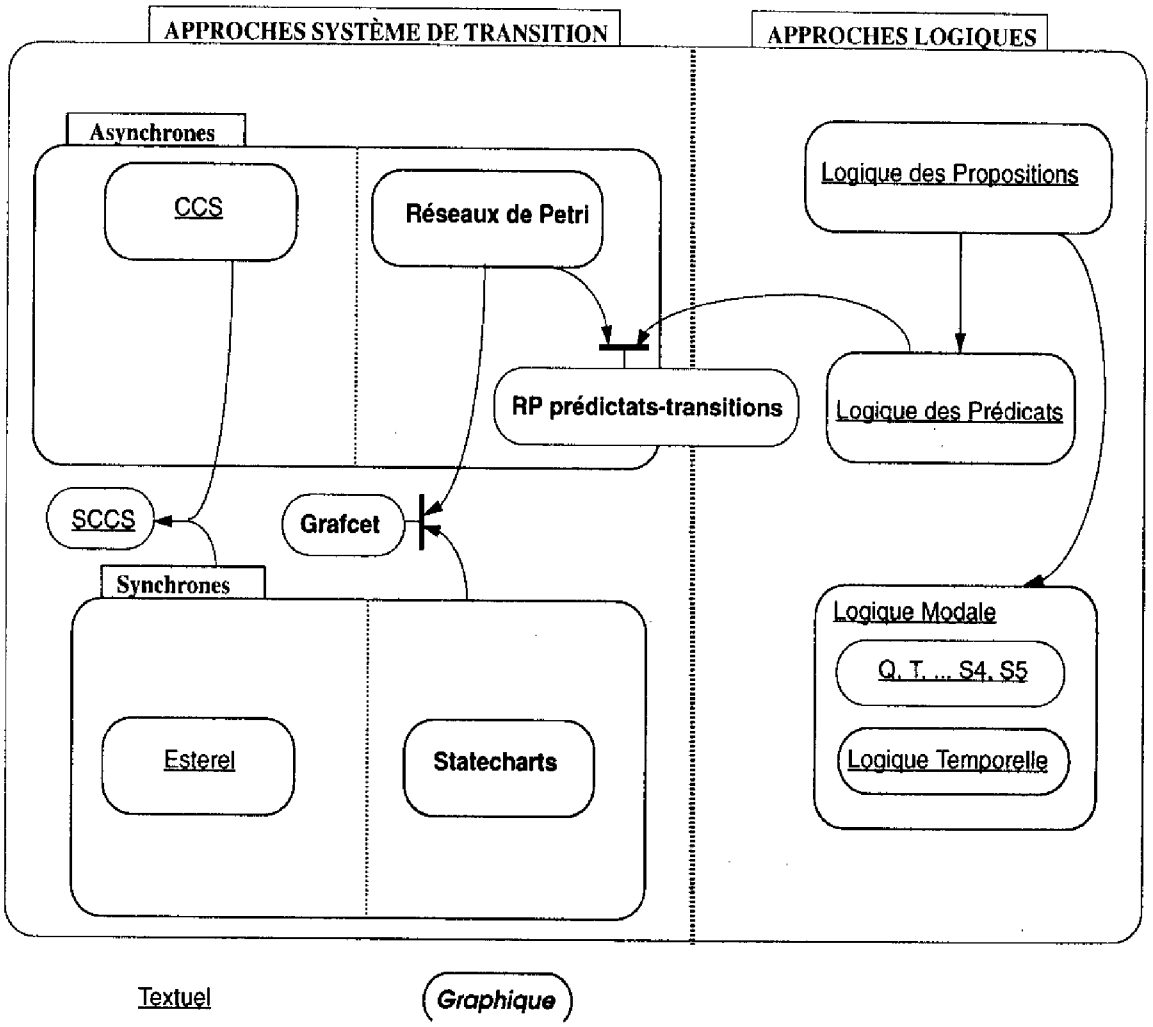


Figure III-1 : Classification des méthodes suivant trois critères, plan du chapitre

gements sur les sorties.

Si l'on affine cette description sommaire, on distingue un ensemble d'états Q reliés par un graphe "δ" dont les arcs sont des transitions étiquetées par des éléments de E. Les sorties sont générées soit sur les transitions soit dans les états. Cette dernière distinction correspond à une technologie électronique de conception des circuits. La définition formelle d'un automate fini déterministe est la suivante :

DÉFINITION 9 "AUTOMATE FINI DÉTERMINISTE (AFD)"
$AFD = (Q, \sum, \delta, q_0, F)$
où Q est un ensemble fini d'états
$\sum$ **: un alphabet d'entrée fini,**
q_0 **: l'état initial,**
$F \subseteq Q$ **: l'ensemble des états finaux,**
$\delta : Q \times \sum \rightarrow Q$, **où** $\forall q \in Q, \forall a \in \sum, \exists\, !q'/\delta(q,a) = q'$

Une catégorie plus intéressante, est celle des automates fini non déterministes (AFND), pour lesquels, une ou plusieurs transitions d'un état sont possibles pour la même entrée. Il est démontré dans [Hopcroft 1979], que pour un AFND, il existe un AFD qui reconnaît le même langage. Enfin, on peut enrichir le modèle en acceptant les transitions "ϵ", dites transitions vides, ou invisibles[3]. Là encore, [Hopcroft 1979] démontre que les AFNDs avec des transitions de type "ϵ" produisent des langages reconnaissables par un AFND donc par un AFD. Ceux sont les AFNDs qui reconnaissent les expressions régulières [4]. En ajoutant un alphabet de sortie aux AFDs on obtient les célèbres machines de Moore et de Mealy :

DÉFINITION 10 "MACHINE DE MEALY"
Une machine de Mealy est un sextuplet $M = (Q, \sum, \Delta, \delta, \lambda, q_0)$:
où $Q, \sum, q_0$ **ont le même sens que dans les définitions précédentes avec :**
- Δ **est l'alphabet de sortie**
- λ **la fonction de sortie :** $Q \rightarrow \Delta$
- $\delta : Q \times \sum \rightarrow \Delta$ [5]

La vision d'un automate comme une boîte noire amène à étudier son fonctionnement, ses entrées (et/ou ses sorties), et induit la notion d'équivalence de langage. L'automate est examiné par rapport à des séquences, qu'il est capable de reconnaître. Ces séquences sont constituées d'événements d'entrées acceptés par l'automate. L'automate de la figure III-2 accepte les séquences ab et ba.

DÉFINITION 11 "EQUIVALENCE DE LANGAGES"
On dira que deux automates sont de "langages équivalents", s'ils acceptent le même ensemble de traces.

[3]Ces transitions correspondent à des évolutions non observables du système, c'est à dire que l'automate peut "tirer" ces transitions sans qu'elles apparaissent dans le mot (trace) résultant

[4]en effet un automate reconnaissant les expressions régulières doit au moins reconnaître ϵ qui est l'expression régulière correspondant au mot vide "i" de Milner dans CCS, reprendra ce concept á travers l'événeme "i"

[5]Pour la machine de Moore, $\delta : Q \rightarrow \Delta$

[Harel 1988], Harel montre que les "graphes de haut niveau"[1] apportent à la fois de la puissance d'expression et permettent une compréhension intuitive de problèmes complexes. Certains des critères cités au chapitre II, vont nous servir de guide à l'évaluation de l'ensemble des méthodes :

- Le temps dans le modèle.
- Les données et le contrôle dans le modèle.
- La vérification formelle de propriétés.
- Le pouvoir communicatif, expressif et déductif.

L'ensemble des méthodes qui sont abordées n'est évidemment pas exhaustif, mais il contient cependant les représentants des grandes classes existantes :

les Réseaux de Petri, les Statecharts, Esterel, Lotos; en ce qui concerne l'approche logique, on abordera des théories, basées sur la logique du premier ordre, et la logique temporelle.

La figure III-1 préfigure le plan de ce chapitre, qui est découpé en deux sections, une pour l'approche système de transition (ST) et l'autre pour l'approche logique. Dans la partie ST, on examinera les méthodes suivant deux sous-sections : l'aspect langage ou graphique et le caractère synchrone ou asynchrone. Dans la partie logique, on classera les méthodes par leur pouvoir expressif.

1. Les approches système de transitions

Cette famille regroupe les descriptions par automate "basique" (machine de Mealy/Moore) ou "évolué" (les Statecharts, Esterel), les réseaux de Petri et les algèbres de processus (CCS, Lotos, ...). Avant une description plus précise de ces méthodes, on peut dégager un ensemble de caractéristiques communes à celles-ci, notamment, les notions d'état et de transition qui sont essentielles.

1.1. Notions aux cœurs des systèmes de transitions

Les notions d'état et de transition sont les fondements de toutes les approches de type "système de transitions".

- L'état représente l'ensemble des valeurs des variables du processus (ou des processus) qui le composent à un "moment donné", c'est à dire entre deux changements d'états.
- Une transition est un changement d'état, c'est à dire une modification d'une ou plusieurs variables du système. Une transition peut être due soit à un événement externe au système, on parlera alors de **stimulus**, soit à un événement interne (expiration d'un délai, terminaison d'une tâche, ...). Elle peut aussi correspondre à l'action du système sur l'environnement, que l'on appelera **réaction**.

[1] Un graphe de haut niveau est un graphe dont les arcs ne sont plus binaires : ils peuvent mettre en relation des ensembles de sommets. Les Statecharts sont un exemple de représentation de graphe de haut niveau

fondements formels, ceux-ci restant présents au niveau de la représentation interne des données.

Choisir de représenter de manière graphique des machines à états, c'est déterminer une représentation pour les états et les transitions. Une façon d'augmenter le pouvoir expressif de ce formalisme, est d'enrichir la notion d'état (Statecharts) ou de transition (Réseaux de Petri). Ces deux moyens se rejoignent en fait, car ils introduisent tout deux un nouveau type de transition qui n'est plus binaire (d'un état à un autre) mais n-aire (d'un sous-ensemble d'états vers un autre sous-ensemble).

1.4.1. Réseaux de Petri

Les réseaux de Petri ([Brams 1982] et [Brams 1983]) sont un outil mathématique, sur lequel repose une représentation graphique simple, qui permet d'interpréter facilement les évolutions d'un modèle et de modéliser une application de façon intuitive. Une fois la modélisation faite, l'outil mathématique sous-jacent permet de corriger le modèle et permet d'en déduire les propriétés.

Utilisés principalement dans le domaine des systèmes de production et notamment dans les cellules flexibles de fabrication ([Valette *et al.* 1982], [Valette 1990]), on les retrouve aussi dans le prototypage d'applications ([Sibertin-Blanc 1988]) où ils s'avèrent particulièrement bien adaptés pour la simulation interactive.

Les réseaux de Petri sont basés sur la théorie des automates (notamment sur la notion d'automate fini), pour lesquels on a voulu ajouter la notion de communication et de synchronisation. Les Réseaux de Petri ne sont pas qu'une simple extension des automates. Le nouveau pouvoir d'expression qu'ils introduisent est accompagné de nouveaux outils d'analyse, qui permettent de vérifier des propriétés portant à la fois sur l'aspect dynamique et statique du réseau.

Les principes de base

- Un Réseau de Petri peut être défini par le quadruplet :
  $RP =< P, T, Pre, Post >$ où :
  - P : est un ensemble de places qui correspondent à un ensemble d'états.
  - T : est l'ensemble des transitions, qui à la différence des labels de transitions des automates, identifient une transition unique.
  - Pour modéliser les transitions, on utilise deux fonctions : $Pre : P \times T \rightarrow N$ et $Post : P \times T \rightarrow N$ qui définissent les matrices correspondant respectivement aux matrices de pré-conditions et de post-conditions.
- **La synchronisation**, ou communication comme on vient de l'évoquer, est l'un des concepts de base des RP. Mathématiquement, elle est matérialisée par les matrices *Pre* et *Post*, ou pour une transition donnée (une ligne de la matrice), plusieurs jetons peuvent être nécessaires dans plusieurs places de départ. Dans le schéma suivant (III-4), la transition t, représente la synchronisation de deux processus $P1$ et $P2$.
  Ce concept existe dans les automates, mais les RP le mettent en valeur. Considérons le RP de la figure III-5 modélisant deux automates A_1 et A_2, se synchronisant dans les états "0" et "1" sur une transition c.
  Si l'on considère un automate équivalent, modélisant le même comportement, il s'en suit une représentation de "type entrelaçage" notamment pour modéliser la dernière étape qui représente l'arrivée dans l'état "01".

1.2. Notions liées

- Certains états sont particularisés, comme **l'état initial** qui représente l'état du processus (valeurs initiales des variables) avant la mise en route du système. **L'état final** est un état stable, atteint par le système, qui désigne la fin d'exécution attendue du système, tandis que **les états puits** représentent la fin malheureuse d'un programme, et sont des états pour lesquels aucune transition vers un autre état n'est possible. Ils résultent souvent d'un interblocage.
- Le parallélisme peut se concevoir de deux façons. Le parallélisme "vrai", qui rejoint l'idée du sens courant du parallélisme, c'est à dire lorsqu'il peut exister, par exemple, dans une chaîne de production, différents postes de fabrications concourant simultanément à la réalisation de pièces rentrant dans la composition d'un même produit. C'est un parallélisme de tâches, qui est de type "gros grains" (en référence à la terminologie déjà employée au chapitre précédent).
  Une autre conception est celle du parallélisme par entrelaçage. Celui-ci considère une tâche comme une succession d'événements discrets et atomiques. Cette notion rejette la possibilité de simultanéité d'événements, en vertu d'une conception continue du temps[2]. Il s'agit d'un parallélisme "virtuel", représenté par un entrelacement d'actions atomiques. Comme on l'a déjà évoqué précédemment, il y a trois types d'événements : les stimuli, les événements internes et les réactions. Ils peuvent être classés en deux catégories, ceux qui sont observables, du point de vue de l'environnement et ceux qui ne le sont pas.
- La synchronisation est une notion indissociable de celle du parallélisme. Des processus qui concourent à un même objectif ont forcément besoin de communiquer. La synchronisation est bien présente dans toutes les méthodes, à un niveau plus ou moins évolué. Le niveau automate de base étant celui où, bien qu'expressible (nous le montrerons dans la partie consacrée aux Statecharts), la notion de synchronisation est complètement illisible pour celui qui n'a pas construit l'automate.
- Le non déterminisme est présent dans les approches ST. Il permet de modéliser les évolutions imprévisibles de l'environnement. Il peut toutefois résulter d'une mauvaise interprétation des caractéristiques du modèle, et pour cela on désirera le détecter. On représente généralement cet indéterminisme par deux transitions concurrentes portant des étiquettes identiques.

1.3. Fondements Théoriques des Systèmes de transitions

Toutes les approches dites ST sont basées sur la théorie des automates. Les automates sont des objets mathématiques dont l'étude permet de donner un support théorique à de nombreux développements liés à l'informatique : l'étude des circuits séquentiels, les algorithmes, ...

1.3.1. Un automate

Un automate ([Hopcroft 1979]) peut être vu comme une boîte noire, comportant un ensemble d'entrées E et un ensemble de sorties S. Des changements sur les entrées provoquent des chan-

[2] voir la définition du premier chapitre

d'observer les deux transitions. On pourra, par exemple, se trouver de façon non déterministe dans l'état "1" sans pouvoir être sensible à la transition c.
Dans le modèle mathématique des RP, c'est à dire dans les matrices *Pre* et *Post* apparaît la notion de conflit structurel, c'est à dire que deux transitions peuvent avoir les mêmes préconditions. Cependant cette notion ne coïncide pas avec celle de non déterminisme, car ces transitions apparaissent de manière distinctes dans les matrices (elles correspondent bien à deux colonnes ou deux lignes différentes). Le non déterminisme est en fait lié à la notion de conflit effectif et donc à l'interprétation du réseau de Petri, il apparaît de manière sur-ajoutée à la méthode, grâce à une fonction d'étiquetage qui peut associer à deux transitions la même étiquette. La notion de non déterminisme est au cœur du modèle automate, tandis que dans les RP elle y est rapportée.

- **Le parallélisme**
  Le point précédent introduisait la notion de synchronisation qui permettait de faire communiquer des entités indépendantes. Cette notion est corrélée à celle de parallélisme. C'est bien un concept qui singularise les RP par rapport aux automates. En effet, les RP présentent deux niveaux de parallélisme (figure III-7) :
  Le parallélisme de type "vrai" qui considère que deux transitions parallèles ne peuvent pas être ordonnées. Ceci est du au fait que l'on peut remplacer toute transition par une séquence "transition-place-transition". Ce parallélisme peut être identifié à un parallélisme "gros grain" déjà évoqué au chapitre II, il s'agit d'un parallélisme de tâches. Effectivement, ce pouvoir de substitution permet de décomposer toute transition en une séquence finie de transitions, comme on peut le faire lorsque l'on affine la description d'une tâche laissée initialement indéfinie. Cette notion de parallélisme est absente dans les automates.
  Le parallélisme de type entrelaçage, est le parallélisme que l'on retrouve dans toutes les approches systèmes de transition. Cette notion de parallélisme interprète deux processus en parallèle comme l'entrelacement de leurs actions. Ces actions sont considérées de manière atomique.

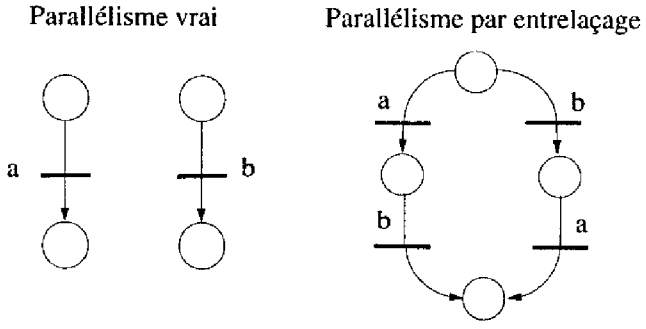


Figure III-7 : Parallélisme vrai et d'entrelaçage

Données et contrôle

- **Le contrôle**
  Les automates ont une vue discrète des événements et des actions qu'ils voient comme des séquences d'événements discrets. Le parallélisme qu'ils considèrent est de type entrelaçage,

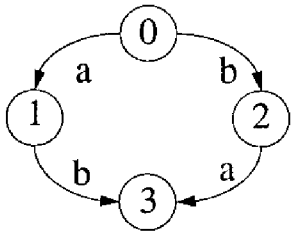


Figure III-2 : Traces acceptées par un automate

Dans la figure III-3 les deux automates acceptent les mêmes traces et sont donc "langages équivalents".

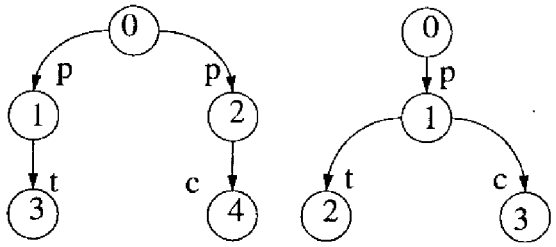


Figure III-3 : Automates équivalent langage mais non équivalent observationnellement

1.3.2. La notion d'observation

Elle permet d'introduire une nouvelle notion d'équivalence, au sens où l'on va pouvoir distinguer deux automates qui sont langages équivalents. Par exemple, imaginons que les deux automates de la figure III-3 modélisent un distributeur de boisson, où l'événement "p", correspond à l'action de mettre une pièce dans le distributeur, "c", à celui de choisir du café, "t", à celui de choisir du thé.

Dans l'automate de la partie gauche, après avoir introduit sa pièce, l'utilisateur n'aura plus le choix de la boisson, puisque l'automate ne sera sensible qu'à une seule des deux requêtes. Dans l'automate de la partie droite après avoir introduit sa pièce, les deux choix seront encore possibles. Ce simple constat a permis à Milner d'introduire une grande famille de méthode de spécification : CCS[6]

1.4. Les approches graphiques

La spécification est la dernière étape où peut intervenir le client. Il est donc intéressant de pouvoir communiquer de façon claire avec lui. Certaines méthodes de spécification ont donc choisi le parti de cette communicabilité, en proposant une traduction graphique et intuitive des

[6]a Calculus Communicating System

1. *K-Borné*, indique que le marquage ne va pas croître de manière infinie. Cela se traduit par le fait qu'il existe un automate fini équivalent au réseau : le graphe des marquages.
2. *Quasi-vivant* : indique que toutes les transitions seront tirables à partir du marquage initial. Cela implique que si l'automate équivalent existe, alors il est connexe.
3. *Vivant* : indique que toutes les transitions seront tirables à partir de tout marquage atteint depuis le marquage initial. Cela implique que si l'automate équivalent existe, alors il est connexe et il contient une composante fortement connexe où toutes les transitions sont tirables.
4. *Réinitialisable* : indique que le réseau peut se retrouver dans son état initial, ce qui implique que le graphe des marquages soit fortement connexe.

- **Les propriétés structurelles**
  Elles sont indépendantes du marquage initial. Elles permettent de prouver que des places sont bornées. Ces propriétés ne trouvent leur intérêt, que dans l'interprétation que l'on peut faire de l'outil réseau de Petri; on ne trouve pas leur équivalent dans les automates. On peut par exemple, si l'on assimile une place à une unité de stockage, montrer que la capacité de cette unité ne sera pas dépassée, ou bien si l'on considère un ensemble de places comme une section critique, prouver des propriétés d'exclusion mutuelle.

Pouvoir communicatif et expressif

- **Le pouvoir d'expression**

Les RP ont un pouvoir d'expression supérieur à celui d'un automate, comme en témoigne la figure III-8. Dans cet exemple, le réseau reconnaît le langage non régulier

$$L = \{(^n\ )^n, n \in N\}$$

qui est l'ensemble des mots comportant autant de "(" que de ")". Les RP non bornés ont effectivement la puissance d'expression des automates à piles.

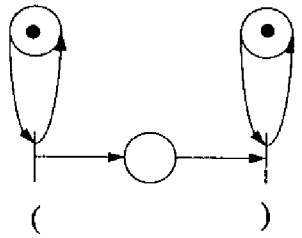


Figure III-8 : réseau reconnaissant des expressions composées de parenthèses bien formées

- **Le pouvoir communicatif**
  Ce pouvoir s'exprime grâce aux qualités de la représentation graphique du modèle mathématique qui sont :
  1. une représentation idéale de la synchronisation
  2. une bonne représentation du parallélisme

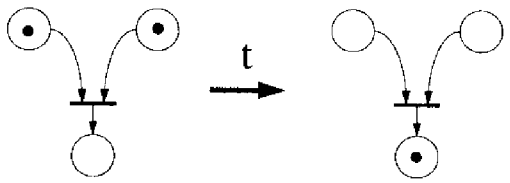


Figure III-4 : Représentation de la synchronisation

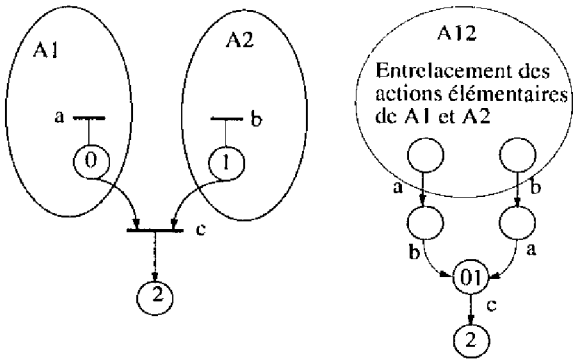


Figure III-5 : Synchronisation, un concept caché dans les automates

Cette comparaison montre clairement que ce concept, bien qu'expressible par automate, reste obscur, tandis que dans les RP, il apparaît de façon claire et lisible. En outre il permet de représenter un problème par sous-entités communicantes ce qui offre la possibilité d'une analyse compositionelle des problèmes.

- **Le non déterminisme**
  A la différence des automates, où le déterminisme est présent par les qualités que l'on attribue à la fonction de transition "*delta*", le non déterminisme, n'est pas un concept se trouvant au cœur de la méthode.

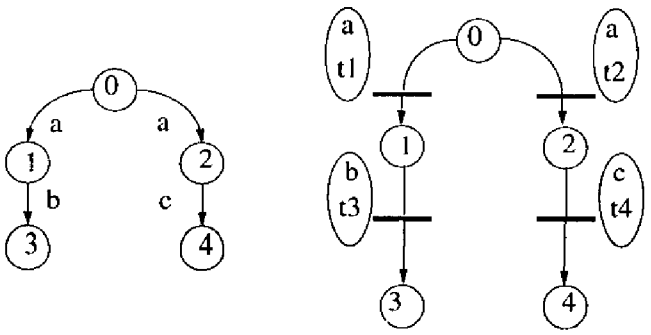


Figure III-6 : Non déterminisme

Dans la partie gauche de la figure III-6, après une transition "a", l'automate peut (comme nous l'avons déjà illustré précédemment avec la machine à café) ne plus être capable

puisse être tirée, et un temps de garde maximum, avant lequel la transition doit être franchie. Ils permettent de décrire les chiens de garde, car cet intervalle défini une restriction temporelle sur un tir de transition, plutôt qu'un gel de jeton.

Le temps intervient aussi dans ce modèle de façon significative puisqu'il va en changer l'interprétation. En effet, intervient la notion de "devoir". Un intervalle lié à une transition, force la le tir de la transition à une date inconnue dans cet intervalle. Non seulement le réseau agit de manière autonome, mais en plus il réagit de manière imprévisible. Cela constitue une imprécision qui peut s'accumuler dans le réseau. En outre elle augmente de manière exponentielle le nombre de marquages possibles.

Dans les modèles précédents, le temps intervient comme une horloge globale. Tout réseau, ou partie de réseau, a besoin à un moment donné, soit de temporisation soit de chien de garde. Il consulte alors une horloge qui lui délivre des dates.

En conclusion on peut relever les problèmes relatifs aux extensions temporelles des RP suivants :

- l'introduction du temps change l'interprétation du modèle,
- l'introduction du temps introduit de l'imprécision (tir non déterminisme à l'interieur d'un intervalle de temps)
- l'introduction du temps nécessite de considérer l'appel à une horloge globale qui renseigne chaque processus,
- les contraintes temporelles ne peuvent pas apparaître de manière multiforme[7].

1.4.2. Traduction de réseaux de Petri en séquences temporisées

Dans la figure III-10, on montre que l'on peut construire un automate générant des séquences temporisées selon le format que l'on a défini à partir d'un réseau de Petri temporisé.

Dans le réseau III-10.a, les places sont temporisées. Pour matérialiser cet écoulement du temps dans l'automate équivalent, on introduit un événement spécifique, "h". On considère que seul cet événement marque le temps. L'automate équivalent de la figure III-10.b, s'obtient à partir du graphe des marquages du RP. On nomme les états par le marquage dans lequel se trouve le réseau. Par exemple, l'état "1000", indique que seule la place p_0 est marquée. On note certains états avec des étoiles pour indiquer que le jeton est indisponible. Ainsi, "0*00" signifie que le jeton se trouve dans la place p_1, mais indisponible jusqu'a l'occurrence de h marquant la fin d'une temporisation d'une unité de temps. Ainsi, le réseau de la figure III-10 trouve son équivalent dans l'automate de la figure III-10.b (où h^3 de la figure représente 3 occurrences de h).

Pour les réseaux de Petri temporels, ce sont des intervalles qui sont associés aux transitions. La borne inférieure représente un délai pendant lequel le jeton reste disponible dans la place antérieure. Par exemple, dans la figure III-11.a un réseau modélisant le concept du timeout est représenté. Dans ce réseau, deux transitions sont en conflits, fin_1 et fin_2. fin_2 représente l'arrivée à temps d'une condition (arrivée du jeton dans la place p_4 : condition). L'arrivée

[7]Une contrainte temporelle s'exprime de manière multiforme si elle peut s'exprimer en terme d'événements. On peut prendre en exemple comme référence une séquence d'événements qui sert de base de temps et les contraintes s'expriment alors en terme de cet événement

ce qui entraîne un controle de type binaire, puisque celui-ci ne se trouve que dans un seul état à la fois.
Dans les automates, on avait un contrôle qui passait d'un état unique à un autre état unique. Dans les RP, le contrôle acquiert de nouvelles propriétés :

- Le contrôle est "quantifié" par un ensemble de jetons.
- Le contrôle est "multiple"; il peut se trouver dans plusieurs états en même temps.
- La transmission du contrôle se fait de manière "n-aire", c'est à dire par une fonction de transition t qui s'applique d'un sous-ensemble de places vers un autre sous-ensemble de places. C'est cette fonction de transition qui permet de traduire la notion de communication entre automate.

Le contrôle est donc caractérisé par un marquage de places. Il faut noter cependant, que pour une certaine catégorie de réseaux (ceux ayant un pouvoir d'expression équivalent aux automates, c'est à dire celui de pouvoir exprimer des expression régulières), cette notion de contrôle, n'est qu'un pouvoir d'abstraction nouveau par rapport aux automates, puisque l'on peut à partir de ces réseaux se ramener à un automate.
Les RP apportent donc comme on l'a déjà vu pour la notion de synchronisation, la faculté de représenter des comportements de manière plus intuitive et plus concise puisque la notion de contrôle "multiple" est plus proche de la réalité.

- **Les données**
  La séparation entre contrôle et données est parfois difficile à définir. Ainsi, par exemple, on a souvent besoin de variables dans des tests de conditions pour exprimer du contrôle, et inversement, le contrôle est toujours présent dans les traitements ne serait-ce que dans les instructions de tests ou de boucles. Cette marge de manœuvre produit une grande diversité dans les solutions possibles pour la spécification d'un même problème.
  Les Réseaux de Petri offrent une souplesse pour introduire l'aspect données dans l'expression du contrôle. Les Réseaux de Petri Colorés [Jensen 1987] permettent ainsi de représenter de manière compacte et claire des problèmes où l'aspect données introduit une combinatoire préjudiciable à la compréhension. Il faut noter cependant, que cette combinatoire persiste dans le modèle mathématique sous-jacent.
  Les Réseaux de Petri Prédicats Transitions [Genrich 1987] [Genrich 1981] participent aussi à l'objectif précédemment cité. Ces Réseaux de Petri permettent d'apporter la richesse d'expression des données que possède la logique des prédicats.

Propriétés

On peut déduire un certain nombre de propriétés d'un réseau de Petri. Ces propriétés permettent de vérifier que le réseau correspond au cahier des charges, mais aussi de l'améliorer par rapport à des considérations de type génie logiciel (modularité, robustesse, ...).

On a comparé les RP aux automates qui constituent leurs racines théoriques. On continue la comparaison, pour les propriétés, notamment par rapport au graphe des marquages, qui comme on l'a vu plus haut avec l'analyse du contrôle constitue l'automate équivalent en terme de langage au réseau de Petri. Ces propriétés peuvent être classées en deux catégories :

- **Les propriétés dépendantes du marquage initial**

dans la place p_1 d'un jeton est l'origine du temps pour les prochaines transitions temporisées. La transition fin_1 laisse 3 unités de temps pendant lesquelles le jeton est disponible en p_1, c'est à dire tirable par fin_2. Si dans l'intervalle [0,3] la condition passe à vraie alors fin_2 est immédiatement tirée. Sinon fin_1 est tirée de manière non déterministe dans l'intervalle [3,5].

L'automate équivalent (III-11.b), décrit dans sa partie droite que la transition fin2 reste possible jusqu'a l'occurrence de la troisième unités de temps, tandis que la partie gauche indique que seule fin_1 est possible dans l'intervalle [3,5[.

On peut donc à partir d'un réseau temporisé obtenir via le graphe des marquages, un automate qui reconnaît un ensemble de traces temporisés suivant le modèle que l'on a évoqué au premier chapitre.

1.4.3. Les Statecharts

La méthode des Statecharts (SC) permet une représentation concise, abstraite et modulaire d'automates complexes. Ces caractéristiques sont obtenues par le biais d'opérateurs de parallélisme et d'abstraction. Ces opérateurs permettent de cacher la combinatoire et la complexité du modèle sous-jacent. Alors que les Réseaux de Petri privilégiaient la notion de synchronisation, les SC mettent en valeur la notion d'abstraction et de composition par raffinage.

Les Statecharts sont utilisés principalement dans le domaine aéronautique pour la spécification d'organe de contrôle et d'autres applications présentant un caractère réactif ([Harel 1985]). Cependant on les voit aussi apparaitre dans d'autres domaines, comme dans les systèmes de production. Dans le cadre d'un contrat D.R.E.D.[8] nous avons évalué les SC pour la spécification des modes de marche de cellules flexibles de fabrication manufacturière ([Delfieu *et al.* 1993]). L'objet de cette étude portait sur la mise en œuvre des principes de la méthode CASPAIM [Bourey *et al.* 1989] à l'aide d'un outil synchrone.

Les principes de base

Les Statecharts enrichissent, comme le font les Réseaux de Petri, le pouvoir des automates. En effet, ils ne sont pas juste une écriture plus abstraite des machines à états. La transition d'un état à un autre (ou d'un ensemble d'état à un autre) suit des règles différentes de celles qui régissent les automates. En outre, l'hypothèse synchrone, ajoute à cette différenciation en modifiant sensiblement la notion habituelle d'état comme on va le voir dans cette présentation où après avoir donné une définition informelle on présentera les propriétés de la méthode. On donne en annexe A, une définition formelle des SC, notamment à travers la définition de la structure interne des diagrammes, des notions de parallélisme, d'orthogonalité, et enfin de la procédure de calcul des tirs de transitions.

- **L'hypothèse synchrone**
  Cette hypothèse est une des caractéristiques principale qui distingue les SC des RP. Comme nous l'avons évoqué au chapitre I, cette hypothèse considère que le système est infiniment plus rapide que l'environnement, et donc que toute réponse à un stimulus est toujours générée dans le même temps que le stimulus. Ainsi la longue séquence de communication interne peut être observée au même instant :

[8] Proposition DRED-MENJS, groupe Automatique, pôle Système à Événements Discrets, "Langages synchrones et Réseaux de Petri"

On relève par contre, une mauvaise représentation du concept d'interruption et de reprise comme en témoigne le schéma III-9. Ces concepts, bien que représentables, engendrent une complexité préjudiciable à la clarté, notamment pour le concept de reprise qui oblige comme on peut le constater à doubler la taille du réseau.

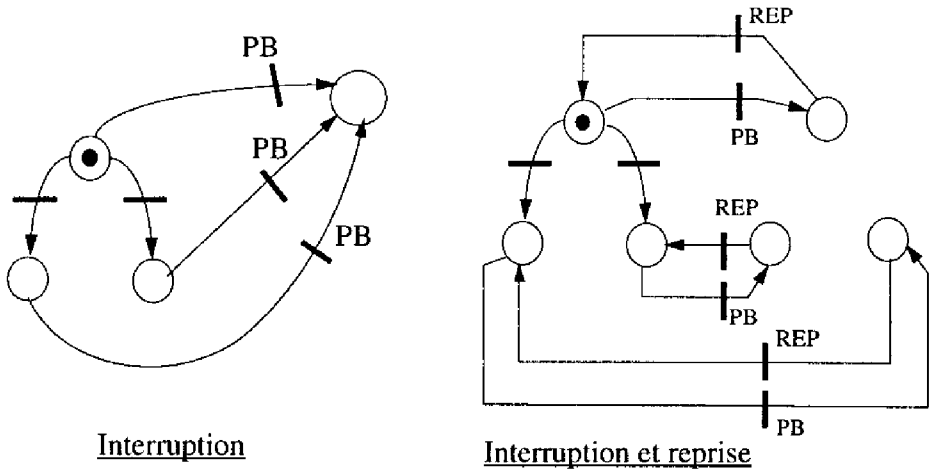


Figure III-9 : Représentation du concept d'interruption et de reprise à l'aide des RP

Le temps

Le temps dans les RP est apparu comme une extension du formalisme. Comme dans d'autres paradigmes (algèbres de processus, logiques, ...), le temps est introduit sous deux aspects : des durées ou des intervalles. Les RP présentant la dualité (au point de vue mathématique) place-transition, celle-ci se retrouve naturellement au niveau des extensions temporelles, certaines attachant le temps aux places, d'autres aux transitions.

Dans les réseaux temporisés, une durée est associée soit à une transition, soit à une place. Dans les deux cas le jeton peut se trouver dans un état indisponible (il est soit bloqué sur une transition soit sur un état). Ce modèle ne permet pas de représenter des mécanismes simples, tel que le chien de garde, car lors de l'écoulement d'une durée, le jeton est indisponible, et donc ne peut être utilisé par des transitions concurrentes.

Dans ce modèle, le temps intervient donc pour geler certains jetons pendant une durée. La notion de date est relative au chemin parcouru par le jeton, et aux places ou aux transitions qu'il a pu rencontrer. Ce modèle est peu adapté pour les systèmes temps réel, mais il est toutefois utilisé dans le domaine de l'analyse de performances.

Le temps change l'interprétation du modèle, car une transition pour laquelle est associée une durée nulle est tirée instantanément lorsqu'elle est sensibilisée. Le système peut de lui même, tirer en cascade une chaîne de transitions sans intervention de l'environnement et ce, jusqu'à ce qu'il arrive dans un état stable. Le temps introduit alors la notion d'état instable.

Les réseaux de Petri temporels de Merlin [Merlin 1975], sont plus généraux que les RDP temporisés : à chaque transition est associé un intervalle de temps, qui constitue, à partir de l'instant où la transition est sensibilisée, un temps de garde minimum avant que la transition ne

et t3, mais en introduisant l'indéterminisme suivant : si les transitions sont examinées dans l'ordre t1, t2, t3, on aura les tirs de t1 et t2. Par contre, si elles sont examinées dans l'ordre t3, t2, t1 on aura le tir des trois transitions.
C'est l'ordre d'examen des transitions qui détermine l'ensemble des transitions effectivement tirées. Cela se traduit au niveau de la procédure TIR, par le fait qu'une étape de calcul de l'algorithme ne peut pas remettre en cause le calcul déjà effectué; l'ensemble T des transitions tirables simultanément ne fait que grandir au fur et à mesure des itérations.

Données et contrôle

Le contrôle est matérialisé par l'ensemble des états actifs au même moment car comme on l'a déjà dit, il se transmet d'un ensemble d'états à un autre. Il y a trois primitives particulières afférentes au contrôle.

- L'entrée par défault dans un état "OU", qui désigne le sous-état dans lequel arrive le contrôle.
- L'état *historique* qui permet, lorsqu'une transition portant sur un état composite déjà atteint au paravant, est sensibilisée, de réintégrer le dernier sous-état quitté. Cette possibilité autorise la représentation du concept de reprise après une interruption.
- L'état *historique** qui permet de réintégrer l'état de plus bas niveau dans la hiérarchie de décomposition.

Si les actions qui apparaissent dans les transitions sont atomiques, les SC peuvent aussi lancer des traitements qui durent, c'est à dire qui s'exécutent en parallèle avec l'automate. Le contrôles de ces tâches (lancement, arrêt, relance, ...) peut se faire sur les transitions ou dans les états.

Comme nous l'avons vu en évoquant l'hypothèse synchrone, dans une réaction en chaîne, le contrôle ne reste pas dans les états intermédiaires. La notion d'état perd donc son sens commun, puisqu'elle peut prendre un caractère furtif. La notion d'état stable ou d'état d'attente d'une sollicitation de l'environnement se substitue à la notion d'état habituelle. On peut qu'un même état peut paraître "stable" ou "instable" selon la configuration d'entrée.

On décrit les données telles qu'elles sont définies dans l'atelier de génie logiciel Statemate ([Harel *et al.* 1990]) qui implémente la méthode Statecharts. Les données y sont alors caractérisées par leur type et leur structure. Les différents types possibles sont le type entier, réel, chaîne de caractères, tableau de bits, et de type enregistrement, ... Les opérations sur ces types sont celles que l'on rencontre habituellement dans les langages de programmation. Les SC offrent donc une richesse intéressante de manipulation des données comparable à celle que l'on trouve dans les langages de programmation. Ces données apparaissent sur les transitions, notamment dans les parties *condition* et *action*.

Exemple :
L'étiquette ([I1 OR I2) = 0B1101] signifie : si le résultat du "ou" logique entre les expressions logiques $I1$ et $I2$ est égal au nombre 1101 exprimé en binaire, alors la condition est vraie.
Il faut noter que les données sont partagées entre les composants parallèles, ce qui peut engendrer des problèmes d'accès simultanés aux variables ("racing conditions").

Propriétés

L'atteignabilité, les conditions d'interblocages, le non déterminisme, les parties mortes des SC et les conditions type "racing" [9] sont des situations prévisibles (de manière statique) par les SC (implémenté dans Statemate).

L'atteignabilité permet de tester s'il existe une configuration dans laquelle la condition spécifiée est vraie. En particulier on peut vérifier pour un état s, l'atteignabilité de la condition $in(s)$. La détection de conditions qui sont toujours vraies, (quelque soit l'état où est effectuée cette recherche) permet de détecter l'interblocage. La recherche de l'indéterminisme se fait en recherchant toutes les situations pour lesquelles existent à un instant donné plusieurs transitions possibles. Les conditions type racing sont détectées de la même façon que l'est le non déterminisme. Une autre propriété intéressante est celle de pouvoir détecter les parties "mortes" du réseau, c'est à dire les états impossibles à atteindre ou les transitions impossibles à tirer.

Pouvoir communicatif et expressif

Le pouvoir communicatif est obtenu grâce à l'implémentation des concepts suivants :

- Simplicité de l'interprétation des machines à états
- Notion d'abstraction
- Notion de parallélisme
- Notion de reprise
- Décomposition hiérarchisée de l'aspect contrôle et transformationnel

L'aspect transformationnel est présent à travers la notion d'ActivityCharts (entités représentant des fonctionnalités de calcul), d'adopter une technique de raffinage successif. Toute fonctionnalité est analysée en décrivant son contrôle et ses sous-activités chacune pouvant être à son tour raffinée.

Le Pouvoir expressif des SC (sans utilisation de variables) est limité comme celui des automates, à celui des expressions régulières. C'est à dire qu'il ne peut exprimer des comportements où figure la notion de mémoire et encore moins de celle calculabilité. Cependant les variables permettent facilement et simplement de combler cette lacune.

Dans la figure III-13.a on exprime la reconnaissance du langage $L = \{a^n b^n / n \in N\}$ (équivalent à l'exemple des expressions bien parenthésées, présenté dans la partie Réseau de Petri). Cette reconnaissance passe par l'utilisation d'une variable, pour compter les occurrences de a, avant celles de b. De même on peut reconnaître le langage $L = \{a^i / i = 2^k, k \in N\}$ (figure III-13.b) où apparaît la notion de calculabilité (les puissances de i devant être des puissances de 2).

Les problèmes

Un certain nombre de problèmes peuvent survenir lors de l'utilisation de la méthode des SC. Ceux-ci sont dus notamment à l'utilisation de la négation d'événement, de variables partagées, ou de la mauvaise interprétation de l'hypothèse synchrone.

[9] Il s'agit de conditions d'accès simultanés multiples à une variable

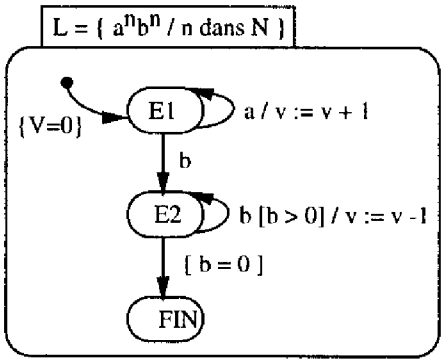


Figure: III-13.a

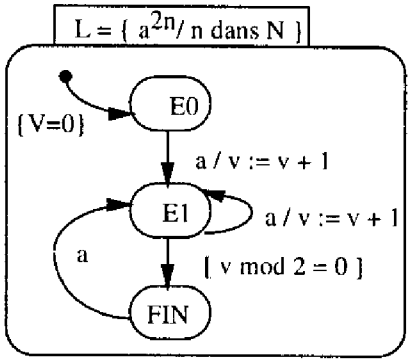


Figure: III-13.b

Figure III-13 : Reconnaissance de langages

Ces problèmes conduisent en général à des comportements non déterministes. On peut classer ces problèmes en trois catégories : ceux concernant les transitions en conflit, ceux ayant trait à la violation de causalité, et ceux dus à l'utilisation de variables partagées.

Problèmes d'utilisation des Statecharts

Dans les SC, on est confronté à un ensemble de situations, donnant des résultats indéterministes ou non conformes aux prévisions attendues [Antoine 1994]. Ces problèmes peuvent être dus à une mauvaise spécification, mais aussi à une mauvaise interprétation de la sémantique de ce formalisme. La plupart de ces problèmes sont dus [Huising 1991] à la combinaison de plusieurs facteurs :

- L'hypothèse synchrone
  Cette hypothèse induit comme on l'a déjà évoqué des chaînes de transitions. A l'intérieur de ces ensembles, on trouve des transitions parallèles et/ou des séquences de transitions.

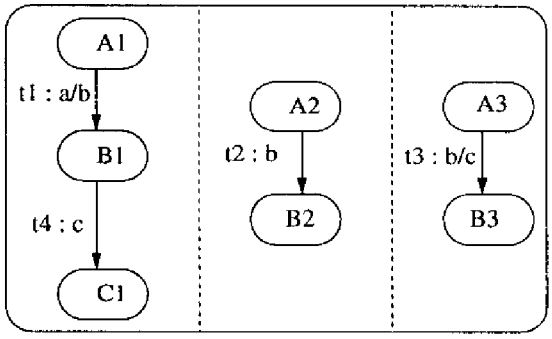


Figure III-14 : Exemple de séquence de transitions

Dans la figure III-14, considérons que le système se trouve dans l'état A_1,A_2 et A_3. Lorsque a est présent, le contrôle passe de l'état A_1 vers l'état B_1. L'événement interne b est alors

généré, ce qui provoque le passage de l'état A_2 vers l'état B_2 et de l'état A_3 vers l'état B_3. c est généré à ce moment, ce qui provoque enfin le passage de l'état B_1 vers l'état C_1. On a donc la séquence $\{t_1,t_2 \parallel t_3,t_4\}$ qui indique que la transition t_1 est suivie de t_2, en parallèle avec t_3 puis suivie de t_4.
L'hypothèse synchrone impose de considérer deux échelles de temps : Tout d'abord celle des "macro-étapes", qui sont des ensembles de transitions "synchrones" qui ont donc la même date, Et ensuite celle au niveau des "micro-étapes", qui sont les transitions constituant une macro-étape.

- La causalité
  Une micro-étape se définit aussi par le fait qu'elle est un constituant d'une chaîne de causalité comme la séquence $t_1 \rightarrow (t_2 \parallel t_3) \rightarrow t_4$ de l'exemple précédent. Une causalité introduit une notion de précédence temporelle. La notion de causalité (implication logique) rentre en "conflit" avec le synchronisme des transitions.
- La négation d'événement
  La négation d'événement, qui apparaît souvent dans la littérature comme un événement, est en fait une condition.C'est en effet un phénomène non furtif, qui prend la valeur vraie dès l'instant où une transition portant cette étiquette est évaluée, si l'événement n'est pas présent.

La combinaison de ces concepts pose des problèmes de consistance globale au niveau des macro-états. Dans l'exemple suivant (fig. III-15), supposons que a et b ne sont pas générés au moment de l'examen du SC. t_1 peut alors être tirée, générant l'événement b, ce qui cause le tir de t_2 qui génère à son tour a. De ce fait, t_1 n'est plus sensibilisable et n'est pas tirée; mais alors t_2 n'est pas tirée, a n'est pas généré et t_1 est donc tirable ...

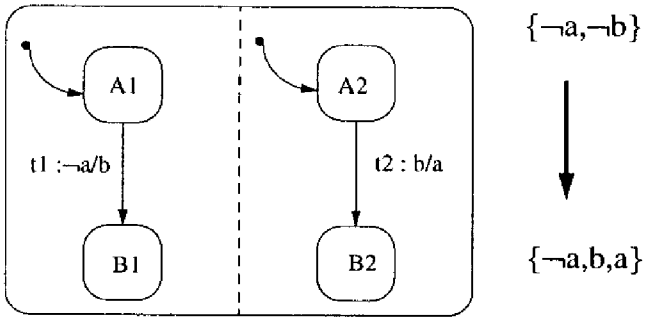


Figure III-15 : Inconsistance globale d'une macro-étape

Ce problème peut être résolu si l'on considère qu'une macro-étape peut être globalement consistante. Une micro-étape apparaissant dans une chaîne de causalité ne peut remettre en cause une micro-étape précédente. Cela se traduit au niveau de la syntaxe formelle, au niveau la fonction TIR précédemment évoquée, qui est construite de manière à être croissante monotone.

Dans l'exemple ci-dessus, la chaîne $\neg a \Rightarrow b \Rightarrow a$, n'a aucun sens au niveau logique, sauf si l'on y rattache une notion temporelle, où chaque micro étape, bien qu'ayant toutes la même date, sont ordonnées dans une séquence. Ce sont ces deux échelles de temps, qui nous permettent d'expliquer cette inconsistance globale.

De façon pratique, on peut interpréter ces échelles de temps de la façon suivante : dans une macro-étape, on place un ensemble de transitions qui sont des conséquences les unes des autres et qui ont toutes la même date.Cependant, elles ont été exécutées de manière séquentielle mais dans un délai négligeable par rapport à la dynamique du procédé. Leur datation les unes par rapport aux autres n'apportent aucune information pertinente pour le problème.

Classification des problèmes

Une analyse plus fine de ces problèmes révèle qu'ils sont dus à une mauvaise interprétation, soit de la notion d'automate communicant, soit de la notion synchrone.

Les conflits que l'on retrouve au niveau des automates, sont essentiellement des problèmes de conflit de transitions et des problèmes d'accès simultanés à des variables. Ces problèmes sont généralement dus à la non prévision de configurations de l'environnement, qui engendrent des comportements imprévus.

- conflits de transitions

  Deux transitions sont concurrentes si partant d'un même état, et ayant pour destination deux états différents, elles portent la même étiquette. On distingue alors, deux cas de figure : les états d'arrivée sont de même niveau (partie gauche fig. III-16), on a alors un cas d'indéterminisme pur; ou bien, un des états d'arrivée se trouve à un niveau différent de l'autre (partie droite fig. III-16), et on peut privilégier alors cette dernière transition (comme le fait Statemate). Dans ce dernier cas la transition la plus interne est donc inutile.

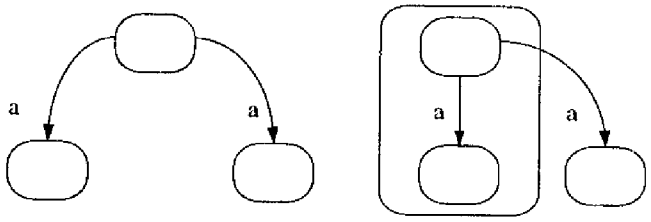


Figure III-16 : Transitions concurrentes

Un autre cas de figure (fig. III-17) est possible lorsque la configuration du système autorise le franchissement de deux transitions concurrentes. Dans cet exemple on suppose que les événements a et b sont présents, que les conditions sont vraies, et que le contrôle du système est dans l'état E_1. On se retrouve alors dans le premier cas cité. Ce dernier cas de figure peut se produire lorsque l'utilisateur n'a qu'une connaissance partielle du comportement de l'environnement et qu'il ne peut prévoir toutes les combinaisons d'entrées possibles. On peut cependant éviter cet indéterminisme, comme dans l'exemple précédent, en donnant la priorité à t_1 : en remplaçant t_2 par $b \wedge \neg a[y > 0]$. Avec cette étiquette ces deux transitions sont incompatibles.

- Accès à des variables partagées
  Les problèmes pouvant intervenir sont de deux types : l'accès à une même variable, qui

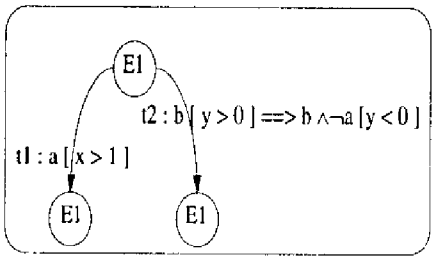


Figure III-17 : Transitions indépendantes

est un problème classique que l'on retrouve dans tout formalisme utilisant la notion de parallélisme.

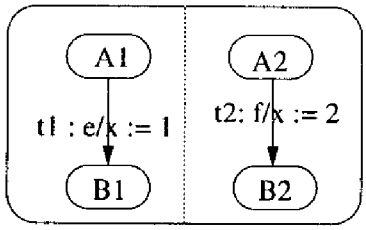


Figure III-18 : Accès à une variable partagée

L'exemple III-18 illustre ce cas. On peut résoudre ce problème, comme dans le cas précédent en donnant des priorités sur les transitions pour éviter ces accès simultanés.
Un autre cas est présenté dans la figure III-19. tr(x=y) est un événement (produit uniquement sur un front montant), indiquant le passage de la valeur logique faux à vraie (cet événement est l'équivalent d'un front montant : si l'expression reste vraie, on ne pourra pas observer cet événement). Dans cet exemple, on considère que le contrôle du système est initialement dans les états E_0, E_1 et E_2 et que x et y sont égaux. Le problème se pose lorsque l'événement e se produit, selon l'interprétation que l'on fait de la mise à jour de la variable x.

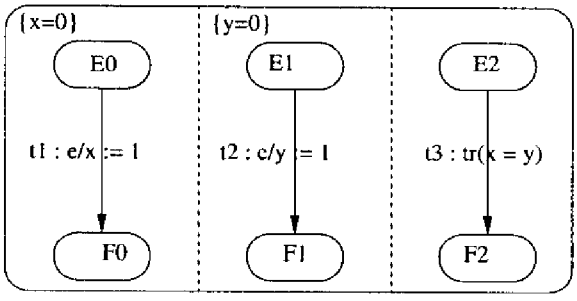


Figure III-19 : Mise à jour et synchronicité

Il existe un non déterminisme selon que la mise à jour de x et y est faite en même temps ou non. La figure III-20 illustre ces deux cas. Dans le cas d'une mise à jour simultanée,

l'expression logique $[x = y]$ reste à vraie. Il n'y a donc pas de front montant qui déclencherait t_3. Dans le second cas, ce front existe car $x = y$ passe brièvement à faux pendant la mise à jour non simultanée de x et y.

Les problèmes que nous venons d'évoquer sont dus à la combinaison de concepts qui enrichissent considérablement le pouvoir d'expression de la méthode. Ils peuvent cependant être évités, comme on l'a montré. En outre le logiciel qui implémente cette méthode s'appuie sur une syntaxe formelle qui lui permet de détecter statiquement un grand nombre de ces problèmes. Dans les SC une grande liberté de choix est laissée à l'utilisateur, liberté qui se paie par les problèmes évoqués ci-dessus.

Remarque 1 : Esterel, l'équivalent langage de Statecharts, a résolu le problème en rejetant d'avance, tout programme non déterministe, ou bien violant le principe de causalité.

1.4.4. Traduction de Statecharts en séquences temporisées

Les Statecharts sont des représentations évoluées d'automates. On peut donc comme on l'a fait pour les RP, générer des traces temporisées à l'aide d'événement jouant le role d'un marqueur temporel.

Dans l'exemple III-21, une contrainte temporelle de type "chien de garde" est posée sur un événement b. On se ramène à notre format en réécrivant l'instruction tin((in(1),2) par h^2. L'origine de temps implicite $en(1)$, [10] se retrouve dans l'expression ah^2c par l'occurrence de a.

Il faut noter toutefois, une perte de précision de cette notation par rapport aux SC. Dans les SC, lors de l'entrée dans l'état E_1, un timer est déclenché et exactement après 2 unités de temps, le contrôle va passer en E_2. L'expression correspondante, en terme de séquence temporisée, encadre l'occurrence de b entre 2 et 3 unités de temps.

1.4.5. Comparaison et exemples traités par les deux méthodes

Les deux formalismes que nous venons de voir sont basés sur des automates communicants, dont le pouvoir d'expression a été augmenté en généralisant la notion de transition, qui passe de binaire à n-aire. Les RP utilisent ce concept pour concrétiser la notion de synchronisation; les SC pour la hiérarchie et le parallélisme. Le parallélisme dans les SC est de type entrelaçage, tandis que celui des RP correspond à l'idée du sens commun, c'est à dire celui où l'on ne peut pas ordonner les transitions. La liberté offerte par les SC permet de spécifier rapidement, de façon très intuitive des applications complexes. Son pouvoir d'abstraction permet une approche descendante. Son opérateur de parallélisme lui donne un pouvoir expressif important tout en gardant un code déterministe. A titre de conclusion nous donnons par la suite des exemples plus favorable aux RP et d'autres plus favorables aux Statecharts.

L'exemple III-22 modélise l'usinage d'une pièce et prend en compte, l'arrivée possible de pannes durant ce traitement, ainsi que la réparation qui permet de réintégrer le dernier état quitté.

[10] soit e le label d'un état : en(e) est un événement généré automatiquement par le système, lors de l'entrée dans l'état "e"

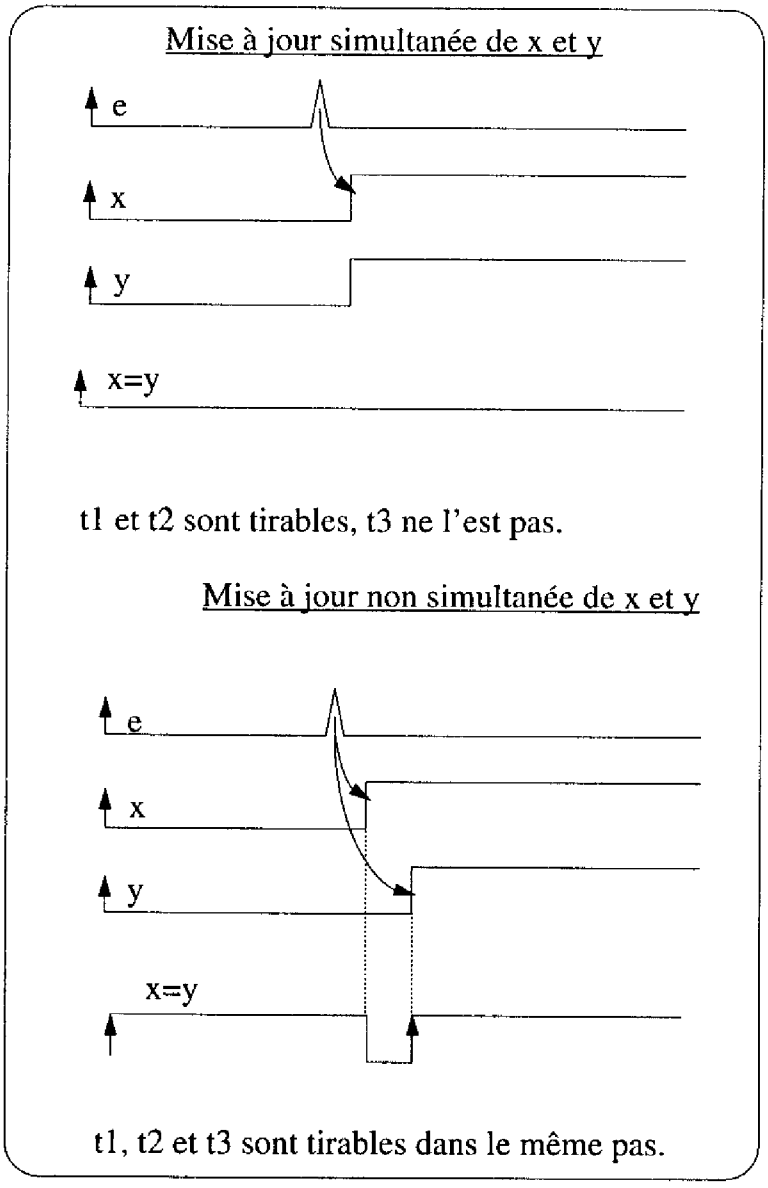


Figure III-20 : Non déterminisme sur la mise à jour

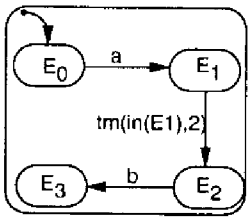


Figure III-21 : Manipulation du temps dans les SC

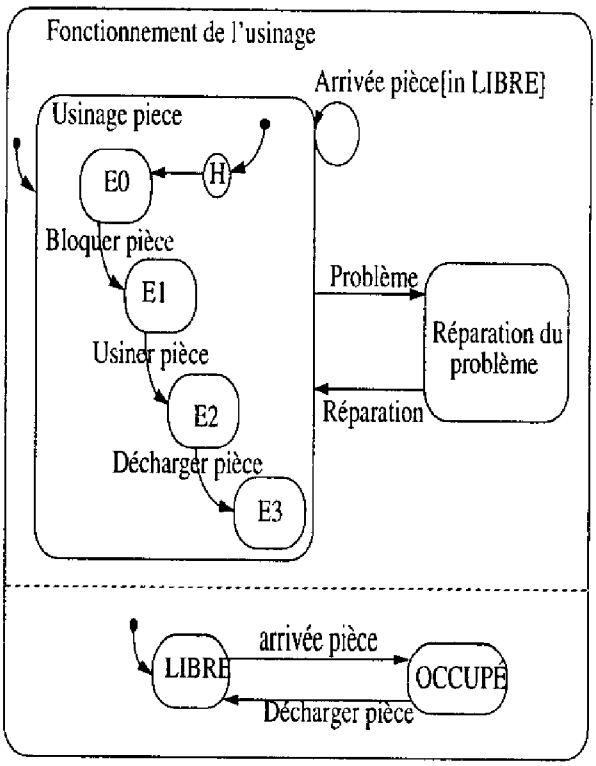


Figure III-22 : Exemple favorable aux Statecharts

Remarque 2 : Dans cet exemple, des pièces arrivées dans l'état OCCUPE, ne sont pas prises en compte.

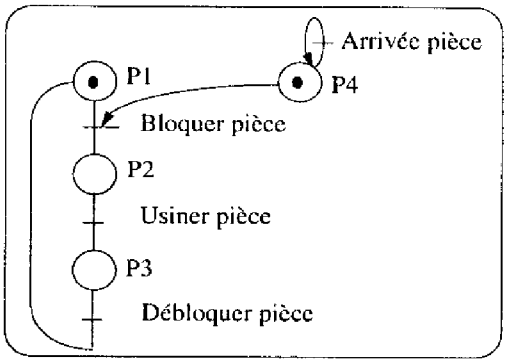


Figure III-23 : Exemple favorable aux Réseaux de Petri

L'exemple favorable aux RP (III-22) modélise aussi un usinage de pièces. Pendant cet usinage, des pièces peuvent arriver et elles s'accumulent alors dans la place p_4. Ce formalisme permet d'éviter d'utiliser une variable pour compter les pièces arrivées, comme on serait sûrement obligé de le faire dans un modèle de type Statecharts. Dans le modèle SC, on aurait de plus été obligé de compter ces arrivées dans un module en parallèle. Par contre, s'il fallait modéliser des états de pannes et de reprise, cela compliquerait le modèle en rajoutant pour chaque transition une transition de panne et une de reprise. Si cette complication n'est pas en soit un problème en terme de calcul, elle dessert en revanche, la lisiblité et la simplicité d'utilisation.

Les RP ont en revanche une sémantique mieux définie car l'hypothèse asynchrone interdit la présence d'états instables et donc ne rentre pas en conflit avec le principe de causalité. En outre, les qualités du réseau peuvent être analyser de manière statique. On privilégiera cette méthode pour des applications où les problèmes de synchronisations et d'accès à des ressources partagées sont cruciaux.

Synchrone-Asynchrone

Ce qui distingue essentiellement ces méthodes est que l'une est dite Synchrone et l'autre Asynchrone. Cette différence apparaît d'abord dans l'interface du modèle avec l'environnement. Dans les SC, celle-ci échantillonne les signaux à l'aide d'une horloge et agrège les signaux qu'elle n'a pu distinguer. Dans les RP, il s'agit d'un module idéalisé, capable théoriquement, de sérialiser tout événement. L'hypothèse synchrone induit aussi la présence d'états instables dans les SC, puisqu'une cascade de transitions (éventuellement consécutives) peuvent se produire en même temps, tandis que dans les RP, des transitions consécutives sont toujours tirées l'une après l'autre quelque soit l'état des entrées.

1.4.6. Approche mixte : Grafcet

Le Grafcet est une approche industrielle mixte entre les réseaux de Petri et les Statecharts. C'est un formalisme intégrant les notions de synchronisation sur une transition mais aussi la notion

synchrone. C'est à dire que lorsque l'on passe d'une situation S_i, à l'instant t, vers une situation S_{i+1}, à l'instant t+1, on franchit toutes les transitions possibles. Par contre, le franchissement d'une transition dure un temps non nul, à la différence des SC.

1.5. Les approches textuelles

Ces méthodes privilégient le caractère expressif par rapport au pouvoir communicatif. Si les approches graphiques contiennent une syntaxe par leur régles de constructions, les langages offrent en général une richesse beaucoup plus grande par le nombre de leur mots clefs et la façon de les combiner. Ils offrent donc un meilleur rapport "pixel sur quantité d'informations". Les approches langages (comme les approches graphiques) sont accompagnées de compilateurs ou d'analyseurs permettant d'obtenir une forme exécutable qui constitue un prototype qui peut être testé par le concepteur, cependant à l'inverse des approches graphiques, c'est cette dernière forme qui permet au client d'intervenir, toute modification devant être faite par l'intermédiaire du concepteur en raison de la difficulté d'expression dans ces méthodes.

Ces approches offrent, par rapport aux approches graphiques, une meilleure densité d'informatio Elles sont, par contre, plus hermétiques et devront donc être utilisées dans des cas où le cahier des charges est bien connu, ou bien, dans le cas où le client est déjà initié à la méthode.

1.5.1. Esterel

Esterel est un langage qui a été défini par Berry [Berry *et al.* 1987] et développé par la société Cisi Ingénierie [Ingenierie 1988] pour le développement des systèmes réactifs. Ce langage est basé sur un modèle synchrone, ce qui fait qu'un programme Esterel doit réagir instantanément à une configuration d'entrée (ensemble de signaux, valeurs de capteurs) en modifiant sont état interne (ensemble des variables locales, variables d'états, ...), et en produisant des signaux de sorties. Le programme réagit comme si ces calculs étaient exécutés par une machine infiniment rapide. C'est donc le séquencement des événements d'entrées qui dicte le comportement du programme, qui est toujours prêt à recevoir de nouvelles entrées. Les événements sont donc toujours traités en temps réel, et ne sont donc ni bufferisés ni ignorés.

Les objectifs du langage sont :

- Spécifier et implémenter une application réactive.
  Ce langage permet de prototyper le système qui peut être simulé par rapport à l'ensemble des événements qu'il doit prendre en compte. On teste son comportement par rapport aux événements qu'il est capable de fournir en sortie (traces). Une fois la spécification réalisée, l'implémentation est effectivement réalisable. En effet, ce code peut être compilé pour produire du code C, ADA ou VHDL qui peut lui-même être incorporé dans le système en développement, comme cela est illustré dans la figure III-24.
  Cette figure fait apparaître les deux phases de réalisation d'un système temps réel en Esterel. La phase de spécification (partie haute de la figure) où l'utilisateur joue le rôle de l'environnement : il teste un module écrit en Esterel en fournissant des événements et en observant les réactions du système. La partie réalisation (partie basse), où l'on indique les différents éléments constituant le système réel : un programme d'interface avec l'environnement et l'automate obtenu par compilation du module Esterel.
  Esterel est un langage qui fournit un jeu d'instruction riche et bien adapté pour la mani-

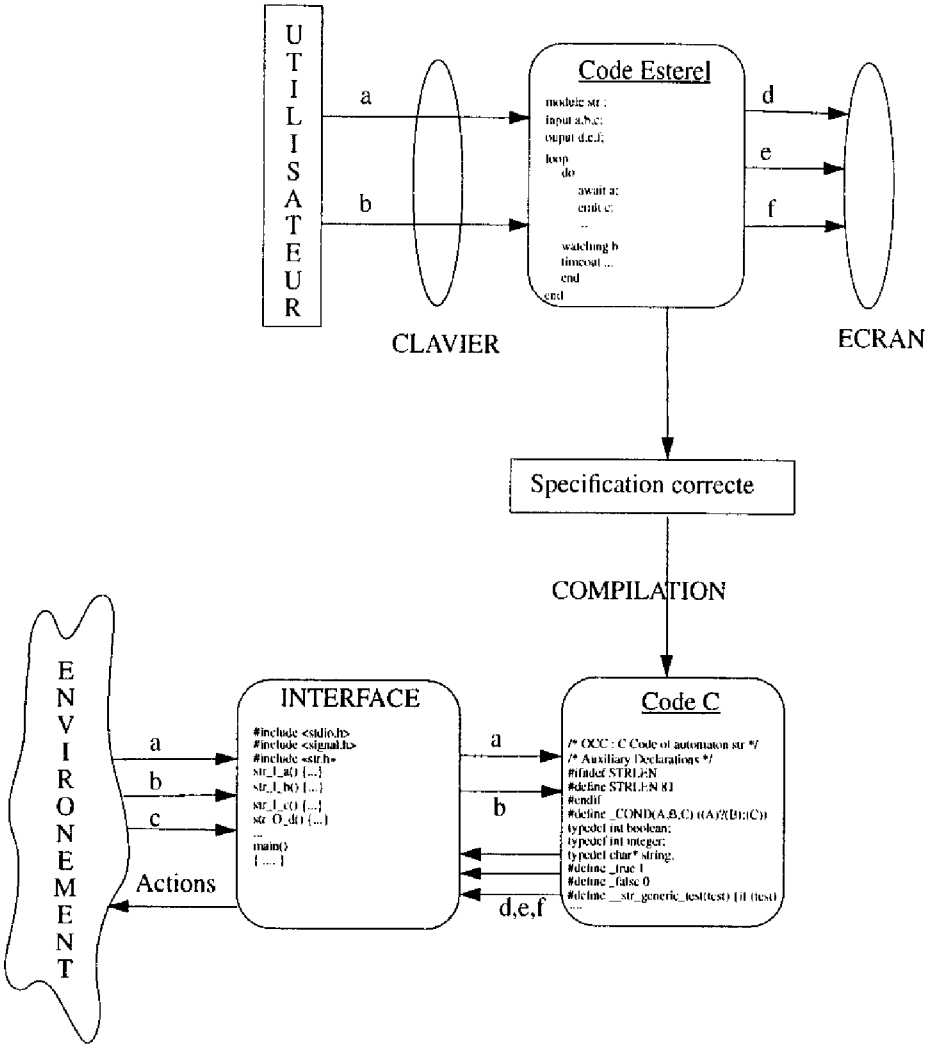


Figure III-24 : Spécification et conception en Esterel

pulation de signaux. Par contre, il ne présente que peu d'instructions pour la manipulation de données ou de types qu'il délègue à un langage hôte. Le langage Esterel peut en effet être traduit en langage cible (Fortran, Ada, ou C), et peut donc importer de ces langages des procédures types ou fonctions.
Une fois compilé, un programme Esterel permet d'obtenir une spécification exécutable et simulable. Une fois interfacé avec un programme qui échantillonne les entrées, et qui commande le procédé, le code du langage cible obtenu peut alors être directement intégré dans la conception du système.

- Faciliter la manipulation des événements.
Un grand nombre d'instructions portant sur les événements permettent l'attente, les interruptions, le lancement périodique, les exceptions et les gardes. Si dans les langages impératifs, les instructions de contrôle de flot portent essentiellement sur les variables (if then else, case, while condition, ...), en Esterel, le contrôle de flot se fait sur les événements.
- Faciliter l'utilisation des concepts de déroutement, et de gestion du temps.
On ne retrouve pas l'interruption telle qu'on la trouve avec les langages impératifs. En effet le retour d'interruption ne se fait pas en général sur la dernière instruction quittée, mais à un point de reprise défini à l'avance.

Exemple :
Le concept d'interruption d'un bloc d'instruction se fait par l'opérateur **do ... watching ... timeout** dont la syntaxe est la suivante :

do
  $instruction_1$;
  ...
  $instruction_n$;
watching S **timeout**
  $instruction_{t1}$;
  ...
  $instruction_{tn}$;
end
$instruction_{n+1}$;
$instruction_{n+2}$;

Si le signal S se produit avant la fin du bloc $instruction_1$, ...$instruction_n$, alors le contrôle va exécuter la fonction de traitement $instruction_{t1}$, ...$instruction_{tn}$, puis va en séquence exécuter $instruction_{n+1}$.Si S ne se produit pas, le contrôle passe directement à $instruction_{n+1}$.

La gestion du temps est facilitée par un ensemble d'instructions qui banalisent le temps. Ainsi on on peut exprimer simplement en Esterel, les propositions suivantes :

- toutes les n occurrences de temps, faire telle action
- faire telle action jusqu'à la terminaison d'un délai.
- exécuter périodiquement une action avec un délai de traitement; en cas de dépassement de délai, on interrompt l'action et on l'exécute à nouveau ...

Exemple :
loop
  await bouteille;
  emit remplir_bouteille;
  await fin_remplissage
each 6 seconde

Cette construction exécute périodiquement le bloc correspondant à l'attente d'une bouteille et à son remplissage. Si une bouteille n'arrive pas, ou si le remplissage n'est pas réalisé au bout de 6 secondes, alors un nouveau délai est armé, et un cycle est à nouveau commencé.

Les principes de base

- **La plupart des instructions ne consomment pas de temps :**
Seules les instructions définissant l'attente d'un événement ou l'expiration d'un délai consomment du temps. Ainsi, les instructions d'affectation, de comparaison, d'émission de signaux, seront exécutées instantanément.

Exemple :
i := 3;
i := i + 1;
j := j + i;
Si ces trois instructions vont être exécutées en un temps nul, en revanche elles vont être exécutées en séquence et au bout du traitement on aura bien i = 4 et j = 5.

Par contre, la construction suivante, correcte dans les langages impératifs (bien que n'ayant aucun sens, et devant provoquer à terme un débordement de registre) pose un problème en Esterel :

Exemple :
i := 3;
loop
  i := i + 1;
  j := j + i;
end loop

En effet, une boucle infinie dure à priori un temps infini, et le corps de la boucle est instantané; on tombe alors devant l'équation : $\infty * 0 = \infty$. Le compilateur décide donc de rejeter ce code. Pour rendre correct ce code, il faut y rajouter une instruction consommant du temps, c'est à dire une instruction d'attente d'événement. Ainsi la construction suivante est correcte.

Exemple :
i := 3;
loop

```
    i := i + 1;
    await a;
    j := j + i;
end loop
```

- **Le parallélisme est de type entrelaçage**
  La taille de l'automate résultant de la mise en parallèle de deux automates varie donc comme le produit des tailles. Il faut donc utiliser cet opérateur avec précaution, car il peut générer des automates trop gourmands en terme de mémoire.
  Un aspect particulier concerne les variables globales. Elles sont partagées en lecture mais pas en écriture. Elles peuvent cependant être modifiées par un des composants, elles deviennent alors "privées" et ne peuvent plus être lues, ni mises à jour par les autres.
  Exemple :

```
x := 3;
loop
    await x a; % attente de x occurrences du signal a
    emit c;
   ||
    x := x + 1;
    await a;
    emit d;
end loop
```

  Les instructions d'affectation s'exécutant instantanément, il y a un conflit sur la variable x : le premier composant doit-il considérer pour x la valeur 3 ou 4 ? c'est à dire, doit-on considérer que x est mis à jour avant d'être lu, ou lu avant d'être mis à jour ?
  On a déjà rencontré ces problèmes dans les SC. Ils restaient ouverts, c'est à dire que le choix de résoudre le conflit (en imposant une priorité) ou de produire de l'indéterminisme était laissé à l'utilisateur. Par contre, le compilateur Esterel rejette ce type de programmes.
- **les événements**

  Ils peuvent porter des valeurs; on les appelle alors des signaux. La valeur de ce signal peut être la combinaison de valeurs de signaux simultanés. Un programme Esterel se comporte comme un automate, un signal d'entrée, provoque des transitions internes (signaux), et produit ensuite un ensemble de signaux de sortie.
  Un automate Esterel peut accepter des signaux simultanés (à travers par exemple des composants parallèles) mais il peut aussi imposer des corrélations sur ses signaux d'entrées. Par exemple il peut exiger que tel signal doit toujours arriver en présence de tel autre, ou jamais en coïncidance avec tel autre. En cas de non respect par l'environnement de telles configurations d'entrée, l'automate n'évolue pas.

Les données

La manipulation de données se fait au moyen de variables, de constantes, et de valeurs de signaux. Il y a trois types prédéfinis : les entiers, les chaînes de caractères et les booléens. Les

opérations classiques (affectation, comparaison, ...) sur ces types sont possibles. Des types plus évolués peuvent être importés, mais alors, on doit définir un ensemble de fonctions d'accès à ces types.

Le contrôle

Il est matérialisé par un état d'attente de signaux extérieurs. Comme dans les SC, un signal peut provoquer dans l'automate Esterel une série de transitions internes, puis une stabilisation sur un état d'attente. Dans le cas de plusieurs composants parallèles, le contrôle est matérialisé par l'ensemble des états stables des différents composants.

Si l'occurrence d'un signal peut déclencher un ensemble de transitions, l'examen des conséquences de ces nouvelles transitions n'est pas à nouveau relancé. Si l'on veut exprimer des transitions de type de "réaction en chaîné' (plusieurs niveaux) on doit rajouter au niveau de chaque instruction d'attente de signaux une primitive (**immediate**) indiquant que la transition doit être tirée quelque soit l'étape d'examen de la réaction en chaîne.

Les instructions de contrôle de flot se confondent essentiellement avec les instructions de manipulation de signaux. Cependant, on trouve aussi les instructions usuelles portant sur le test de variable : **if** ... **then** ... **else**, **case** ... ainsi que les instructions de boucle **loop**, **each**, **every**, **repeat**, ... **end**.

Le pouvoir communicatif et expressif

Bien qu'étant un langage, Esterel fournit des outils simples et relativement intuitifs, notamment pour la manipulation de signaux. S'il n'est pas un outil qui peut, comme les SC, faciliter la communication avec le client, il peut en revanche permettre de concevoir simplement de petites applications temps réel sans avoir à mettre en œuvre les outils usuels de la programmation temps réel que sont les mécanismes d'interruption, les chiens de gardes, les sémaphores, les horloges, ... qui sont assez lourds à utiliser.

Au niveau du pouvoir expressif, si l'on modélise facilement les concepts de déroutement, de traitement périodiques et sporadiques, de timeout, il est en revanche difficile de modéliser la notion d'automate, et plus encore lorsque les états des automates sont eux-mêmes des automates (hierarchie d'automate).

Les propriétés

Les propriétés vérifiables sont les mêmes que celles des SC, puisque ces deux formalismes ont été fondés à partir du même modèle : l'atteignabilité, qui est la recherche dans un arbre d'un état particulier, et la franchissabilité de transitions (vivacité), qui est la recherche d'une configuration pour laquelle une transition donnée est sensibilisable.

Le temps

Tout signal est un marqueur temporel et donc le temps, peut s'exprimer en terme d'événements quelconques. Cependant, seuls les événements externes marquent l'écoulement du temps ; les évolutions internes ne consomment pas de temps. Esterel a une notion logique et discrète, puisqu'il admet en entrée des signaux simultanés.

Il y a deux échelles de temps dans Esterel. La première apparaît dans l'interface du programme Esterel, elle permet de fournir à l'automate, des signaux en échantillonant le procédé.

Cette échelle permet de faire un premier ordonnancement entre les événements. La seconde apparaît au niveau du module Esterel où ceux sont les signaux qui marquent l'écoulement du temps. C'est sur cette représentation duale du temps que sont basés les travaux que nous présenterons dans la deuxième partie.

Conclusion

Esterel est un langage permettant d'implémenter rapidement et simplement de petits systèmes réactifs. Cependant, nous pensons qu'il manque au niveau de sa syntaxe des instructions permettant d'implanter de façon claire la notion d'état et de hiérarchie d'état, et qu'il est de toute façon difficile de représenter ces concepts de façon textuelle.

Le jeu d'instructions est trop étroit et oblige à une utilisation peu orthodoxe et peu lisible des instructions dès que l'on passe à des problèmes de complexité moyenne. En outre une utilisation trop importante de l'opérateur parallèle fait exploser la taille du code et le rend donc inutilisable.

1.5.2. Traduction d'Esterel en séquences temporiées

C'est la méthode qui s'accorde le plus facilement à notre approche. En effet la représentation du temps se fait aussi par des occurrences d'événements. Ansi, le code suivant :

Exemple :
await a;
awai 2 h;
awai b;
représente la séquence ah^2b.

Nous quittons les approches synchrones pour un autre type de méthodes aussi basées sur les automates communicants, mais dont le leitmotiv est la notion d'observabilité.

1.5.3. CCS

Nous allons à travers le langage Lotos, décrire les grands principes de la famille des algèbres de processus. C'est Milner [Milner 1980] qui a introduit les principaux concepts, et Lotos [Bolognesi 1988] est l'un des langages qui les met en œuvre selon le standard OSI[11]. Milner a fondé sa théorie sur deux idées : l'observabilité et la communication synchronisée.

Si on retrouve ces principes dans toutes les méthodes que nous avons présentées jusqu'a présent, Milner les a en revanche posés en axiome de sa théorie. En effet, définir un système, c'est donner les séquences des événements que l'on peut observer en le faisant interagir avec son environnement. Il en résulte que montrer l'équivalence de deux systèmes, c'est montrer qu'ils sont équivalents "observationnellement". Il donnera notamment, une définition formelle de l'équivalence observationnelle, puis de la congruence observationnelle, qui lui permettront de fonder une démarche axiomatique. La notion de communication synchronisée, combinée à celle de restriction, donneront au modèle l'abstraction et la hiérarchie.

[11] "Open Systems Interconnexion" est une norme internationale, concernant entre autre, les architectures de réseaux, soutenu par l'International Standardisation Organisation

L'idée de définir un système par l'ensemble des séquences d'entrées/sorties qu'il est capable d'accepter ou de fournir peut s'illustrer par l'exemple suivant, où l'on décrit l'interface d'une machine distribuant du café.

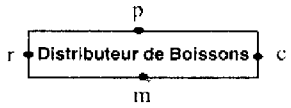


Figure III-25 : système vu par CCS

On décrit d'abord un système par ses "ports" de communication avec l'environnement. Ainsi, ce processus peut communiquer par "p", qui indique l'entrée d'une pièce de monnaie, "c" et "r" qui sont aussi des entrées, indiquant, soit la demande de café, soit le remboursement (annulation de la commande : action de l'utilisateur) et "m" qui est une sortie qui indiquant que la monnaie du remboursement est rendue au consommateur. Ces entrées/sorties ont deux états : actives ou impossibles, selon l'état du système et la séquence déjà réalisée.

Observer un système c'est communiquer avec lui. Ainsi il faudra, pour décrire ce système se synchroniser sur ces ports (p,m,r), et analyser les échanges qu'il est capable de réaliser selon les événements déjà observés (après l'événement p, le système devrait être capable de communiquer sur les ports c ou r).

Pour CCS, le système et son environnement sont deux entités comparables placés dans un même ensemble. La communication de deux composants se fait de manière synchrone, c'est à dire que chacun des composants attend que l'autre soit prêt; le rendez-vous est alors atomique. Cependant c'est une méthode qui est asynchrone, dans la mesure, où le système ne traite qu'une seule entrée à la fois (on suppose que le système est interfacé avec un observateur qui est capable de sérialiser toutes les entrées). Sauf dans le cas d'une synchronisation, il n'y a qu'une seule transition tirée à la fois.

Equivalence observationnelle

Cette notion est plus fine que l'équivalence de langages. L'observabilité permet parfois de distinguer deux séquences d'événements identiques qui n'ont pas été simulées de la même façon.

L'exemple suivant (figure III-26) donne deux spécifications du fonctionnement d'une machine à café simplifiée (elle ne distribue qu'un seul café). La spécification du fonctionnement est la suivante : après avoir introduit une pièce, l'utilisateur se voit présenter deux choix : valider en appuyant sur "c" pour avoir son café, ou sur "r" pour exiger le remboursement, qui est exécuté par l'action "m". L'option café est disponible tant qu'elle n'a pas été choisie.

Les deux automates ci-dessus reconnaissent les même traces d'exécution. On peut décrire formellement leur comportement par le système d'équations suivant :

$$S_0 = pS_1 \qquad T_0 = pT_1 + pT_2$$
$$S_1 = c + rS_2 \qquad T_1 = c$$
$$S_2 = mS_0 \qquad T_3 = rmT_0$$

Si l'on réduit ces expressions en remplaçant S_1 et S_2 par leurs définitions dans S_0, on obtient :

$$S_0 = p(c + rmS_0) \qquad T_0 = pc + prmT_0$$

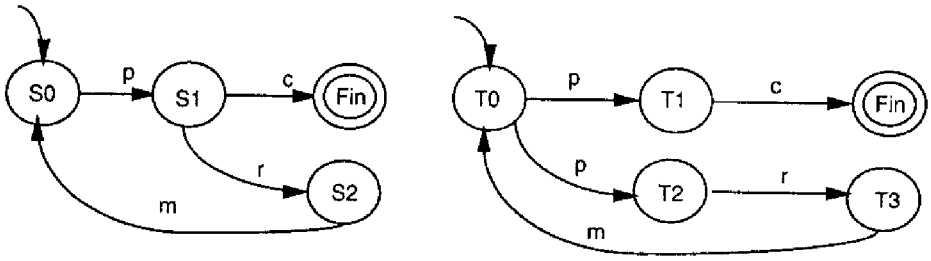


Figure III-26 : Machines à café non équivalentes observationnellement

En admettant que l'opérateur de séquence soit distributif sur l'opérateur de choix, ce qui revient à admettre que : $(a(b+c) \iff ab+ac)$, alors on obtient finalement deux équations identiques.

$S_0 = pc + prmS_0$ $\qquad\qquad$ $T_0 = pc + prmT_0$

Ce qui donnent les mêmes langages : $S_0 = (prm)^*pc = T_0$[12].

Cependant, l'utilisateur de la machine à café T_0 (machine de droite dans la figure III-26), peut être parfois surpris par ce fonctionnement. Après avoir mis une pièce, l'automate choisi de façon non déterministe l'une ou l'autre des alternatives (vers T_1 ou T_2). Le consommateur peut alors ne se voir proposer **que** le remboursement ou **que** le choix du café.

Maintenant, supposons qu'il n'y ait plus de café et que par un mécanisme interne cette option soit désactivée. Dans l'automate S_0, on se verra à tous les coups proposer le remboursement. Dans T_0, il y aura parfois la possibilité pour le système d'entrer en état d'interblocage (état T_1), où l'on perdra sa monnaie.

Il est maintenant clair, que deux automates produisant le même langage peuvent être distingué, observationnellement, et n'avoir pas du tout le même sens. C'est sur ce simple constat qu'est fondé toute la théorie CCS.

La définition formelle de l'équivalence observationnelle qui passe par la définition d'une relation de bissinulation est présentée en Annexe.

Action interne

Nous avons évoqué des actions observables qui sont le résultat de l'activité du processus. CCS introduit aussi, la notion d'action interne, sous la forme d'un événement anonyme noté **i**. Cet événement matérialise, soit une communication entre deux sous composants, soit l'évolution non déterministe du système vers un autre état.

- **i** peut modéliser une action non déterministe comme dans l'exemple de notre machine à café, où l'on peut vouloir représenter un dysfonctionnement (au dépend du consommateur, pas du gérant !) de la machine, par l'équation suivante :
  $S_0 = p(c + rmS_0 + iS_0)$
  Après avoir introduit la pièce, la machine peut spontanément retourner dans son état initial, sans avoir ni servi de café, ni proposé le remboursement.

[12] $E^* = \bigcup_{n \in N} E^n$: opérateur de fermeture de Kleene

- **i** peut modéliser une abstraction de la communication de deux sous-composants. Considérons que nous ayons, dans notre machine à café, deux sous-composants : l'un qui commande le fonctionnement normal, et l'autre détectant les pannes (III-27). Ils communiquent tous les deux par le port panne. Cette information n'étant d'aucune utilité au niveau supérieur, elle est cachée et devient une transition interne non déterministe au niveau supérieur. Pour l'automate on représentera cette transition grâce à la transition **i**.

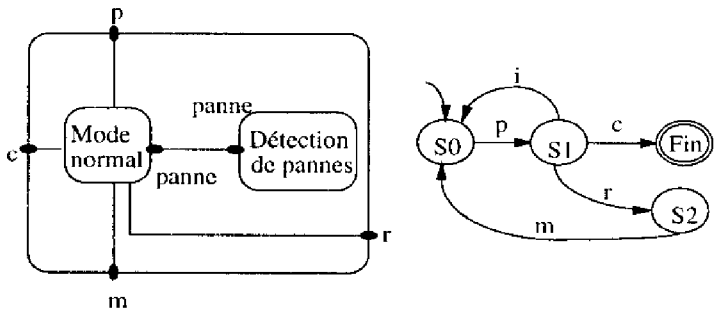


Figure III-27 : Transition interne

Communication synchronisée

La communication se fait à travers des ports dont les étiquettes identifient le signal auquel ils sont rattachés. Ces ports peuvent être reliés entre eux, comme sur la figure III-27 pour les ports "panne", s'ils ont la même étiquette. Lorsque ces deux sous-composants sont prêts pour la transition "panne", elle est tirée de manière synchrone. Une communication synchronisée qui est cachée devient une action interne.

Non déterminisme

Il est lié à l'action interne i, qui permet de modéliser une évolution spontanée du système.

Parallélisme

C'est un parallélisme d'entrelaçage, où on ne retrouve pas la notion de parallélisme vrai (RP), c'est à dire un parallélisme ou des évolutions indépendantes ne peuvent pas se réaliser simultanément. Les seules actions simultanées sont celles intervenant dans une synchronisation de processus.

La notion de parallélisme est directement liée à la notion d'observateur. Celui-ci voit un processus à travers des événements qui en sont les constituants élémentaires observables. En outre, il ne peut faire qu'une seule observation à la fois. Donc l'opérateur parallèle, entrelace une à une, les opérations élémentaires des processus. On trouve dans Lotos plusieurs versions de cet opérateur : la composition parallèle, qui relie les ports de même étiquette, la synchronisation totale, qui force deux composants à évoluer de manière synchrone, et l'entrelaçage pur où les deux composants évoluent indépendamment.

Données et contrôle

Les données sont le point faible de Lotos. La représentation des valeurs, des expressions, et des structures de données se fait à partir d'un langage de spécification de types abstraits algébriques (ACT ONE [Ehrig 1985]). Tout type (entier, chaîne de caractère, booléen, ...) se définit comme un objet mathématique appelé *algèbre* qui est défini par deux éléments :

- Sa signature, qui contient les types des variables utilisées dans la définition de l'objet et un ensemble de définitions d'opérations, qui sont en général des opérateurs de construction de ces objets.
- Un ensemble d'équations, qui doivent être satisfaites par tous les objets du type ainsi défini. Ces équations définissent des règles de réécriture qui permettent de définir des formes canoniques des objets.

Le problème dû à la construction d'objets non triviaux vient du fait que le processus de détermination d'une définition est lié à l'intuition. De ce fait, il est difficile d'obtenir une définition optimale.

La passation de contrôle d'un état à un autre est liée à la notion de dérivation d'une expression : $a.B \rightarrow^a B$. Dans le cas de composants parallèles, le contrôle passe d'un composant à l'autre; dans le cas de synchronisation, les évolutions sont simultanées.

La notion d'état telle qu'on la conçoit dans un système, (état des variables, états des ports) est implicite. Tel port d'entrée/sortie sera verrouillé, tandis que tel autre permettra le passage d'un signal. L'état du système sera représenté par l'état de dérivation des composants de l'expression générale.

Deux autres opérateurs de passage de contrôle existent :

- un opérateur d'interruption : $P[> Q$
  qui permet d'interrompre un comportement "normal" P, pour aller réaliser une fonction d'interruption Q. Au départ il se comporte comme P. Cependant, dès que la première action de Q se produit, P est interrompu et Q s'exécute. Dans le cas où P s'exécute entièrement, Q est abandonnée.
- un opérateur de phase : $P >> Q$
  Dès qu'un des composant de P se termine avec succès alors P se termine aussi et le contrôle passe à Q.

Le temps

Si LOTOS (Language Of Temporal Ordering Specification) comporte dans sa profession de foi le temps, c'est surtout en terme d'ordonnancement d'actions, et non, pour spécifier des durées entre des événements.

Plusieurs approches introduisent le temps sous forme quantitative : Timed Lotos [Leduc 1992], TIC [Quemada *et al.* 1992], RT_LOTOS [Courtiat *et al.* 1993], ..., on y retrouve les mêmes types d'extensions temporelles, que celles qui ont été rajoutées aux réseaux de Petri. On va donc y retrouver les mêmes problèmes (changements par rapport à la sémantique du modèle).

En effet, certains événements deviennent urgents, et "doivent" être tirés. L'environnement n'est plus le seul à décider de l'évolution du processus mais il doit compter avec une horloge interne qui interagit avec lui sur le système.

Dans Timed Lotos, le temps est introduit au moyen de deux opérateurs :

- $\triangle^{[d_1,d_2]}B$ qui est un opérateur unaire s'appliquant sur un comportement B (voir l'annexe sur Lotos pour la notion de comportement). Basé sur le même principe que les réseaux de Petri temporels, cet opérateur force l'occurrence du premier événement de B dans l'intervalle $[d_1, d_2]$, cette occurrence devient urgente à l'approche de la borne d_2. Lorsque cette occurrence est tirée, un événement interne est déclenché, $(thcta)$ il matérialise l'expiration non déterministe du délai dans l'intervalle $[d_1, d_2]$.
  $\triangle^{[d_1,d_2]}Q \rightarrow^{\theta} Q'$
- $\lfloor P[]\ g\ []Q \rfloor_g^d B$, est un opérateur de synchronisation temporisée (voir en annexe l'opérateur de synchronisation gardé). Si cette synchronisation n'apparaît pas avant la date d, alors cette expression se dérive en B.

Il est pris comme postulat, que le temps ne résout pas les choix. Cependant les alternatives (ou composants parallèles) vieillissent également. La seule façon alors de résoudre les choix se fait par **i**, qui prend un caractère urgent : $P[]\triangle^{[t,t]}iQ \rightarrow^{t} P[]iQ$i (voir en annexe la définition de l'opérateur de choix []) après t unités de temps, cette expression se dérive en $P[]iQ$ puis comme **i** est urgent cette expression se comporte alors comme Q.

Cette notion permet de modéliser le concept de Timeout. Si l'on reprend le sens de l'expression ci-dessus, elle signifie que P doit commencer à s'exécuter avant l'expiration du délai t, sinon c'est Q qui prendra le relais.

C'est une sémantique dite du "délai minimum", du fait de l'urgence de **i** qui fait résoudre sans attendre des choix, participant ainsi à cette politique. L'interprétation donnée à **i**, ne nous satisfait pas puisque son caractère urgent est en contradiction avec les raisons qui ont été évoquées pour sa création. **i** est une abstraction de communication de sous-systèmes de bas niveau, sur laquelle, le niveau supérieur n'a aucun contrôle. On ne peut donc pas imposer une quelconque urgence à cet événement.

TIC est une autre version de LOTOS qui étiquette les actions par des dates relatives à l'action précédente. Dans cette méthode, le temps ne résout pas le choix. Seules les actions (étiquetées) le font. Un événement est rajouté pour indiquer le passage du temps "seul" (sans occurrence d'actions). Une fonction d'âge est aussi introduite et a pour objet de vieillir les expressions, notamment pour indiquer que les composants parallèles ou les alternatives vieillissent à la même vitesse. Cette fonction n'étant défini que de manière dénotationnelle (règles de réécriture simples), elle n'interfère pas au niveau de la sémantique des opérateurs existant.

Les restrictions temporelles agissent sur les entrelacements de comportement. Ceux-ci sont définis en fonction des dates des occurrences des événements à entrelacer. Il n'y a plus de degré de liberté et une expression contenant deux composants en parallèle se dérive en général en un seul composant.

Lorsqu'une action est étiquetée avec une date non nulle, ou bien elle se déroule à la date prévue et la dérivation se poursuit normalement, ou alors l'expression qui la contient se transforme en processus bloqué (stop). La notion de "devoir" est donc appliquée aux événements observables ainsi qu'à **i**.

Il faut noter la possibilité, dans cette méthode, d'étiqueter une action par une durée nulle. Cela introduit alors la possibilité d'événements simultanés que l'on a dans les langages synchrones. Dans le cas le plus défavorable, si l'on combine cette hypothèse (qui change la sé-

mantique du parallélisme) à l'opérateur de récursion, on pourrait générer des suites infinies d'événements (de durée nulle), ce qui constituerait le même paradoxe que celui rencontré dans Esterel (les boucles infinies instantanées).

La démarche nous semble intéressante. Cependant comme pour la version précédente, l'interprét; de i ne nous satisfait pas. Il ne nous semble pas judicieux de pouvoir étiqueter i, qui par définition est un événement non maîtrisable, dont l'occurrence est théoriquement indéterministe.

Il faut noter que ces approches introduisent des idées intéressantes :

- Décrire un système par ses événements observables.
- Le parallélisme peut avoir une représentation mathématique simple si l'on considère la notion d'entrelaçage.
- Les différents type d'entrelaçage : d'entrelaçage gardé, synchronisation pure, ou d'entrelaçage pur.

Cependant nous leur reprochons :

- Le temps quantitatif déforme la sémantique du modèle.
- Une utilisation importante de l'opérateur de parallélisme "segmente" la compréhension globale du système. Pour comprendre le système dans sa globalité, il faut entreprendre un test dynamique. Le système ne peut alors se comprendre, que lorsque l'on fait communiquer tous ses composants. Une analyse statique, comme cela est possible avec les réseaux de Petri pour les propriétés structurelles (indépendantes du marquage), n'est pas possible. Par exemple, si l'on considère un ensemble de contraintes, en Lotos, la mise en œuvre se fera en mettant en parallèle les comportements correspondants à chaque contrainte. Pour vérifier que cet ensemble est cohérent, il faudra trouver au moins une exécution réussie.

2. Approche Logique

On désigne par le qualificatif "logique", toutes les approches basées sur un système axiomatique. Ce sont des théories (au sens mathématique du terme), c'est à dire des ensembles comprenant des axiomes (vérité ne nécessitant aucune démonstration), des règles (permettant des transformations syntaxiques) et des théorèmes, qui sont des expressions construites à partir des axiomes et des règles.

L'expression de ces différents objets se fait à l'aide d'un langage qui comprend en général, un noyau de règles de construction syntaxique basé sur les connecteurs usuels de l'algèbre de Boole AB ={Vrai, Faux, et, ou, non}, et un ensemble d'extensions qui les particularisent.

Exemple :
On donne par exemple le langage permettant de construire la théorie des nombres entiers.

L = {"0" (élément de base), "succ"(la fonction successeur)}

En appliquant n fois la seule fonction sur la seule constante du langage, on obtient un élément "assimilable" à l'entier n. Donc l'ensemble des entiers naturels $\mathcal{N}$ peut se définir comme la fermeture de la fonction "succ" sur l'élément "0".

$N = \{0, succ(0), \cdots, succ(\cdots succ(0)\cdots), \cdots\}$

Ces approches se distinguent des approches "système de transitions" (ST), dans le sens où leur représentation du système est à la fois plus abstraite (elle représente souvent des classes de systèmes) et plus concise (on est parfois obligé de décrire des éléments aussi détaillés que des opérateurs arithmétiques, ou de préciser les caractéristiques de chaque entité utilisée).

Si les approches logiques décrivent essentiellement le procédé de manière statique, les approches ST privilégient la dynamique. En effet, si certaines propriétés peuvent tout de même être analysées de manière statique, c'est à dire indépendamment des conditions d'exécution (comme l'analyse des composantes conservatives pour les réseaux de Petri, la détection de condition de blocage pour les Statecharts, la détection d'erreur de causalité dans Esterel, ...), le nombre de ces propriétés est restreint. En revanche, pour les approches logiques, il n'est limité que par la richesse syntaxique du langage.

On donne en annexe, une présentation globale des systémes logiques et de leur fondements axiomatiques. On présente les définitions de base (formules, axiomes, théorèmes, interprétations), puis on aborde la logique des propositions, la logique des prédicats, la logique modale et la logique temporelle. Les méthodes logiques de spécification des systèmes temps réel que nous allons aborder sont basées soit sur la logique des prédicats soit sur la logique temporelle.

2.1. Méthodes de spécification basées sur la logique

Nous allons aborder les caractéristiques principales des méthodes de spécification basées sur des systèmes axiomatiques, c'est à dire des théories. Nous allons présenter RTL, qui est une théorie basée sur la logique du premier ordre, puis nous aborderons une théorie basée sur la logique temporelle, et enfin, une logique temporelle d'intervalle RTIL.

2.1.1. RTL

Introduite par Jahanian et Mok [Jahanian 1986], RTL est une logique des prédicats, dans laquelle le temps est introduit sous la forme d'une fonction non interprétée qui associe une date à l'occurrence d'un événement.

La première étape de la méthode est de capturer la sémantique du système temps réel à travers un modèle événement/action. Ce modèle est un pseudo-langage qui permet de définir les contraintes temporelles du système. Les contraintes temporelles qualitatives, c'est à dire celles concernées par l'ordre relatif des actions, sont déterminées à l'aide d'instructions classiques de séquence, de synchronisation, et de parallélisme; tandis que les contraintes temporelles quantitatives sont capturées à l'aide de deux types de contraintes : périodique et sporadique.

Outre cette classification de contraintes temporelles, ce modèle classe les événement en quatre types :

- les débuts de tâches notés $\uparrow A$.
- les fins de tâches notées $\downarrow A$.
- les transitions internes qui marquent le changement de valeur d'une variable quelconque du système.
- les événements externes.

Ce modèle est ensuite transformé de manière automatique en un ensemble C d'axiomes et de

théorèmes. Cette théorie a ensuite pour but de vérifier une propriété. Une fois celle-ci exprimée à l'aide du langage RTL, la méthode de preuve consiste à montrer qu'il n'existe pas de fonction de datation (il n'existe pas de fonction d'assignation de dates pour les événements) qui soit compatible avec $C \wedge \neg F$. Le problème comme nous l'avons déjà évoqué, est qu'il n'existe pas de procédure de déduction pour les logiques du premier ordre. Par contre, RTL peut se ramener à la théorie arithmétique de Presburger [Shostack 1979] pour laquelle existe une telle procédure.

On remarquer que cette approche ne peut se passer d'un modèle type "système de transitions", lequel est ensuite transformé en un ensemble comprenant un nombre important d'axiomes et de théorèmes. Cette démarche est intéressante dans la mesure où elle établit une classification de contraintes temporelles et qu'elle définit pour chacune d'entre elles une expression spécifique. Elle base ensuite l'analyse des contraintes temporelles sur un ensemble minimal de contraintes prédéfinies (périodiques et sporadiques).

2.1.2. Théorie fondée sur la logique temporelle

Nous allons présenter une théorie proposée par Bernstein [Bernstein 1981], basée sur la logique temporelle. L'approche générale développe l'idée qu'il n'est pas nécessaire de se préoccuper de la puissance d'exécution de la machine d'implantation, mais plutôt que le "spécifieur" doit porter son effort sur l'analyse des contraintes temporelles ce qui lui permettra de déduire l'ordonnancement temporel du programme, et la puissance de calcul nécessaire. La logique temporelle, intervient alors pour prouver que les contraintes temporelles ont bien été prises en compte. Bernstein introduit une théorie spécifiquement adaptée aux STR. Elle permet notamment de prouver des propriétés du type :

(prêt à recevoir une entrée) $\rightsquigarrow^n$ **(Prêt à recevoir une nouvelle entrée)**

qui expriment que le temps de traitement de données ne doit pas excéder le délai minimum n, séparant l'arrivée de données consécutives. L'opérateur $\rightsquigarrow$, est un opérateur modal exprimant une caractéristique de type "correction totale", il peut en effet se réécrire par :

$$P \rightsquigarrow Q \equiv \Box(P \rightarrow \Diamond Q)$$

Qui signifie : "Lorsqu'une propriété P est vérifié, alors Q le sera".

Cette théorie s'appuie aussi sur un modèle d'exécution de programme, qui est modélisé par une séquence d'états dont chaque transition est une opération indivisible. Ici le temps n'apparaît pas, comme dans la précédente théorie, comme un paramètre d'une fonction de datation, mais au niveau du modèle d'interprétation, dans la relation liant les états.

Un état porte une structure de donnée, comportant la date à laquelle le contrôle est passé dans l'état, une fonction d'affectation de valeurs des variables de programmes, un ensemble d'opérations candidates à l'exécution, et un ensemble d'opérations suspendues. Ces deux derniers ensembles capturent les dépendances de données comme on le faisait dans la précédente théorie, à l'aide du pseudo-langage.

Bernstein définit des sous-séquences, qui représentent l'exécution d'une série d'opérations, qui se termine dès lors, qu'il ne reste aucune opération candidate à exécuter. Il considère que le temps d'exécution de ces sous-séquences est négligeable comparé au temps de sommeil d'un processus. C'est à dire que le séquencement des événements provoqués par les périphériques

du système domine l'ordonnancement temporel du programme. Cette idée importante a été reprise comme hypothèse de base pour la conception des langages synchrones, que nous avons déjà évoqué dans ce chapitre.

Chaque état étant daté, il est possible de poser des contraintes d'intervalle entre des événements. Bernstein propose alors des formules dont l'interprétation est celle de contrainte temporelle "au plus tôt" et "au plus tard".

L'axiomatique de cette théorie comporte des règles de composition de contraintes temporelles :

- La règle de progrès :

$$\frac{A \rightsquigarrow^{<n} B, C \rightsquigarrow^{>n} D}{A \wedge C \Longrightarrow (\neg B \Box \neg (D \vee \bigcirc D))}$$

  Dans la prémisse, on a deux contraintes temporelles, l'une de type maximum, indiquant que B doit arriver au plus tard n unités de temps après A, et l'autre qui indique que D doit arriver n unités de temps au plus tôt après C. Cette règle d'inférence indique que si A et C sont arrivés en même temps, alors B est arrivé avant D.

- La règle d'addition compose deux contraintes temporelles de type maximum :

$$\frac{P \rightsquigarrow^{>n} Q, Q \rightsquigarrow^{>m} R, PATH(P,Q,R)}{P \rightsquigarrow^{>n+m} R}$$

  Elle signifie, que si Q est attendu au plus tard n unités de temps au plus tôt après P, si R est attendu m unités de temps au plus tôt après Q, et s'il existe une séquence d'exécution où ces trois événements sont consécutifs, alors R est attendu au plus tôt après $n + m$ unités de temps.

Cette théorie qui s'applique sur des programmes, contient pour chaque type d'instruction (**if then else**, **while**, ...) un axiome, décrivant l'effet de l'instruction considérée.

Bernstein a introduit des idées importantes que l'on retrouve dans les langages synchrones. Comme eux il considère que la vérification de propriétés temporelles est indépendante de la vitesse d'exécution du matériel d'implantation, et que l'ordonnancement temporel du programme est dominé par les événements d'entrées/sorties. Pour cela, il fait l'hypothèse que le temps d'exécution de sous-séquences, (celles qui traitent l'arrivée de données) est négligeable, c'est à dire, dit d'une autre façon, que ce temps est inférieur au délai inter-arrivée de données le plus petit ("négligeable" est à prendre au sens de "sans conséquence" puisqu'il n'y a pas de perte de données).

En outre, il introduit deux contraintes temporelles types, minimum et maximum, avec lesquelles il espère pouvoir exprimer tout type de contraintes temporelles, et pour lesquelles il présente deux types de compositions, qui sont des règles d'inférences. Nous reprendrons ces idées au niveau de notre approche dans la deuxième partie de ce mémoire.

2.1.3. RTIL

Le but de la théorie RTIL [Razouk 1989] est de vérifier, si les traces d'exécutions produites par un système temps réel sont consistantes avec la description formelle du comportement de programme désiré.

Jusqu'à présent, le temps était modélisé par une estampille, correspondant soit à un état (Logique temporelle), ou bien, à l'occurrence d'un événement (RTL). Dans cette théorie apparaît la notion de date "temps réelle", c'est à dire ne correspondant pas à l'observation d'un événement. Par exemple, assurer dans les théories précédentes qu'une variable v conserve la valeur zéro entre les dates 5 et 10, n'était possible que si les dates 5 et 10 correspondaient, soit à la date d'entrée dans un état (pour la théorie de Bernstein) soit à l'occurrence d'un événement (pour celle de Jahanian et Mok).

Le modèle d'exécution est une séquence d'états et de transitions. La notion d'intervalle est liée à celle de séquence d'états : c'est un état initial, une séquence d'états internes, et un état final. Le temps est introduit en associant des étiquettes temporelles aux points de transitions. Cependant, la notion de datation se démarque tout particulièrement des autres démarches :

- La séquence des dates forme une suite croissante mais non nécessairement monotone.
- Tous les événements simultanés, du point de vue de l'observateur/dateur sont attachés au même point de transition.
- Tous les événements non simultanés, mais datés dans le même instant, sont sérialisés comme un ensemble ordonné, mais ayant la même date.

Cette approche, tout à fait originale, fait apparaître deux échelles de temps : une pour la saisie des événements, qui permet de distinguer des événements, et l'autre qui les date avec une précision donnée. Nous reprendrons à notre compte cette représentation du temps dans notre approche.

Dans cette approche, un intervalle est donc défini par une séquence et une suite de dates. Un point "temps réel" est la donnée d'une date, qui ne correspond pas avec l'occurrence d'un événement.

Exemple :
Considérons l'intervalle I de la figure III-28, défini par la séquence de transition $(t_1, t_2, t_3, t_4, t_5)$, associée à la séquence de date $(0, 2, 2, 8, 10)$. Un exemple de "point temps réel" (pseudo-transition partageant un état, et ne correspondant pas un événement) est la date 9 qui partage s_4. On peut remarquer dans cette séquence que si les transitions t_2 et t_3 ont la même date, elles sont en revanche, consécutives.

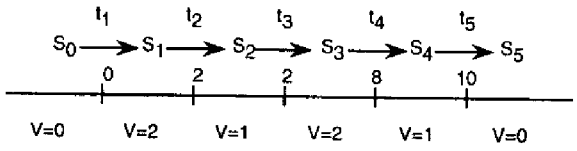


Figure III-28 : Point temps réel

Cette théorie est basée sur l'évaluation d'une formule sur un contexte, qui est un intervalle $[I]\alpha$, où α est une formule booléenne, construite sur les connecteurs de la logique des propositions et des opérateurs modaux. Une fonction de "datation" appliquée à une formule et à une séquence d'exécution, permet de déterminer le premier intervalle où la formule est vraie. Ainsi dans l'exemple : $I[\mathcal{O}(v = 2)] = [\{2, 10\}, \{s4\}]$.

Pour capturer les contraintes temporelles, les auteurs introduisent des contraintes périodiques et sporadiques, telles qu'elles sont définies dans RTL. Cette logique considère qu'un observateur est capable de produire une trace composée d'événements datés. Si plusieurs événements ont la même date, ils sont tout de même sérialisés.

Les auteurs de cette théorie signalent qu'une recherche devrait être faite pour développer une représentation standard et indépendante du modèle futur d'analyse des traces d'exécution datées. C'est en fait l'idée que nous développons avec notre modèle de traces temporisées.

2.1.4. Relation de RTIL avec la notion de séquences temporisées

La notion de trace et d'observateur qui est considérée dans cette théorie est similaire à la notre. La représentation des traces, telles que nous la concevons, est tout à fait compatible avec ce modèle de représentation.

L'entrelacement d'occurrence de l'événement h et d'événements observables, correspond tout à fait à la notion d'intervalle et de point "temps réel" présente dans cette théorie. Les traces temporisées que l'on a évoquées au chapitre I, sont des suites croissantes (non forcément monotones) de dates, avec des événements éventuellement consécutifs mais ordonnés.

Ainsi, la trace $s_1\ h\ h\ s_2\ s_3\ h\ h\ h\ h\ h\ h\ s_4\ h\ h\ s_5$ est une autre représentation de la séquence donnée dans l'exemple de la figure III-28.

3. Approche mixte

Si l'on a vu que toutes les théories étaient liées à un modèle qui est un système de transitions, il existe des passerelles dans l'autre sens. Il est en effet possible de passer d'un système logique à un système de transition avec les Réseaux Prédicats Transitions (RPPT).

Un problème exprimé sous la forme d'un ensemble de clauses de Horn peut être traduit dans un formalisme dérivé des réseaux de Petri, les RPPT, qui permet de donner à un système logique l'aspect d'un système de transition.

Les RPPT permettent de chaîner des squelettes de raisonnements. D'autre part, l'analyse linéaire du réseau sous-jacent (la recherche de composantes conservatives) permet de déterminer des scénarios de bons raisonnements.

4. Comparaison globale et générale

La comparaison des mondes "logique" et "système de transitions", est difficile, tant leurs objectifs sont différents. Les systèmes de transitions ont essentiellement pour but de modéliser le contrôle, mais leur pouvoir de vérification de propriété est plus faible que celui des systèmes logiques.

Cependant, on peut dire qu'ils sont complémentaires, puisqu'un système logique s'appuie toujours sur un système de transitions. L'un privilégie l'aspect dynamique, l'autre l'aspect déductif. Les systèmes de transitions sont plus aisément compréhensibles, et facilitent donc la communication. Par contre les systèmes logiques ont une vue plus abstraite de l'application, et permettent d'en exprimer facilement les propriétés.

Les systèmes logiques ont pour unique but de vérifier des propriétés. Cette capacité est

enrichie au niveau du langage par des opérateurs qui permettent de poser une grande variété de "questions"[13] sur le système, et de réaliser une vue très abstraite (on le voit comme une théorie mathématique) et très détaillée (on est obligé d'axiomatiser, l'arithmétique, la notion du temps que l'on désire, ...) de la spécification. Les exemples que l'on peut trouver dans la littérature montrent l'explosion du nombre d'axiomes, pour des petits exemples, ce qui pose un réel problème quant à l'utilisation de ces méthodes pour des problèmes de complexité normale. Leur aspect formel en font un outil de travail pour spécialistes, et ne peuvent servir à communiquer avec le client comme on peut le faire avec les Réseaux de Petri ou les Statecharts.

5. Conclusion

Si aucune méthode présentée jusqu'a présent ne permet de répondre à l'ensemble des problèmes que posent la spécification des systèmes temps réel, en revanche, une fois l'objectif de la spécification posé (preuve d'une propriété de non dépassement de capacité, simulation d'un processus d'interruption avec reprise, établissement de la conformité d'une implémentation par rapport à une spécification, ...), il existe des méthodes plus adaptées que d'autres pour répondre à ces problèmes.

L'approche que nous allons présenter dans la prochaine partie, n'est pas une nouvelle méthode qui résoudrait tous les problèmes plus facilement et plus efficacement, ...Mais elle se présente comme un outil complémentaire, sur le point où les autres achoppent : **le temps.** Si toutes ces méthodes se sont vues proposées des extensions "temporisées", en revanche peu d'outil "dédiés" existent pour la vérification de propriétés temporelles quantitatives.

L'ensemble des méthodes de spécification que nous avons présentées nous a aussi permis de montrer qu'un format de traces temporisées commun pouvait exister. Notre approche est basée sur ce format de traces, sous lequel ces dernières peuvent être dérivées par des méthodes types systèmes de transition. Notre proposition se positionne donc en aval de la spécification usuelle des systèmes temps réel et a pour but de vérifier si les contraintes temporelles ont bien été prises en compte dans la spécification.

[13]Poser une question, c'est prouver qu'un ensemble de clause définissant le Système S est consistant avec la question Q, c.a.d que $F = S \rightarrow Q$, ce qui peut se montrer en montrant que $\neg F$ est insatisfaisable

PARTIE B

Une approche pour l'expression et la validation des contraintes temporelles

CHAPITRE IV

UNE APPROCHE POUR L'EXPRESSION DES CONTRAINTES TEMPORELLES

1. Introduction

Dans le chapitre I, à partir d'exemples simples, nous avions montré que toute contrainte temporelle pouvait être interprétée comme une propriété de séquencement d'événements. Cela ne signifie pas que les contraintes temporelles que nous évoquerons ne se résument qu'à des contraintes de précédence. Les contraintes que nous manipuleront seront aussi quantitatives, c'est à dire qu'elles mettront en jeu des intervalles entre des événements.

Exemple :
$TC = ahbchhf$
représente les contraintes temporelles suivantes:

- une seconde de temps a et b une unité de temps
- trois secondes de temps entre a et f
- deux secondes de temps entre c et f
- c précéde f, ...

Cette séquence contient donc implicitement un ensemble de contraintes temporelles.

Cette considération est basée sur l'existence d'un événement particulier marquant l'écoulement du temps. On retrouve la notion de marqueur temporel dans le langage Esterel [Berry *et al.* 1987], où le temps est exprimé de manière multiforme, c'est à dire où tout événement peut servir de base de temps. Cependant nous nous démarquons de ce concept, dans la mesure où cet événement est effectivement produit par une horloge, et qu'il prend un caractère particulier puisque l'on considère que seul cet événement marque l'écoulement du temps. Deux événements non séparés par une occurrence du marqueur temporel, ont la même date, ce qui n'est pas le cas en Esterel. Le temps s'écoule dans ces séquences de manière croissante mais non monotone, comme dans la théorie RTIL que nous avons vue précédemment.

On a montré à partir des méthodes type système de transitions, (cf chapitre III), qu'il était possible de produire directement (en Esterel) ou de dériver (à partir des réseaux de Petri, des Statecharts, ou de Lotos) ce genre des traces comportant des événements entrelacés avec des occurrences d'un marqueur temporel. Ces traces que nous appelons "séquences temporisées", constituent donc un format commun de représentation de traces d'exécution temporisées.

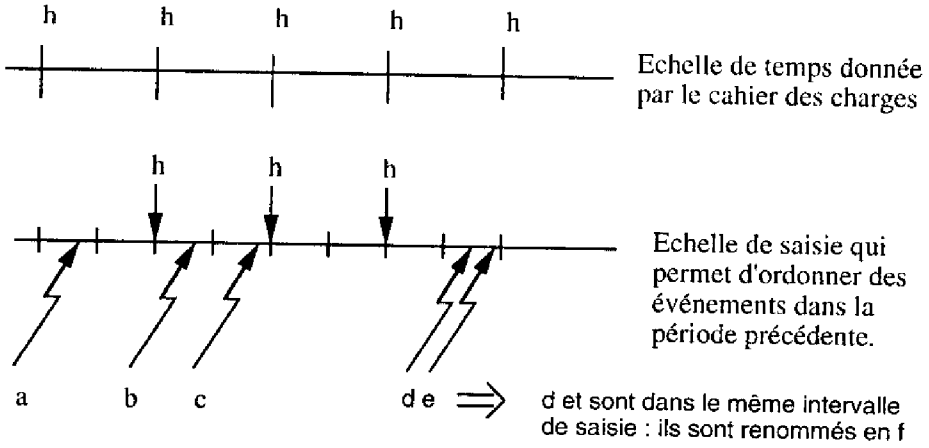
Mais quelle signification peut-on donner à ces traces ? A quel type de temps correspond cette notion, par rapport aux définitions que l'on a pu donner au chapitre I ?

1.1. Interprétation des séquences temporisées

Soit un cahier des charges donnant une précision de travail au niveau de la réception des événements. Les séquences temporisées correspondent à l'observation d'un processus (par ses événements observables) selon deux fréquences différentes (voir la figure IV-1). L'une, qui permet de dater les événements selon la précision donnée dans le cahier des charges. L'autre échelle correspond à une échelle de saisie. Cette échelle a une fréquence plus grande qui permet de saisir un ordre relatif entre les occurrences d'instants de la première fréquence. Cette dernière, permet de donner un sens aux séquences temporisées. Dans l'exemple précédent, la fréquence donnée dans le cahier des charges est de l'ordre de la seconde, et donc dans la séquence $TC = ahbchhf$, les événements c et c sont séparés par une seconde. Par ailleurs, si b et c ont la même date, ils sont en revanche ordonnés grâce à la fréquence de saisie. Par contre, si deux événements ne peuvent être départagées par la fréquence de saisie (figure IV-1) alors on considère qu'ils sont renommés en un signal.

La période de la fréquence de datation correspond à la précision juste nécessaire pour le système, tandis que la fréquence de saisie permet de saisir un niveau supplémentaire de précision, celui de l'ordre des événements dans la période dite de "de précision". Ce concept était présent dans les notions de macro-pas et micro-pas [Harel *et al.* 1987] qui permettent de justifier les inconsistances de tir synchrone de transitions (voir le chapitre précédent), ainsi que dans le modèle d'interprétation de RTL [Razouk 1989].

La figure suivante illustre ces deux échelles de temps, où l'une sert à établir un ordre entre les événements, et l'autre sert à les dater par rapport à une fréquence plus petite. Ainsi, si b et c ont la même date, ils sont en revanche ordonnés. Par contre, d et e qui sont dans le même top, sont arrivés dans un laps de temps tellement proche, par rapport à la fréquence de saisie, qu'il y a une incertitude sur leur ordre d'arrivée, et ne sachant pas comment on peut les ordonner, ils sont agrégés en un seul signal f, représentant la concordance des deux signaux [1].



Séquence temporisée obtenue = a h b c h h f

Figure IV-1 : Deux échelles de temps

1.2. Caractéristiques du temps considéré

Le temps adopté est :

- discret : les dates correspondent à des nombres d'occurrences d'événements donc à des entiers
- logique : on admet la simultanéité des événements
- non abstrait : le temps est représenté par un événement
  Il y a deux niveaux de temps :
  - La temps qualitatif, qui est capturé par une fréquence de saisie
  - Le temps quantitatif, c'est à dire mesuré entre deux événements observables

Remarque 3 : La simultanéité d'événements (entre deux événements observables) que nous adoptons est un point de vue plus proche de la réalité que le point de vue asynchrone (qui n'admet pas le principe de simul-

[1] Si un événement arrive en concordance avec un événement h, c'est à dire si la fréquence de saisie ne permet pas de le distinguer de h, on considère alors que h est arrivé avant. Cela signifiera que cet événement aura la date h. Cette hypothèse se conçoit aisément, car l'erreur commise est alors de l'ordre de la fréquence de saisie, qui est inférieure à la précision demandée.

tanéité). En effet, la capacité de discerner deux événements consécutifs qu'a l'observateur [2] est de toute façon bornée. Il lui est donc impossible d'ordonner tous les événements.

2. Approche générale

L'approche que nous développons a été proposée initialement par A.E.K. Sahraoui ([Sahraoui 1994]). Elle n'est pas une nouvelle méthode de spécification, mais se veut complémentaire par rapport aux méthodes de spécification que l'on a précédemment présentées pour les aspects temporels. Elle permet l'expression et la validation des contraintes temporelles de façon séparée ([Delfieu 1994a]) par rapport à la spécification complète (qui prend en compte toutes les contraintes : temporelles et fonctionnelles).

Cette complémentarité peut mieux se résumer par le schéma IV-2 où la partie gauche représente les étapes classiques de conception d'un STR, tandis que la partie droite représente les différentes étapes de la méthode que nous proposons. Dans cette méthode, la première phase consiste à extraire les contraintes temporelles du cahier des charges sous une forme que nous préciserons par la suite. La deuxième étape a pour but de recomposer toutes ces contraintes de manière à en obtenir une forme générale. Cette forme générale, exprimée sous forme de grammaire constitue un analyseur syntaxique de traces. Effectivement, une grammaire (régulière) se transforme facilement en analyseur par le biais de l'outil Lex&Yacc. Un analyseur syntaxique est un programme ayant pour entrée des mots (qui seront les séquence temporisée que nous avons définies) et qui produit un résultat de type booléen : Vrai si le mot peut être généré par la grammaire Faux sinon. L'objet de notre analyseur sera donc de vérifier les jeux de tests produits (partie gauche de la figure IV-2) par la simulation de spécifications intégrant le temps. Selon le résultat de cette analyse, on poursuivra la conception du système, ou l'on reviendra sur la spécification de l'application.

Remarque 4 : L'analyse des contraintes temporelles peut aussi fournir des indications de performances pour réaliser la future architecture d'implémentation (partie basse de la figure IV-2)

L'objectif de la suite du mémoire est la présentation de ces différentes étapes. Pour pouvoir évoquer l'extraction de contraintes temporelles, nous allons d'abord les analyser. Nous proposerons ensuite une représentation formelle et enfin nous montrerons comment réaliser l'analyseur.

3. Analyse d'une contrainte temporelle

Dans un cahier des charges comportant une description en langage naturel, ces contraintes peuvent se repérer par les expressions : "à chaque période", "lorsque", ...Dans toutes ces expressions, sont exprimées des contraintes, qui ne se réfèrent pas toutes à la même origine temporelle.

Dès l'extraction, une analyse du cahier des charges permet de distinguer deux types de contraintes: les contraintes identifiées par les expressions "au minimum", "après", "à la date", qui sont des contraintes sur des datations d'événements ; et un second type de contraintes qui

[2] Un observateur est celui qui permet de sérialiser les entrées/sorties d'un système. Il peut aussi s'interpréter comme un sous-système qui joue l'interface entre le programme et l'environnement

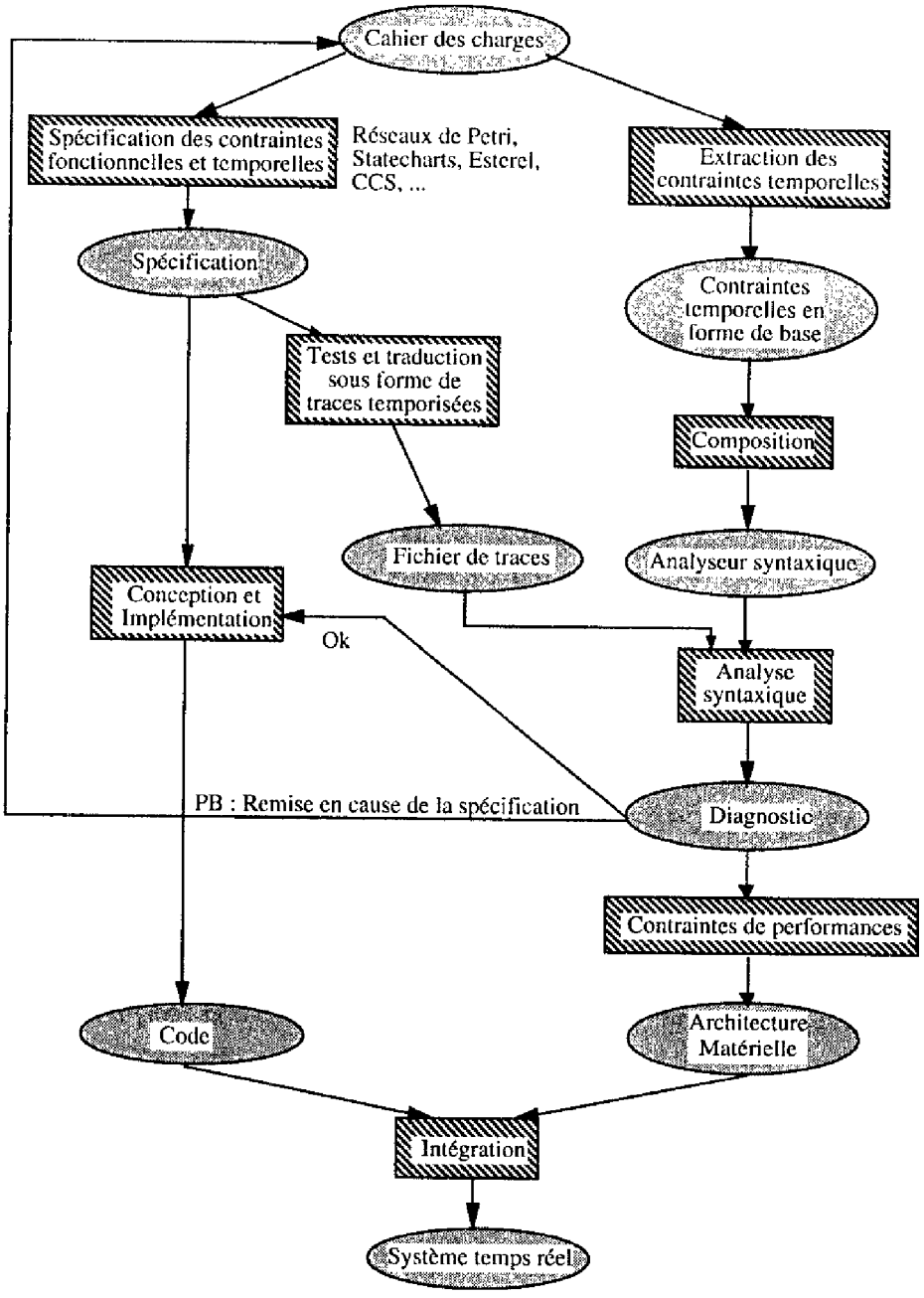


Figure IV-2 : Approche générale

sont des contraintes de performances repérées par les expressions : "avant", "au maximum". La difficulté résidera alors dans la satisfaction effective de ces contraintes, qui est due à la criticité du temps de réponse et souvent à la limite de la technologie ou du savoir faire logiciel. Cependant même si ces contraintes sont a priori difficile à réaliser du point de vue technique, comme on l'a évoqué au chapitre II, **ce n'est pas un problème relevant de l'étape de spécification**. De plus, les contraintes de performances ne sont pas toutes à priori forcément incompatibles avec la puissance de calcul de la machine d'éxécution. On considérera donc toutes les contraintes temporelles comme **satisfaisables** par le système et donc toutes ses contraintes seront toutes prises en compte de la même façon dans notre approche. Par la suite, nous ne faisons d'ailleurs plus de distinction entre contrainte de performance et les autres contraintes qui deviennent respectivement contrainte "maximum" et contrainte "minimum".

3.1. Nos objectifs quant à l'utilisation de ces contraintes

Si le domaine du STR est souvent associé à celui de la recherche de l'exécution la plus rapide, en revanche, les méthodes de spécification des STR ne proposent pas de solution pour obtenir des gains de performances. Nous considèrerons que le problème de performance devra être traité lors de la conception détaillée, et qu'il existe à priori une solution (architecture spécifique ou parallélisation) pour obtenir la performance requise. Les contraintes de performances étant à priori toutes satisfaisables (sinon le système n'est pas réalisable), nos objectifs sont donc les suivants :

- Vérifier que la spécification est bien conforme à ce que veut le client, en terme de contraintes temporelles [3].
- Vérifier qu'une spécification a bien pris en compte toutes les contraintes temporelles présentes dans le cahier des charges.
- Montrer qu'il n'y a pas de redondance entre les contraintes. Une redondance peut même si elle n'introduit pas d'incompatibilité révéler un problème dans l'expression du cahier des charges tel qu'il a été conçu par le client (ce qui renvoie au premier point).
- Montrer aussi qu'une implémentation est possible, et indiquer une performance minimale[4] nécessaire pour l'architecture cible.

Il existe des contraintes produisant des ensembles infinis de traces. Dès lors, il est nécessaire d'envisager une représentation plus compacte que celle d'un ensemble de traces. En outre, une représentation formelle permettrait de pouvoir raisonner sur ces contraintes : composer des contraintes, trouver l'intersection de contraintes, ...

3.2. Représentation des contraintes

Nous avons établi qu'une contrainte pouvait être représentée par une séquence, ou un ensemble de séquences temporisées, qui souvent ont des similitudes syntaxiques. Un ensemble de sé-

[3] Cette étape peut paraître superflue pour des applications traditionnelles, mais la criticité et le coût de prototypage des STR justifient en général son utilité.

[4] Il faut noter que cette performance dépend fortement des choix fondamentaux de conception : systèmes distribués, exécutifs temps réel, ...et donc cela nous donnera plutôt une indication sur une performance minimale.

quences peut donc être considéré comme un langage, dans la mesure où il définit un ensemble de règles de constructions syntaxiques.

3.2.1. Grammaire d'une contrainte temporelle

Toute contrainte temporelle s'exprime au niveau des traces par une propriété particulière. Par exemple, une contrainte exprimant "a est attendu au plus tôt 3 unités de temps après o" est représentée par l'ensemble des mots commençant par le préfixe $ohhh$:

$$C_{min} = ohhha, ohhhha, ohhhhha, \cdots$$

En représentant n occurrences consécutives de h par h^n, et si l'on introduit l'opérateur "+" pour exprimer les différentes alternatives, cet ensemble peut alors être représenté par l'expression :

$$C_{min} = (oh^3a + oh^4a + \cdots + oh^na + \cdots) = oh^3 \sum_{i=0}^{\infty} h^ia$$

Un problème réside cependant, dans la compacité de la représentation, due à l'infinitude de cet ensemble. Ce problème peut être résolu en introduisant l'opérateur de fermeture de Kleene (noté $\star$). Cet opérateur caractérise la fermeture de l'opérateur de séquencement, c'est à dire l'ensemble des puissances d'une expression.

$$h^\star = \sum_{i=0}^{\infty} h^i$$

représente l'ensemble de toutes les puissances possibles de h, c'est à dire 0, 1 ou n occurrences de h. Notre contrainte s'écrit alors,

$$C_{min} = oh^3h^\star a$$

De même pour représenter un ensemble fini de séquences comportant un nombre croissant de h, on peut utiliser la représentation en somme $\sum$. Par exemple, la contrainte temporelle exprimant : "b est attendu au maximum cinq unités de temps après a", pouvant s'exprimer par l'ensemble de séquences

$$C = \{ab, ahb, ah^2b, ah^3b, ah^4b, ah^5b\}$$

sera représentée de manière par l'expression :

$$\sum_{i=0}^{5} ah^ib$$

L'ensemble des contraintes temporelles peuvent s'exprimer de façon plus générale sur la structure $S = (\sum_a, +, ., \star)$, où "+" représente l'aternative et "." le séquencement. Cette structure est aussi dénommée dans la terminologie de la théorie des langages : "ensemble des expressions régulières".

L'ensemble des expressions régulières représente des ensembles de mots construits sur un alphabet (noté $\sum_a$) et sur les opérateurs "+" et ".", il se définit récursivement de la façon suivante :

DÉFINITION 12 "EXPRESSION RÉGULIÈRE"

1) $\emptyset$ est une expression régulière et dénote l'ensemble vide
2) ϵ est une expression régulière et dénote le mot vide
3) $\forall a \in \sum_a$, a est une expression régulière
4) si r et s sont des expressions régulières alors :
$r + r$, rr, r^* sont des expressions régulières

On peut montrer par induction sur la taille d'une expression régulière, qu'il existe un automate fini (dont on a donné la définition au chapitre III) qui accepte le même langage. Cet ensemble a donc le même pouvoir d'expression que les automates.

En outre, cette classe d'expression est intéressante car il existe des algorithmes permettant de montrer des propriétés sur les langages que l'on peut transposer aux contraintes temporelles qui peuvent être exprimées à l'aide de tels langages :

- Montrer qu'une contrainte temporelle admet au moins une exécution possible.
- Montrer qu'une contrainte temporelle admet un ensemble infini de traces
- Montrer que deux contraintes temporelles, admettent le même ensemble de traces.

Ces algorithmes sont notamment basés sur le principe que tout automate admet un automate minimimal déterministe unique, à un isomorphisme près; voir [Hopcroft 1979]

Cependant, la notation qui nous intéresse est plus générale. En effet certaines contraintes temporelles ne peuvent pas être exprimées dans le cadre des expressions régulières.

Considérons la contrainte C_{dl} = "Après o, si n occurrences de a arrivent, alors le système doit attendre une délai de n unités de temps avant l'occurrence de b". Cette contrainte requiert un délai (compris entre le dernier a reçu et b) égal au nombre d'occurrences de a arrivés depuis le début de la séquence. Le délai n'est pas connu de manière statique mais dépend de l'exécution.

On peut montrer, à l'aide du lemme de "pompage" [Bar-Hillel *et al.* 1961], que cette contrainte ne peut être exprimée par une expression régulière (donc pas par un automate). Par contre, C_{dl} peut être représentée par une forme plus générale, celle de grammaire algébrique. Ce formalisme embrasse toute la complexité possible en terme de propriétés de séquences et sera capable de représenter toutes nos contraintes temporelles.

Si l'on a abordé un cas particulier de grammaire, à travers les expressions régulières et les automates finis, celui des grammaires linéaires à droite (ou à gauche), on donne maintenant la définition générale des grammaires que nous allons utiliser.

DÉFINITION 13 "GRAMMAIRE POUR LA REPRÉSENTATION DE CONTRAINTE TEMPORELLE"

Une grammaire temporelle est un quadruplet :

$$(\sum_a, NT, P, S)$$

où :
- $\sum_a$ est l'alphabet.
Une grammaire est un ensemble de règles syntaxiques, qui permettent de construire des mots, à partir d'un alphabet qui est l'ensemble des symboles apparaissant dans les séquences temporisées. Cet ensemble est donc constitué de l'ensemble des événements sur lesquels portent la contrainte, ainsi que de l'événement h.

- NT : l'ensemble de variables ou ensemble de symboles Non Terminaux, c'est à dire pouvant être substitués par une expression composée de terminaux et de non terminaux.
- P : ensemble de règles de productions, c'est à dire les règles de constructions syntaxiques, où une règle est une correspondance entre des symboles : $(\sum_a \cup NT)^* \rightarrow (\sum_a \cup NT)^*$.
- S : premier axiome à appliquer.

Exemple :
Pour la contrainte précédemment évoquée, on a la grammaire suivante :
$C_{dl} = (\sum_{dl}, NT_{dl}, P_{dl}, S)$ où :
- $\sum_{dl} = \{o, a, b, h\}$
- $NT_{dl} = \{S, T\}$
- P_{dl} est constitué des règles suivantes :

$$\begin{aligned} S &\rightarrow oTb \\ T &\rightarrow aTh + ah \end{aligned}$$

On peut comprendre cet ensemble de règles de production comme un programme informatique, où l'axiome S est le programme principal qui appelle la fonction récursive T dans le contexte "o a". La fonction T, ayant le choix (opérateur "+") entre les comportements aTh (appel récursif dans le contexte "a h"), ou bien ah qui marque l'arrêt de la contrainte. Le contexte "a h", mais aussi le "mot d'arrêt" "ah" permettent d'assurer un nombre égal de a et de h, comme l'exige la contrainte.

3.2.2. Résumé des concepts introduits

- Séquences temporisées : Ceux sont des traces d'exécutions où le temps apparaît sous la forme d'un événement spécifique : "h".
- Contraintes temporelles : On peut définir une contrainte temporelle par l'ensemble des séquences temporisées qui respectent cette contrainte.
- Reconnaissance d'une séquence temporisée par une CT : Une séquence est reconnue ou respecte une contrainte si elle appartient à l'ensemble des séquences temporisées qui identifie la contrainte. A ce moment de définition des concepts, on s'est aperçu que la notation ensembliste était trop limitée notament à cause des ensembles infinis. On a alors introduit la notion d'expressions régulières qui permet de représenter des ensembles infinis.
- Expression régulière : une expression régulière, est une expression batie à l'aide des opérateurs de choix et de séquence et qui considère la fermeture de ces opérateurs. Les expressions régulières correspondent à des cas particuliers de grammaire : les grammaires régulières. Cependant, toutes les contraintes temporelles ne peuvent pas s'exprimer en terme de grammaire régulière.
- Un langage est un ensemble des mots, dans notre approche, de séquences temporisées. Une contrainte temporelle constitue donc un langage. Un langage est engendré par une grammaire. Tout mot du langage peut être reconnu par une grammaire.

- Grammaire : Une grammaire est la formalisation la plus générale pour représenter les contraintes temporelles. Elle embrasse toute les types de contraintes temporelles.
- Reconnaissance d'un mot par une grammaire : c'est un processus qui partant d'un séquence permet par une suite de substitutions de retrouver l'axiome (S).

  Exemple :
  Si l'on considère la grammaire précédente et la séquence *oaaa***ah***hhhb*, *ah* peut être remplacé par T, en vertue de la règle T.
  donc *oaaaahhhhb* peut se réécrire en *oaaa***T***hhhb*
  Dans *oaa***Th***hhb*, *aTh* peut se réécrire en T
  Donc la séquence se transforme en *oaa***T***hhb*
  En appliquant deux fois ce principe de réécriture, on obtient la nouvelle séquence : *oTb* qui peut se réécrire en S. Ce qui nous permet de conclure :
  - Cette séquence temporisée appartient à la contrainte temporelle
  - On l'a reconnue par les règles : TTTTS.

4. Classification des contraintes

Peu de travaux sur la spécification des systèmes temps réel présentent une classification des contraintes temporelles. Parmi ceux-la, on va évoquer celui de Dassarathy, qui a été un des travaux à la base de notre recherche. Nous proposons ensuite notre propre type de classification, par rapport à l'approche originale que nous avons développée.

4.1. Base de l'approche

Comme nous l'avions évoqué au premier chapitre, les contraintes temporelles simples mettent en jeu deux événements : un marquant l'origine relative des temps et le début de la prise en compte de la contrainte, tandis que l'autre marque marque l'achèvement de la contrainte.

Exemple :
"Après avoir Décroché le combiné (événement D), l'opérateur doit Composer son numéro (événement C), au plus tard 20 secondes après"
Dans cette contrainte, D marque l'origine du temps par rapport à laquelle est exprimée la durée, et C l'événement attendu.

Dassarathy évoque dans [Dasarathy 1985] une typologie de contraintes selon le type d'événements : Stimulus (événement produit par l'environnement) ou Réaction (action du système : action interne ou réaction vers l'environnement). En outre, il classe les contraintes en trois types :

- Les contraintes de type minimum, c'est à dire celles basées sur des délais minimum.
- Les contraintes de type maximum, c'est à dire celles qui imposent un délai à ne pas dépasser.
- Les contraintes d'intervalle. Ici, Dassarathy considère que les événements peuvent "durer", et que l'on peut donc poser des contraintes sur la durée de leur occurrence.

4.1.1. Contrainte maximum

On analyse toutes les combinaisons Stimulus (S), Réaction (R).

- S-S-max : Un temps maximum est accordé entre les occurrences de deux stimuli.
  Exemple :
  Après le premier chiffre, le second chiffre doit être composé en moins de 5 secondes.

  C'est une contrainte posée sur le comportement attendu de l'environnement. Ce n'est pas une contrainte qui définit le comportement du système, mais elle implique que le système prévoit les alternatives où l'environnement répond de façon attendue ou non. Si le système se veut robuste, il devra prévoir une marche à suivre en cas de violation de contrainte. De plus, lors du test du système, cette contrainte peut indiquer un ensemble de scénarios à tester.
- S-R-max : Un temps maximum est accordé entre un stimulus et la réponse du système.
  Exemple :
  Après la composition du numéro, le système téléphonique doit renvoyer à l'appelant un compte rendu de recherche, en au plus 1 minute.

  C'est une contrainte sur le système qu'on pourrait qualifier de contrainte de performance. Cependant ce terme est inapproprié car la durée imposée n'est pas forcément incompatible avec la future puissance du système. Par contre, on peut la qualifier de "sporadiqué'. Lorsqu'un stimulus se produit, la contrainte doit alors être vérifiée.
- R-S-max : Une durée maximum est accordée entre une réaction du système et une action de l'environnement. Là encore, on peut faire la même remarque que pour S-S-max. C'est une contrainte sur l'environnement, qui permet en fait au système, d'avoir une image, un modèle du comportement de l'environnement.
- R-R-max : Un temps maximum est alloué entre deux réactions du système. C'est aussi une contrainte sur le comportement du système comme S-R-max.

Comme on l'a évoqué, S-S-max et R-S-max peuvent servir à déterminer des scénarios de test. Par contre, elles ne nous intéressent pas en terme de contraintes temporelles car elles ne portent pas sur le système. Seuls les deux autres cas seront des contraintes à prendre en compte dans les spécifications.

4.1.2. Contrainte minimum

- S-S-min : une durée minimum est attendu entre deux stimuli. Les remarques que nous avons faites pour S-S-max, sont également valables pour cette contrainte. Il s'agit d'une contrainte qui indique au système comment devrait agir l'environnement, et elle contribue à établir un modèle de l'environnement.
  Elle peut aussi indiquer une capacité de saisie des stimuli, et donner cette fois-ci des indications sur la performance requise.
  Exemple :
  Un délai minimum de 0.5 seconde doit s'écouler entre la composition de deux chiffres.

- S-R-min : Un temps minimum est requis entre un stimulus et la réaction du système. Il s'agit, comme dans le cas maximum, de contrainte comportementale. Le système doit "laisser" s'écouler un délai avant de réagir. Comme cette contrainte ne pose pas de contrainte sur la date précise de réaction, le système réagit de façon non déterministe après l'écoulement du délai (la réaction pouvant alors intervenir à tout moment).

  Exemple :
  Après avoir composé le numéro de téléphone, l'usager ne peut être servi avant 15 secondes (temps de recherche minimal d'un abonné). Par contre, on peut (en première approximation) ne pas connaître la borne supérieure de ce délai, qui peut par exemple dépendre de la charge du réseau.

- R-S-min : Un temps minimum entre une réaction et un stimulus. C'est une contrainte portant sur l'environnement, et qui peut cependant s'interpréter comme un taux de service du système. Si S est une nouvelle entrée, et R la dernière réponse à l'avant-dernier stimulus, alors cette contrainte exprime qu'une nouvelle entrée ne peut être admise avant une certaine limite de temps.
- R-R-min : Un temps minimum entre deux réactions du système.

On ne tiendra pas compte de S-S-min, ni de R-S-min. Par contre, cette dernière contrainte peut, dans certains cas, indiquer à travers un délai minorant l'intervalle séparant deux stimuli indiquant ainsi la performance future que devra avoir le système.

S-R-min et R-R-min sont les seules contraintes qui portent sur le système. Il s'agit dans ces cas de s'assurer que le système ne va pas réagir trop vite.

4.1.3. Contrainte de durée

Dassarathy arguant qu'il est parfois nécessaire de moduler la durée d'un signal, introduit des contraintes de durée sur des événements. Il introduit alors des "stimuli continus". Ce type de contrainte sort du contexte "système de transitions", et nous ne les prendrons pas en compte dans notre travail.

Exemple :
Appuyer sur un bouton pendant au moins 15 secondes mais pas plus de 30 secondes.

Cette classification nous semble un peu simple, car elle est loin de présenter l'ensemble des contraintes temporelles que l'on peut trouver dans un cahier des charges. Nous allons présenter maintenant une classification théorique des contraintes, selon le type de complexité portant sur les durées.

4.2. Classification des contraintes temporelles

Nous avons évoqué une contrainte temporelle C_{dl} qui n'était pas de type régulier et cette distinction nous a amené à établir un autre type de classification. En effet, la durée est ici variable, contrairement aux premiers exemples où elle était constante.

On va donc classer les contraintes temporelles suivant le type de fonction agissant sur la durée de la contrainte : constante, linéaire ou polynomiale.

Nous définissons ainsi **les contraintes régulières**, qui sont des contraintes dont la grammaire supporte une restriction sur la construction de ses règles de production : elles doivent être linéaires à droite, c'est à dire du type $A \rightarrow \alpha B$ où $\alpha \in \sum^*$. Cet ensemble identifie les contraintes qui imposent, sur les événements, des durées de type constant (constante finie ou infinie).

Exemple :
Soit la contrainte exprimant : "a est attendu au plus tôt 3 occurrences de temps après o et au plus tard 5 unités de temps après o". Cette contrainte peut alors être représentée par la grammaire temporelle suivante :

$$S \rightarrow o \sum_{i=3}^{5} h^i a$$

Le deuxième type de contrainte que nous définissons sont **les contraintes linéaires.** Ces contraintes proposent un niveau de difficulté supérieur aux précédentes. Elles permettent en effet de poser des durées de longueur variable, selon des fonctions linéaires du nombre d'occurrences d'autres événements. Ainsi, par exemple C_{dl}, présente une durée dont la valeur est une fonction (fonction identité) du nombre d'événements a arrivés précédemment. D'un point de vue notation, ces contraintes correspondent à un type de grammaire particulier : les grammaires indépendantes du contexte, dont les règles de production ont le schéma suivant : $A \rightarrow \alpha B \beta$ où $\alpha, \beta \in \sum^*$ et $A, B \in NT$

Exemple :
Soit la contrainte exprimant : "Après o, si n occurrences de a arrivent, alors le système doit attendre une délai de $2n + 3$ unités de temps avant l'occurrence de b".
Cette contrainte peut alors être représentée par la grammaire temporelle suivante :

$$\begin{aligned} S &\rightarrow oTb \\ T &\rightarrow aThh + ahhhhh \end{aligned}$$

Le troisième type de contrainte que nous définissons est celui **des contraintes temporelles polynomiales.** Il correspond à des contraintes où les durées sont d'ordre polynomial. Ce genre de contrainte correspond à des grammaires de type non restreint, c'est à dire, où il n'y a aucune restriction sur les règles de production : $\alpha \rightarrow \beta$ où $\alpha, \beta \in \sum^*$

Exemple :
Soit la contrainte exprimant : "Après o, le système doit attendre un délai de 2^n unités de temps avant l'occurrence de b". Cette contrainte peut alors être représentée par l'expression suivante : $L = \{o\, h^{2^n}\, b / n \in N\}$ peut être formalisée par la grammaire : (S, S_1, A, B, C, D, E sont des non terminaux)

$$S \rightarrow o\, S_1\, b \quad (1)$$
$$S_1 \rightarrow A\, C\, h\, B \quad (2)$$

$$C\,h \rightarrow h\,h\,C \quad (3)$$
$$C\,B \rightarrow D\,B \quad (4)$$
$$C\,B \rightarrow E \quad (5)$$
$$h\,D \rightarrow D\,h \quad (6)$$
$$A\,D \rightarrow A\,C \quad (7)$$
$$h\,E \rightarrow E\,h \quad (8)$$
$$A\,E \rightarrow \epsilon \quad (9)$$

Indications pour la compréhension de la grammaire

1. Le non terminal S_1 sert à débuter la contrainte
2. Les non terminaux A et B marquent les débuts et fins de chaînes.
3. La règle (3) multiplie par deux le nombre de h.
4. La règle (4) permet d'augmenter la puissance n de h.
5. La règle (5) arrête le processus.
6. Le non terminal D marque le passage de n à $n+1$
7. La règle (8) permet de déplacer E vers le symbole de début de chaîne A.
8. La règle (9) marque la fin (ϵ : mot vide).

Notre classification des contraintes temporelles selon leur complexité, se calque sur celle de Chomsky ([Chomsky 1956], [Chomsky 1959]) pour les grammaires (excepté que nous n'avons pas trouvé de contraintes temporelles "utiles", différenciant la classe "dépendantes du contexte" de la famille des "sans restriction"). Cette classification, outre son intérêt descriptif, indique les formalismes adéquats pour la représentation des différents types de contraintes.

Justification de cette approche

On peut résumer l'apport de notre approche en quelques points :

- La notion de traces temporisées permet de proposer une représention homogène des événements et du temps.
- La notion de grammaire embrasse toute la complexité de contraintes possibles.
- Cette représentation du temps permet de traiter le temps de manière statique, sans avoir à utiliser des méthodes stochastiques (réseaux de Petri), ou sans introduire la notion d'urgence ou de devoir (Lotos), et sans utiliser d'horloge interne au système (Statecharts).
- Le traitement du temps est réduit à une analyse syntaxique par grammaires.

5. Extraction de contraintes temporelles

La typologie de Dassarathy fait la distinction entre stimulus et réaction. Cette distinction est utile, dans la mesure où elle met en lumière que certaines combinaisons ne relèvent pas de la

spécification. Lors de l'extraction, on peut donc écarter toute contrainte portant sur S (R-S-max, S-S-max, S-S-min), sauf celles de type R-S-min, lorsqu'elles servent à indiquer un taux de service maximum.

Le processus d'extraction est complexe, car les contraintes sont souvent imbriquées les unes dans les autres. Il faut donc proposer une méthode d'analyse qui procède par raffinements successifs. La décomposition que nous proposons n'est pas la seule possible, mais elle dégage cependant, une structuration simple qui met à jour des liens entre les contraintes par niveau de détail croissant.

Au plus haut niveau, on retrouve généralement un ensemble de contraintes qui forme un ensemble conjonctif de contraintes : au premier examem du cahier des charges, on relève un ensemble de points qui doivent être tous respectés. L'examen de chacun de ces points fait alors apparaître des contraintes sporadiques ou périodiques. En affinant encore, on tombe sur des contraintes régulières, linéaires ou polynomiales. S'agissant des contraintes régulières, on peut ensuite les décomposer en minimum ou maximum. Cette décomposition peut mieux se résumer par la grammaire suivante :

$$
\begin{aligned}
< Cahier\ des\ charges > \rightarrow\ & < etCT > \wedge < etCF > \\
< etCT > \rightarrow\ & < CT > \wedge\ < CT > [\wedge\ < CT >] \\
< ouCT > \rightarrow\ & < CT > \vee < CT > [\vee < CT >] \\
< CT > \rightarrow\ & < etCT > + < ouCT > \\
& +PER < CT > +SPOR < CS > + < CTbas > \\
< CTbas > \rightarrow\ & < CTreg > + < CTlin > + < CTpol > \\
< CTreg > \rightarrow\ & < etCTregbas > + < ouCTregbas > \\
< etCTregbas > \rightarrow\ & < CTregbas > \wedge\ < CTregbas > [\wedge\ < CTregbas >] \\
< ouCTregbas > \rightarrow\ & < CTregbas > \vee < CTregbas > [\vee < CTregbas >] \\
< CTregbas > \rightarrow\ & < CTmin > \vee < CTmax >
\end{aligned}
$$

- $< etCF >$ représente l'ensemble des contraintes fonctionnelles (contraintes qui ne relevent pas du temps : exclusion mutuelle, évitement d'interblocage, ...). Comme nous l'avons expliqué dans la présentation générale de notre approche (voir figure IV-2) nous ne nous intéressons qu'a la prise en compte des contraintes temporelles, aussi ces contraintes ne rentrent donc pas dans le cadre de notre étude.
- $< etCT >$ Au premier niveau d'examen, la partie temporelle d'un cahier des charges peut être analysée comme un ensemble conjonctif de contraintes temporelles : il faut respecter toutes les contraintes temporelles. Dans les niveaux inférieurs on va dégager d'autres types de liaison (disjonctif, ...) entre les contraintes.

- $< CT >$ est une contrainte temporelle de haut niveau, qui peut être soit une conjonction de contraintes, une disjonction de contraintes, une contrainte périodique, une contrainte sporadique, ou une contrainte de base.
- $< ouCT >$ est une disjonction de contrainte de haut niveau.
- $< CTbas >$ est une contrainte de "base, de haut niveau", c'est à dire régulière, linéaire ou polynomiale.
- $< CTreg >$ est une contrainte temporelle régulière, qui est un ensemble conjonctif ou disjonctif de contraintes régulières.
- $< etCTregbas >$ est un ensemble conjonctif de contraintes.
- $< ouCTregbas >$ est un ensemble disjonctif de contraintes.
- $< CTregbas >$ est une contrainte soit minimum, soit maximum.

Nous avons restreint cette analyse aux contraintes temporelles régulières car l'ampleur de ce travail nous semblait trop importante. Il aurait été alors nécessaire de développer $< CTlin >$ et $< CTpol >$ comme on l'a fait pour $< CTreg >$.

Cette décomposition fait apparaître plusieurs opérateurs de liaison possibles entre les contraintes :

- $\bigvee$ Opérateur de disjonction
- $\bigwedge$ Opérateur de conjonction
- PER Opérateur périodique
- $SPOR$ Opérateur sporadique

Après l'extraction qui est faite selon cette décomposition, on obtiendra des contraintes de base. Celles-ci vont alors être exprimées, de manière formelle, sous la forme de grammaires types. Notre objectif est alors, de recomposer l'ensemble de ces contraintes par les opérateurs mis à jour par la première analyse, pour retrouver une expression générale et formelle du comportement temporel du cahier des charges.

Pour réaliser cet objectif il nous faut définir l'ensemble des contraintes temporelles de bases et définir l'ensemble des opérateurs de liaison entre les contraintes. Comme nous avons restreint notre travail aux contraintes temporelles régulières, on ne va pas définir de contraintes de base pour les contraintes linéaires et polynomiales, et on ne définira pas non plus l'effet des opérateurs sur ces dernières.

5.1. Grammaires types des contraintes de base

Les contraintes temporelles régulières de base lient deux événements par une durée selon deux types, minimum ou maximum, où le premier événement définit l'origine des temps pour la durée, et le second événement est celui sur lequel porte la contrainte. Elles peuvent donc être simplement définies par la donnée du quadruplet : *(origine, attendu, duree, type)*

Grammaire d'une contrainte minimum $:= (o, a, n, minimum)$

$$\begin{aligned} S &\rightarrow oh^n T \\ T &\rightarrow hT + a \end{aligned}$$

Exemple :
"Après avoir Décroché le combiné téléphonique, ne pas Composer le numéro avant 3 secondes". L'analyse de cette contrainte donne le quadruplet $(D, C, 3, min)$ qui se représente alors par la grammaire :

$$\begin{aligned} S &\rightarrow Dh^3T \\ T &\rightarrow hT + C \end{aligned}$$

Grammaire d'une contrainte maximum $:= (o, a, n, maximum)$

$$S \rightarrow o\sum_{i=0}^{n} h^i a$$

Exemple :
"Après avoir Décroché le combiné téléphonique, Composer le numéro avant 8 secondes". L'analyse de cette contrainte donne le quadruplet $(D, C, 8, max)$ qui se représente alors par la grammaire :

$$S \rightarrow D\sum_{i=0}^{8} h^i C$$

5.2. Opérateurs

Ces opérateurs (binaires ou unaires) s'appliquent de l'ensemble des grammaires vers l'ensemble des grammaires. Pour définir certains de ces opérateurs, on pourra donner la grammaire résultante. Par contre pour l'opérateur de conjonction on sera obligé de le définir par induction.

5.2.1. Opérateur périodique : PER

La façon la plus simple de définir un comportement périodique, est de se donner une période et un événement marquant la fin de la périodicité. Soit CI une contrainte temporelle qui doit être vérifiée toutes les n unités de temps, jusqu'à ce que le programme se termine par l'événement fin. L'expression de cette contrainte s'exprime alors sous la forme :

Opérateur périodique $:= (n, fin, CI)$

$$S \rightarrow \sum_{i=0}^{n} h^i CIh^{n-i}S + fin$$

Exemple :
"Toutes les 5 secondes, vérifier que si quelqu'un Décroche son combiné, il aura commencé à numéroter avant 3 secondes".

Dans cet exemple on a une contrainte initiale CI, liant l'événement C (Commencer à numéroter) à l'événement D (quelqu'un Décroche). Cette contrainte est de type maximum d'ou l'expression de CI :

$$CI \quad - \quad D\sum_{i=0}^{3} h^i C$$

L'exemple impose d'appliquer un opérateur de périodicité sur CI. La nouvelle grammaire $PER\,CI$ s'exprime sous la forme :

$$S \quad \rightarrow \quad \sum_{i=0}^{5} h^5 CI h^{5-i} S + fin$$
$$CI \quad \rightarrow \quad D\sum_{i=0}^{3} h^i C$$

5.2.2. Opérateur de sporadicité : $SPOR$

La sporadicité est l'apparition d'une condition ou d'un événement déclenchant la prise en compte d'une contrainte temporelle. Elle se définit simplement par la donnée d'un événement et d'une contrainte, et s'écrit sous la forme :

Opérateur sporadique $:= (o, CI)$

$$S \quad \rightarrow \quad oCI + hS$$

Note : le membre hS permet d'attendre l'événement o qui déclenche la contrainte sporadique.

Exemple :
"Dès que quelqu'un Décroche son combiné, vérifier que, aprés avoir composé le premier chiffre il aura composé le second avant 3 secondes".

On a une contrainte initiale CI qui lie l'événement 1 : composer le 1^{ier} numéro, à l'événement 2 : composer le 2^{nd} numéro. Cette contrainte est de type maximum d'ou l'expression de CI :

$$CI \quad \rightarrow 1 \sum_{i=0}^{3} h^i 2$$

L'exemple impose d'appliquer un opérateur de sporadicité sur CM. La nouvelle grammaire $SPOR\,CM$ s'exprime sous la forme :

$$S \quad \rightarrow D\,CI + hS$$
$$CI \quad \rightarrow 1 \sum_{i=0}^{3} h^i 2$$

5.2.3. Opérateur de disjonction : $\vee$

Cet opérateur est un opérateur binaire qui exprime que deux contraintes temporelles CT_1 et CT_2 sont alternatives. En terme de grammaire, il s'écrit simplement grâce à l'opérateur + :

Opérateur de disjonction := (o, CT_1, CT_2)

$$S \rightarrow CT_1 + CT_2$$

Exemple :
"Le système central, identifie une communication "Humaine" par le signal D, ou une communication de type "Fax" par le signal S_1. S'il s'agit d'une communication Humaine, l'appelant doit composer son premier chiffre (événement C) avant 3 secondes. S'il s'agit d'un Fax, le système doit recevoir le signal S_2, indiquant le début de transmission de données avant une seconde. Il y a deux contraintes initiales : Fax et H.

Fax lie l'événement S_1 à l'événement S_2 par une contrainte maximum :

$$Fax \rightarrow S_1 \sum_{i=0}^{1} h^i S_2$$

Fax lie l'événement S_1 à l'événement S_2 par une contrainte maximum :

$$Fax \rightarrow S_1 \sum_{i=0}^{1} h^i S_2$$

Après l'aplication de l'opérateur de disjonction on obtient $Fax + H$:

$$\begin{aligned} S &\rightarrow H + Fax \\ H &\rightarrow D \sum_{i=0}^{3} h^i C \\ Fax &\rightarrow S_1 \sum_{i=0}^{1} h^i S_2 \end{aligned}$$

5.2.4. Opérateur de conjonction : $\wedge$

Cet opérateur nécessite une analyse plus approfondie. En effet, on a besoin de garantir la cohérence entre les contraintes : deux contraintes peuvent être incompatibles. Cette définition se fait en énumérant tous les cas possibles de conjonction (on pourra employer aussi le terme de composition) entre des contraintes de base : minimum et maximum. Elle présente une combinatoire importante (type et type de dépendances) que nous allons étudier dans le chapitre suivant.

Conclusion

Dans ce chapitre, on a décrit la représentation du temps que nous avons adoptée, ainsi que la représentation des contraintes temporelles qui en découle. Nous avons ensuite présenté, en plusieurs étapes, le contexte général de l'approche que nous proposons :

- L'extraction des contraintes temporelles suivant une décomposition structurée.
- La recomposition de ces contraintes par un ensemble d'opérateurs : périodicité, sporadicité, disjonction et conjonction. Le résultat de cette recomposition devant produire un analyseur syntaxique.
- Le test par l'analyseur de traces temporisées provenant des méthodes classiques de spécification.

Nous proposons, pour la phase d'extraction une analyse descendante et structurée des contraintes temporelles, qui nous permet d'identifier les liens unissant les contraintes : périodicité, sporadicité, disjonction, conjonction. Le raffinage des contraintes temporelles se fait jusqu'à ce que l'on appelle des contraintes temporelles de base, pour lesquelles on a défini une représentation formelle.

La phase de recomposition, se base sur les liens mis en évidence dans l'analyse. Nous avons redéfini ces liens en opérateurs par rapport au formalisme adopté. L'opérateur de conjonction fait l'objet du chapitre suivant.

CHAPITRE V

L'OPÉRATEUR DE CONJONCTION : COMPOSITION DE CONTRAINTES TEMPORELLES

1. Introduction

On ne s'intéresse, comme on l'a mentionné, qu'aux contraintes régulières qui sont celles qui sont obtenues par conjonction ou par disjonction de contraintes. On peut remarquer que toute combinaison de conjonction et de disjonction bien parenthésée peut se développer en somme de produit :

$((C_1 \wedge (C_2 \vee C_3)) \vee C_4) \wedge C_5 \equiv (C_1 \wedge C_2 \wedge C_5) \vee (C_1 \wedge C_3 \wedge C_5) \vee (C_4 \wedge C_5)$

Cela permet donc de séparer deux étapes : la conjonction et la disjonction de contrainte. L'opérateur de conjonction n'a donc besoin d'être défini que sur des formes conjonctives. La définition que nous allons donner comporte deux étapes :

- 1. $CTB_i \wedge CTB_j$ où CTB_i et CTB_j sont deux contraintes de bases
- 2. $CT_n \wedge CTB_i$ où CTB_i est une contrainte de base et CT_n une contrainte régulière obtenue par composition de n contraintes de bases.

Exemple :
"A doit donner un coup de téléphone cet après-midi avant trois heures, de plus, A ne peux quitter son travail avant cinq heures."

La phase d'extraction de contraintes relève deux contraintes temporelles une de type minimum et l'autre de type maximum. Les durées exprimées dans ces contraintes, sont relatives à une origine des temps implicite, qui est le début de l'après-midi, que l'on notera *da*. On note t et q les événements marquant le fait de téléphonner et le fait de quitter le bureau. Les contraintes sont alors :
$C_1 = (\mathbf{da}, \mathbf{t}, 3, max)$
$C_2 = (\mathbf{da}, \mathbf{q}, 5, min)$
Pour C_1 on a la grammaire :

$$S \rightarrow \mathbf{da}(\sum_{i=0}^{3} h^i \mathbf{t})$$

Pour C_2 on a la grammaire :

$$\begin{aligned} S &\rightarrow \mathbf{da}h^5T \\ T &\rightarrow hT + \mathbf{q} \end{aligned}$$

Ces deux contraintes ayant la même origine des temps, elles ne sont pas indépendantes. La grammaire acceptant des mots qui respectent ces deux contraintes est :

$$\begin{aligned} S &\rightarrow \mathbf{da}(\sum_{i=0}^{3} h^i t h^{5-i})T \\ T &\rightarrow hT + \mathbf{q} \end{aligned}$$

En effet, on peut vérifie qu'elle accepte les mots :

$$\mathbf{da}\ h\ h\ \mathbf{t}\ h\ h\ h\ \mathbf{q}$$

$$\mathbf{da}\ \mathbf{t}\ h\ h\ h\ h\ h\ \mathbf{q}$$

$$\ldots$$

Par contre elle ne reconnaît pas les mots :

$$\mathbf{da}\ h\ h\ h\ h\ \mathbf{t}\ h\ \mathbf{q}$$

$$\mathbf{da}\ \mathbf{t}\ h\ h\ h\ h\ \mathbf{q}$$

Notre objectif est par la suite de présenter l'algorithme général permettant d'obtenir toutes ces formes composées.

2. Conjonction de contraintes de base

Croiser deux contraintes de base nécessite au préalable d'analyser les dépendances qui peuvent exister entre elles. En effet, si l'on considère le cas le plus trivial de conjonction, c'est à dire lorsque les contraintes portent sur le même événement et qu'elles ont la même origine des temps, la conjonction peut alors s'interpréter comme l'intersection des ensembles des traces correspondant aux deux contraintes. On aura donc une grammaire résultante comportant un ensemble plus restreint de mots. Cependant, dans d'autres cas de dépendances, comme dans l'exemple précédent, le résultat de la conjonction ne réduit pas le nombre de traces, mais impose cependant sur les traces résultant de la composition de nouvelles règles de construction syntaxique.

Remarque 5 : La composition, si elle ne réduit pas dans tous les cas le nombre de traces possibles, produit en revanche des règles de constructions plus restrictives sur la forme des traces réduisant ainsi leur degré de liberté.

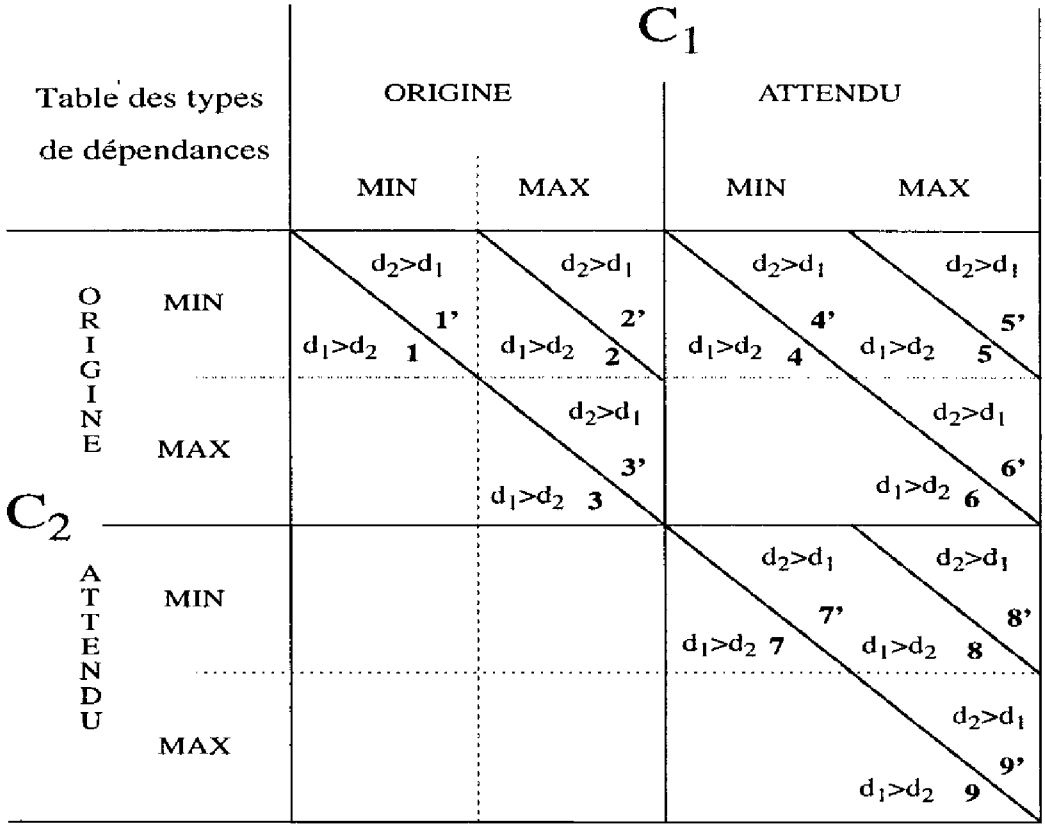
Soient les contraintes $C_1 = (o_1, a_1, d_1, T)$ et $C_2 = (o_2, a_2, d_2, T)$. Les dépendances dont de quatre types :

- $o_1 = o_2 \wedge a_1 \neq a_2$: les contraintes ont la même origine des temps et portent sur des événements différents.
- $a_1 = a_2 \wedge o_1 \neq o_2$: les contraintes portent sur le même événement, mais à partir de deux origines temporelles différentes, dont on ne connaît pas à priori, l'ordre relatif.
- $a_1 = o_2 \wedge o_1 \neq a_2$ et son dual $a_2 = o_1 \wedge o_2 \neq a_1$: les contraintes sont consécutives, l'une déclenche l'autre.
- $o_1 = o_2 \wedge a_1 = a_2$: les contraintes ont une même origine des temps et elles portent sur le même événement, c'est le cas le plus facile à résoudre.

Une autre combinatoire est introduite par le type des contraintes ainsi que l'ordre relatif des durées. L'ensemble des cas que nous allons étudier est présenté dans le schéma V-1.

Commentaires sur la figure V-1

- Cette figure décrit tous les cas de dépendances entre une contrainte $C_1 = (o_1, a_1, d_1, T_1)$ et une contrainte $C_2 = (o_2, a_2, d_2, T_2)$. En ligne on a l'inventaire des cas possibles sur la contrainte C_1, en colonne sur la contrainte C_2.
- Les cases sont numérotées, celles qui apparaîssent vides correspondent (cases duales) à des cases déjà numérotées : comme par exemple la case blanche qui se trouve sous la case $(1 - 1')$ trouve son équivalent dans la case $(2 - 2')$.
  En effet :
  $C_1 = (o_1, a_1, d_1, minimum) \bigwedge C_2 = (o_1, a_2, d_2, maximum)$
  est le cas dual de
  $C_1 = (o_1, a_1, d_1, maximum) \bigwedge C_2 = (o_1, a_2, d_2, minimum)$
- Les intersections lignes-colonnes indiquent les cas de dépendances : dans la case $(5 - 5')$, l'attendu de C_1 est égal à l'origine de C_2, C_1 est une contrainte de type maximum et C_2 une contrainte de type minimum.



10 : Même attendu même origine

Figure V-1 : Ensemble des cas de dépendances entre les contraintes temporelles

- Dans les cas 4,5 et 6 il s'agit de contraintes contigües, où l'événement attendu de l'une déclenche l'autre.
- Enfin dans les cas 7,8 et 9 il s'agit d'un événement qui est lié à deux origines dont on ne connaît pas l'ordre à priori.

On va donner dans chaque cas un exemple, puis on établira l'arbre des traces valides par rapport aux deux contraintes, qui nous permettra dans un premier temps d'obtenir un ensemble de formules développées. Dans un deuxième temps, à partir de ce dernier ensemble, on dérivera un ensemble de formules plus compact qui nous permettra enfin, de trouver des formules générales.

2.1. Même événement origine

Un événement marquant une origine temporelle peut être produit par l'environnement (grandeur atteignant une valeur de consigne, action sur des capteurs, ...) ou par le système (expiration d'un délai, date de mise en route du système).

Dans les cas qui correspondent aux parties 1, 2 et 3 de (fig. V-1), on décrira le cas de contraintes minimum ou maximum qui sont dépendantes par l'origine. Ce sont typiquement des contraintes liant deux événements par des dates exprimées par rapport à la même référence temporelle.

2.1.1. Même type de contrainte

- Minimum et $d_1 > d_2$
  Soient deux contraintes C_1 et C_2 de type minimum et de même origine temporelle :

$$C_1 = (o_1, a_1, 5, min)$$

$$C_2 = (o_1, a_2, 3, min)$$

On peut représenter l'ensemble des séquences qui respectent la contrainte par un graphe orienté dont les arcs représentent les occurrences des événements impliqués dans les contraintes. On choisit pour faciliter la compréhension, de représenter l'écoulement du temps par un déplacement en diagonale de gauche à droite, tandis que les autres événements provoquent des déplacements dans les autres directions (diagonalement de droite à gauche, horizontalement ou verticalement). Le fait que plusieurs transitions partent d'un même nœud indique que plusieurs alternatives sont possibles à partir du prefixe constitué par la séquence des événements qui se sont déroulés depuis l'origine . Les boucles indiquent que le système est en état d'attente d'un événement. Ainsi, la figure V-2 représente le graphe des séquences "compatibles" avec C_1 et C_2 (fig. V-2).
Dans cette figure, on note trois boucles sur h qui permettent "d'attendre a_1 ou a_2". Ces boucles sont caractéristiques des contraintes minimum. Une façon de formaliser cet arbre est d'attacher à chaque nœud un symbole non terminal. Pour chaque non terminal, on donne une règle de dérivation qui correspond aux arcs sortant du nœud.

Remarque 6 : De manière à permettre plus facilement la compréhension des formules, on utilise des indices pour les symboles non terminaux qui représentent le temps écoulé, depuis le début de la prise en compte de la contrainte. Ainsi, U_2 signifie que deux unités de temps se sont écoulés (et que a_2 s'est

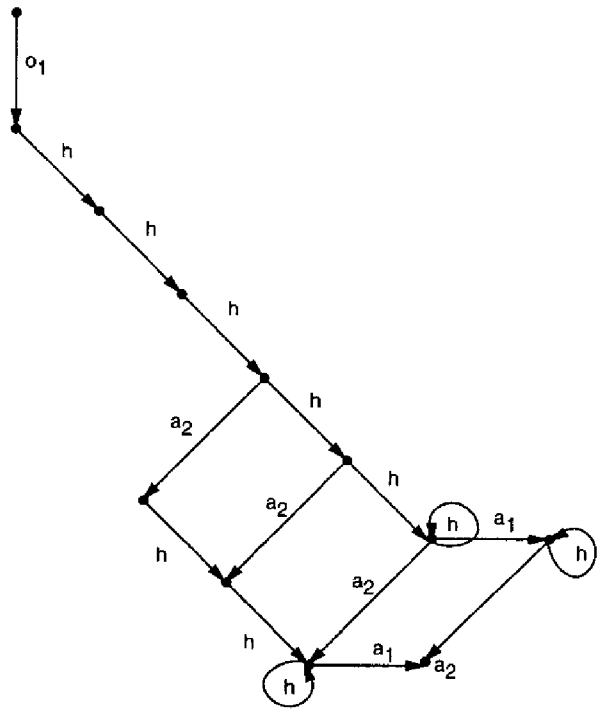


Figure V-2 : origine identiques, min-min

produit).

$$
\begin{array}{lll}
S \rightarrow o_1 h^3 T & & \\
T \rightarrow hT_1 + a_2 U & U \rightarrow hU_1 & V \rightarrow hV + a_2 \\
T_1 \rightarrow hT_2 + a_2 U_1 & U_1 \rightarrow hU_2 & \\
T_2 \rightarrow hT_2 + a_2 U_2 + a_1 V & U_2 \rightarrow hU_2 + a_1 &
\end{array}
$$

On peut réduire ces formules, en remplaçant chaque fois que cela est possible, un non terminal par sa dérivation. Cependant on sera obligé de garder les dérivations récursives, c'est à dire, celles comportant dans la partie droite de la règle, une occurrence du non terminal de la partie gauche. On va s'attacher, en réduisant, à faire ressortir la structuration de ces traces, c'est à dire en faisant apparaître des sommes ou des factorisations qui permettront de généraliser ces formules, et d'en dégager des algorithmes qui permettront de les implémenter.
Ainsi en réduisant les formules ci-dessus on relève la forme $h^i a_2 h^{2-i}$ qui indique un entrelacement de a_2 avec la séquence h^2 :

$$
\begin{array}{ll}
S \rightarrow o_1 h^3 (h^2 T_2 + h a_2 h U_2 + a_2 h^2 U_2) & \\
U_2 \rightarrow hU_2 + a_1 & \text{attente de } a_1 \\
T_2 \rightarrow hT_2 + a_2 U_2 + a_1 V_0 & \text{attente de } a_1 \text{ et } a_2 \\
V_0 \rightarrow hV_0 + a_2 &
\end{array}
$$

On peut maintenant généraliser, c'est à dire donner les formules pour des valeurs quel-

conques de d_1 et de d_2.

$$\begin{aligned} S &\rightarrow o_1.h^{d_2}[h^{d_1-d_2}.T + (\sum_{i=0}^{d_1-d_2} h^i.a_2.h^{d_1-d_2-i})V] \\ T &\rightarrow a_1.U + a_2.V + h.T \\ U &\rightarrow h.U + a_2 \\ V &\rightarrow h.V + a_1 \end{aligned}$$

Pour $d_1 < d_2$ on obtient la formule symétrique, de plus, la formule initiale reste valable pour $d_1 = d_2$.

- Maximum
Soient deux contraintes C_1 et C_2 de type maximum et de même origine temporelle :

$$C_1 = (o_1, a_1, 5, max)$$

$$C_2 = (o_1, a_2, 3, max)$$

La figure des traces compatibles avec C_1 et C_2 est donnée dans le schéma (fig. V-3) : L'analyse de cette la figure V-3 nous donne les équations :

$$\begin{array}{llll} S \rightarrow o_1 T & & & \\ T \rightarrow hT_1 + a_1 U + a_2 V & U \rightarrow a_2 + hU_1 & V \rightarrow a_1 + hV_1 \\ T_1 \rightarrow hT_2 + a_1 U_1 + a_2 V_1 & U_1 \rightarrow a_2 + hU_2 & V_1 \rightarrow a_1 + hV_2 \\ T_2 \rightarrow hT_3 + a_1 U_2 + a_2 V_2 & U_2 \rightarrow a_2 + hU_3 & V_2 \rightarrow a_1 + hV_3 \\ T_3 \rightarrow a_1 a_2 + a_2 V_3 & & V_3 \rightarrow a_1 + hV_4 \\ & & V_4 \rightarrow a_1 + hV_5 \\ & & V_5 \rightarrow a_1 \end{array}$$

En réduisant on identifie des sommes de type $a_1 h^i a_2$ et $a_2 h^i a_1$ dont la borne supérieure décroit. De plus, on ne relève pas de boucle récursive. La généralisation donnera donc que la règle de production suivante :

$$\begin{aligned} S \rightarrow\ & o_1(a_1(a_2 + ha_2 + h^2 a_2 + h^3 a_2) + a_2(a_1 + ha_1 + h^2 a_1 + h^3 a_1 + h^4 a_1 + h^5 a_1) + \\ & h(a_1(a_2 + ha_2 + h^2 a_2) + a_2(a_1 + ha_1 + h^2 a_1 + h^3 a_1 + h^4 a_1)) + \\ & h^2(a_1(a_2 + ha_2) + a_2(a_1 + ha_1 + h^2 a_1 + h^3 a_1)) + h^3(a_1 a_2 + a_2(a_1 + ha_1 + h^2 a_1))) \end{aligned}$$

En généralisant on obtient :

$$S \rightarrow o_1 \sum_{i=0}^{d_2} h^i.[a_1.(\sum_{j=0}^{d_2-i} h^j.a_2) + a_2.(\sum_{j=0}^{d_1-i} h^j.a_1)]$$

Pour $d_2 > d_1$ on a une formule duale, tandis que pour $d_1 = d_2$ la formule reste valable.

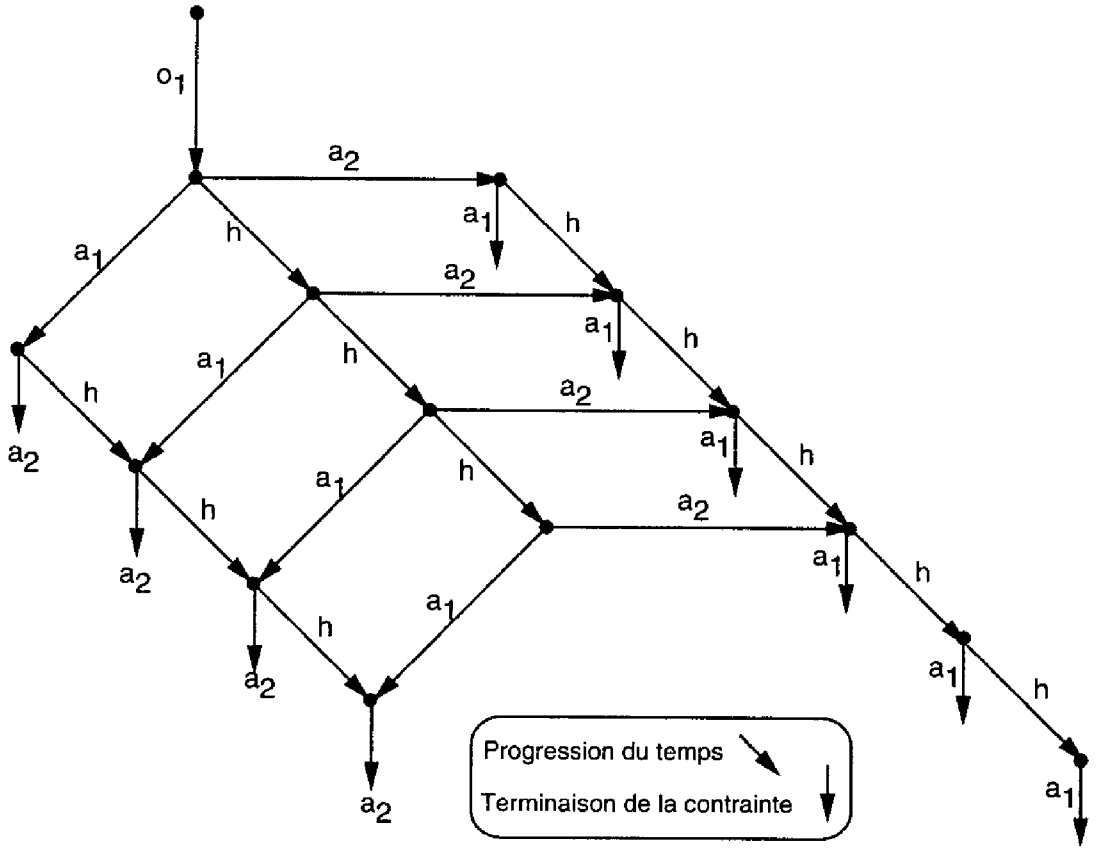


Figure V-3 : origines identiques, Max-Max

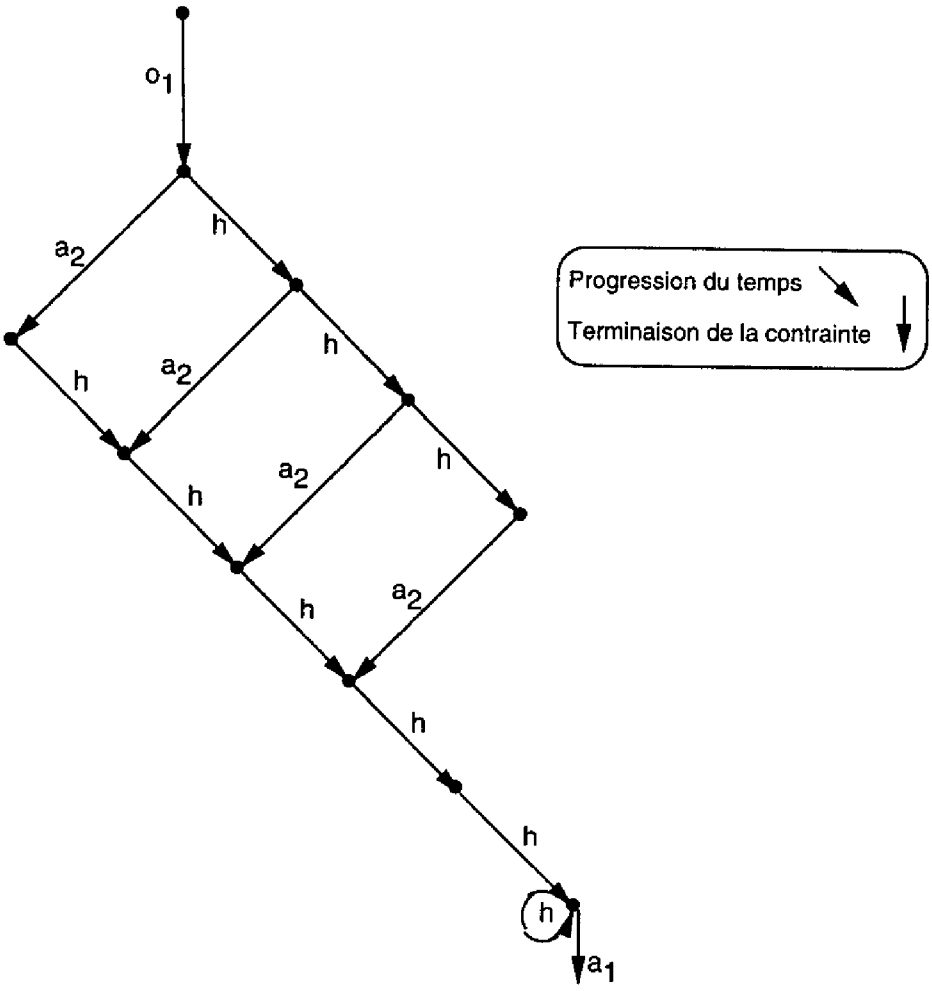


Figure V-4 : Origine identique, min-Max, $d_1 > d_2$

2.1.2. Types différents

Étant donné que le type des contraintes est différent, on aura selon les valeurs relatives de d_1 et de d_2 des formules différentes. On aura souvent trois cas ($d_1 > d_2$, $d_1 = d_2$, $d_1 < d_2$).

- $d_1 > d_2$
  Prenons l'exemple de deux contraintes C_1 et C_2 de même origine temporelle telles que :

$$C_1 = (o_1, a_1, 5, min)$$

$$C_2 = (o_1, a_2, 3, max)$$

  La figure des traces respectant C_1 et C_2 est donnée dans le schéma V-4 : En développant on obtient :

$$\begin{array}{ll} S \rightarrow o_1 T & \\ T \rightarrow hT_1 + a_2 V & V \rightarrow hV_1 \\ T_1 \rightarrow hT_2 + a_2 V_1 & V_1 \rightarrow hV_2 \\ T_2 \rightarrow hT_3 + a_2 V_2 & V_2 \rightarrow hV_3 \\ T_3 \rightarrow a_2 V_3 & V_3 \rightarrow hV_4 \\ & V_4 \rightarrow hV_5 \\ & V_5 \rightarrow hV_5 + a_1 \end{array}$$

  En réduisant on identifie une forme en $h^i a_2 h^{3-i} T$ qui correspond à la forme de géométrique de type "treillis" de la figure V-4 (partie en haut à gauche) :

$$\begin{array}{rcl} S & \rightarrow & o_1(a_2h^3 + ha_2h^2 + h^2a_2h + h^3a_2)h^2T \\ T & \rightarrow & hT + a_1 \end{array}$$

  En généralisant pour ($d_1 > d_2$) :

$$\begin{array}{rcl} S & \rightarrow & o_1[(\sum_{i=0}^{d_2} h^i a_2 h^{d_2-i})h^{d_1-d_2}T] \\ T & \rightarrow & hT + a_1 \end{array}$$

- $d_1 < d_2$
  Prenons l'exemple de deux contraintes C_1 et C_2 de même origine temporelle telles que :

$$C_1 = (o_1, a_1, 3, min)$$

$$C_2 = (o_1, a_2, 5, max)$$

  La figure des traces respectant C_1 et C_2 est donnée dans le schéma V-5 : En associant à chaque nœud un symbole terminal on obtient l'ensemble d'équations suivant :

$$\begin{array}{lll} S \rightarrow o_1 T & & \\ T \rightarrow hT_1 + a_2 V & U_3 \rightarrow a_2 + hU_4 & V \rightarrow hV_1 \\ T_1 \rightarrow hT_2 + a_2 V_1 & U_4 \rightarrow a_2 + hU_5 & V_1 \rightarrow hV_2 \\ T_2 \rightarrow hT_3 + a_2 V_2 & U_5 \rightarrow a_2 & V_2 \rightarrow hV_3 \\ T_3 \rightarrow hT_4 + a_1 U_3 + a_2 V_3 & & V_3 \rightarrow hV_4 + a_1 \\ T_4 \rightarrow hT_5 + a_1 U_4 + a_2 V_4 & & V_4 \rightarrow hV_5 + a_1 \\ T_5 \rightarrow a_1 U_5 + a_2 V_5 & & V_5 \rightarrow hV_5 + a_1 \end{array}$$

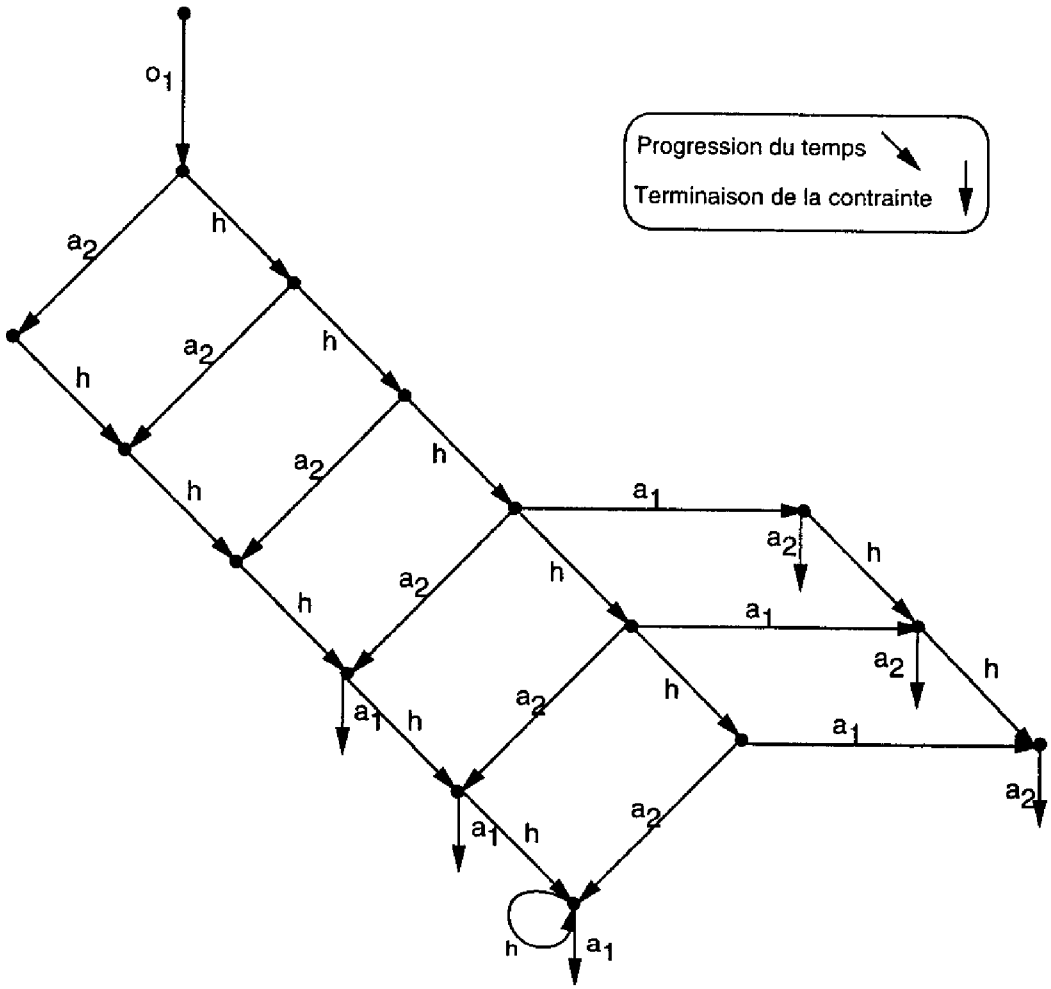


Figure V-5 : Origine identique, min-Max, $d_2 > d_1$

En réduisant, on identifie notamment une forme en

$$h^i a_1 (\sum_{j=0}^{2-i} h^j a_2)$$

$$\begin{aligned} S &\rightarrow o_1[h^3[a_1(a_2 + ha_2 + h^2 a_2) + ha_1(a_2 + ha_2) + h^2 a_1 a_2] + \\ &\quad (a_2 h^3 + ha_2 h^2 + h^2 a_2 h + h^3 a_2)(a_1 + ha_1 + h^2 V)] \\ V &\rightarrow hV + a_1 \end{aligned}$$

En généralisant :

$$\begin{aligned} S &\rightarrow o_1[h^{d_1}(\sum_{i=0}^{d_2-d_1} h^i a_1 (\sum_{j=0}^{d_2-d_1-i} h^j . a_2)) + \\ &\quad [(\sum_{i=0}^{d_1} h^i a_2 h^{d_1-i})[(\sum_{j=0}^{d_2-d_1-1} h^j a_1) + h^{d_2-d_1} V]] \\ T &\rightarrow hT + a_1 \end{aligned}$$

- $d_1 = d_2$

$$\begin{aligned} S &\rightarrow o_1[(\sum_{i=0}^{d_1} h^i a_2 h^{max(d_1-i,0)})T \\ T &\rightarrow hT + a_1 \end{aligned}$$

2.2. Même événement attendu

Ce cas correspond à deux contraintes temporelles sur un même événement attendu, mais à partir de deux origines temporelles différentes non ordonnées. Ceci correspond à un passé de type ramifié (voir la définition au chapitre I).

2.2.1. Même type de contrainte

- Minimum
  Soient deux contraintes C_1 et C_2 de type minimum et de même événement attendu :

$$C_1 = (o_1, a_1, 5, min)$$

$$C_2 = (o_2, a_1, 3, min)$$

La figure des traces compatibles avec C_1 et C_2 est donnée dans le schéma V-6 :
En développant on obtient :

$$\begin{array}{lll} S \rightarrow o_1 U + o_2 V & & \\ T \rightarrow hT_1 & U \rightarrow o_2 T + hU_1 & V \rightarrow hV + o_1 T \\ T_1 \rightarrow hT_2 & U_1 \rightarrow o_2 T_1 + hU_2 & \\ T_2 \rightarrow hT_3 & U_2 \rightarrow o_2 T_2 + hU_2 & \\ T_3 \rightarrow hT_4 & & \\ T_4 \rightarrow hT_5 & & \\ T_5 \rightarrow hT_5 + a_1 & & \end{array}$$

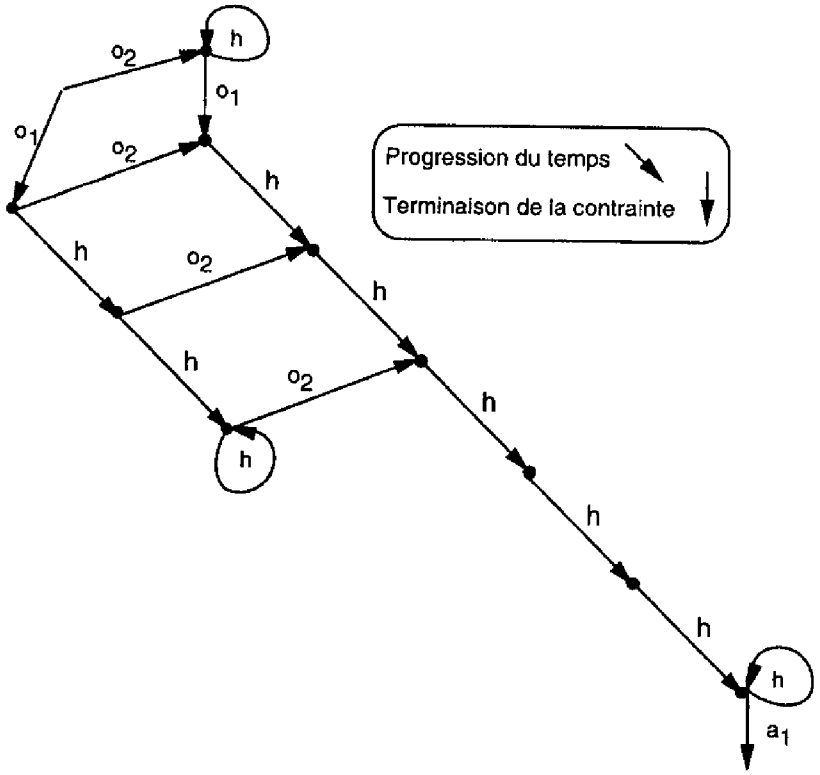


Figure V-6 : événement attendu identique, min-min

En réduisant :

$$\begin{aligned} S &\rightarrow o_1(o_2h^5T + ho_2h^4T + h^2U) + o_2V \\ U &\rightarrow hU + o_2h^3T \\ V &\rightarrow hV + o_1h^5T \\ T &\rightarrow hT + a_1 \end{aligned}$$

En généralisant :

$$\begin{aligned} S &\rightarrow o_1(\sum_{i=0}^{d_1-d_2-1} h^i o_2 h^{d_1-i}T + h^{d_1-d_2}U) + o_2V \\ U &\rightarrow hU + o_2h^{d_2}T \\ V &\rightarrow hV + o_1h^{d_1}T \\ T &\rightarrow hT + a_1 \end{aligned}$$

Pour $d_2 > d_1$, on obtient une formule duale. Pour $d_1 = d_2$, on a la grammaire très symétrique suivante :

$$\begin{aligned} S &\rightarrow o_1U + o_2V \\ U &\rightarrow hU + o_2h^{d_1}T \\ V &\rightarrow hV + o_1h^{d_1}T \\ T &\rightarrow hT + a_1 \end{aligned}$$

- Maximum

Soient deux contraintes C_1 et C_2 de type maximum et de même événement attendu :

$$C_1 = (o_1, a_1, 5, max)$$

$$C_2 = (o_2, a_1, 3, max)$$

La figure des traces compatibles avec C_1 et C_2 est donnée dans le schéma V-7 : En développant on obtient :

$$\begin{array}{lll} S \rightarrow o_1U + o_2V & & \\ T \rightarrow hT_1 + a_1 & U \rightarrow o_2T + hU_1 & V \rightarrow hV_1 + o_1T_1 \\ T_1 \rightarrow hT_2 + a_1 & U_1 \rightarrow o_2T + hU_2 & V_1 \rightarrow hV_2 + o_1T_1 \\ T_2 \rightarrow hT_3 + a_1 & U_2 \rightarrow o_2T + hU_3 & V_2 \rightarrow hV_3 + o_1T_2 \\ T_3 \rightarrow a_1 & U_3 \rightarrow o_2T_1 + hU_4 & V_3 \rightarrow o_1T_3 \\ & U_4 \rightarrow o_2T_2 + hU_5 & \\ & U_5 \rightarrow o_2T_3 & \end{array}$$

En réduisant :

$$\begin{aligned} S \rightarrow\ & o_2[o_1(a_1 + ha_1 + h^2a_1 + h^3a_1) + ho_1(a_1 + ha_1 + h^2a_1) + h^2o_1(a_1 + ha_1) + h^3o_1a_1] \\ & + o_2[(o_2 + ho_2 + h^2o_2)(a_1 + ha_1 + h^2a_1 + h^3a_1) + h^3o_2(a_1 + ha_1 + h^2a_1) + \\ & h^4o_2(a_1 + ha_1) + h^5o_2a_1] \end{aligned}$$

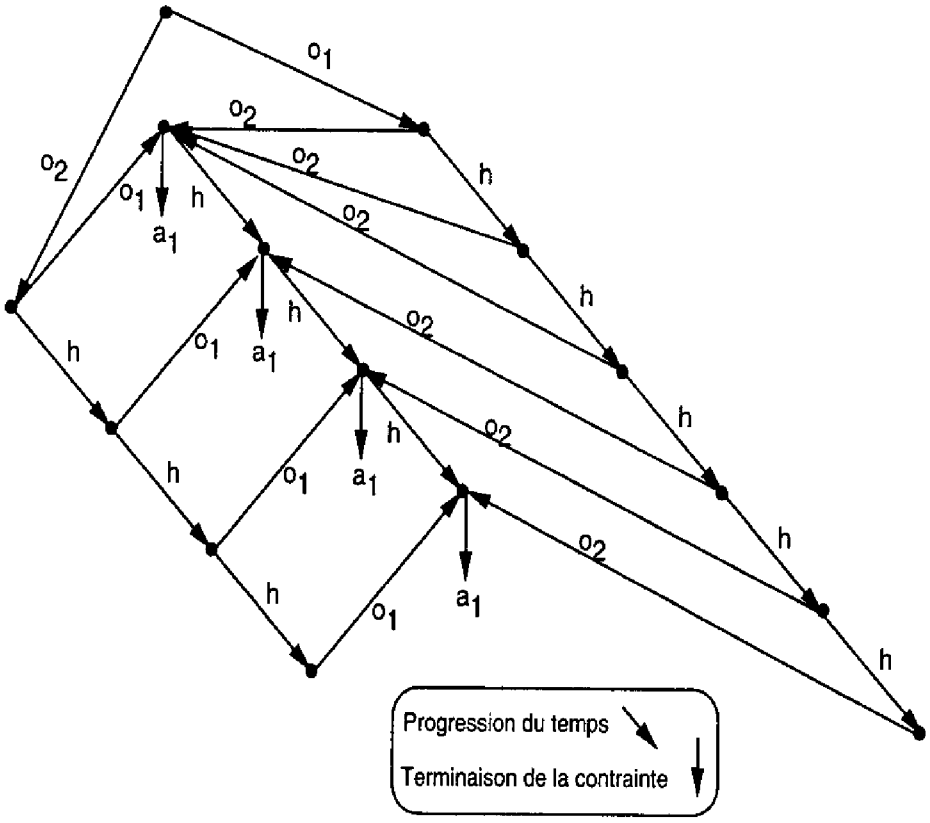


Figure V-7 : événement attendu identique, max-max

En généralisant :

$$S \rightarrow o_2[\sum_{i=0}^{d_2} h^i o_1(\sum_{j=o}^{d_2-i} h^j a_1)] + o_1[(\sum_{i=0}^{d_1-d_2} h^i o_2)(\sum_{j=o}^{d_2} h^j a_1) + (\sum_{i=d_1-d_2+1}^{d_1} h^i o_2)(\sum_{j=o}^{min(d_1-i,d_2)} h^j a_1)]$$

Pour $d_2 > d_1$, on obtient une formule duale, tandis que pour $d_1 = d_2$, la formule générale reste valable.

2.2.2. Types différents

- $d_1 > d_2$

Soient deux contraintes C_1 et C_2 de type minimum et maximum et de même événement attendu :

$$C_1 = (o_1, a_1, 5, min)$$
$$C_2 = (o_2, a_1, 3, max)$$

La figure des traces compatibles avec C_1 et C_2 est donnée dans le schéma V-8 :
En développant on obtient :

$$\begin{array}{llllll}
S \rightarrow o_1 T & & & & & \\
T \rightarrow hT_1 & & & & & \\
T_1 \rightarrow hT_2 & & & & & \\
T_2 \rightarrow hT_3 + o_2 U_2^2 & U_2^2 \rightarrow hU_3^2 & & & & \\
T_3 \rightarrow hT_4 + o_2 U_3^3 & U_3^2 \rightarrow hU_4^2 & U_3^3 \rightarrow hU_4^3 & & & \\
T_4 \rightarrow hT_5 + o_2 U_4^4 & U_4^2 \rightarrow hU_5^2 & U_4^3 \rightarrow hU_5^3 & U_4^4 \rightarrow hU_5^4 & & \\
T_5 \rightarrow hT_5 + o_2 U_5^5 & U_5^2 \rightarrow a_1 & U_5^3 \rightarrow hU_6^3 + a_1 & U_5^4 \rightarrow hU_6^4 + a_1 & U_5^5 \rightarrow hU_6^5 + a_1 & \\
 & & U_6^3 \rightarrow a_1 & U_5^4 \rightarrow hU_6^4 + a_1 & U_6^5 \rightarrow hU_7^5 + a_1 & \\
 & & & U_6^4 \rightarrow hU_7^4 + a_1 & U_7^5 \rightarrow hU_8^5 + a_1 & \\
 & & & U_7^4 \rightarrow a_1 & U_8^5 \rightarrow hU_9^5 + a_1 & \\
 & & & & U_9^5 \rightarrow a_1 &
\end{array}$$

On utilise ici un double indiçage qui nous permet d'éviter d'introduire de nouveaux symboles de non terminaux, tout en gardant l'information du temps écoulé dans l'alternative courante. Dès lors, si U signifie que o_1 et o_2 se sont produits, U_4^3 indique que dans cette alternative, trois unités de temps se sont écoulées entre o_1 et o_2 et que depuis, une unité de temps s'est écoulée.
En réduisant :

$$\begin{aligned} S &\rightarrow o_1 h^2[o_2 h^3 a_1 + h o_2 h^2(a_1 + h a_1) + h_2 o_2 h(a_1 + h a_1 + h^2 a_1) + h^3 T] \\ T &\rightarrow hT + o_2(a_1 + h a_1 + h^2 a_1 + h^3 a_1) \end{aligned}$$

En généralisant :

$$S \rightarrow o_1 h^{d_1 - d_2}[h^{d_2} T + \sum_{i=0}^{d_2-1} h^i o_2 h^{d_2-i}(\sum_{j=0}^{i} h^j a_1)]$$

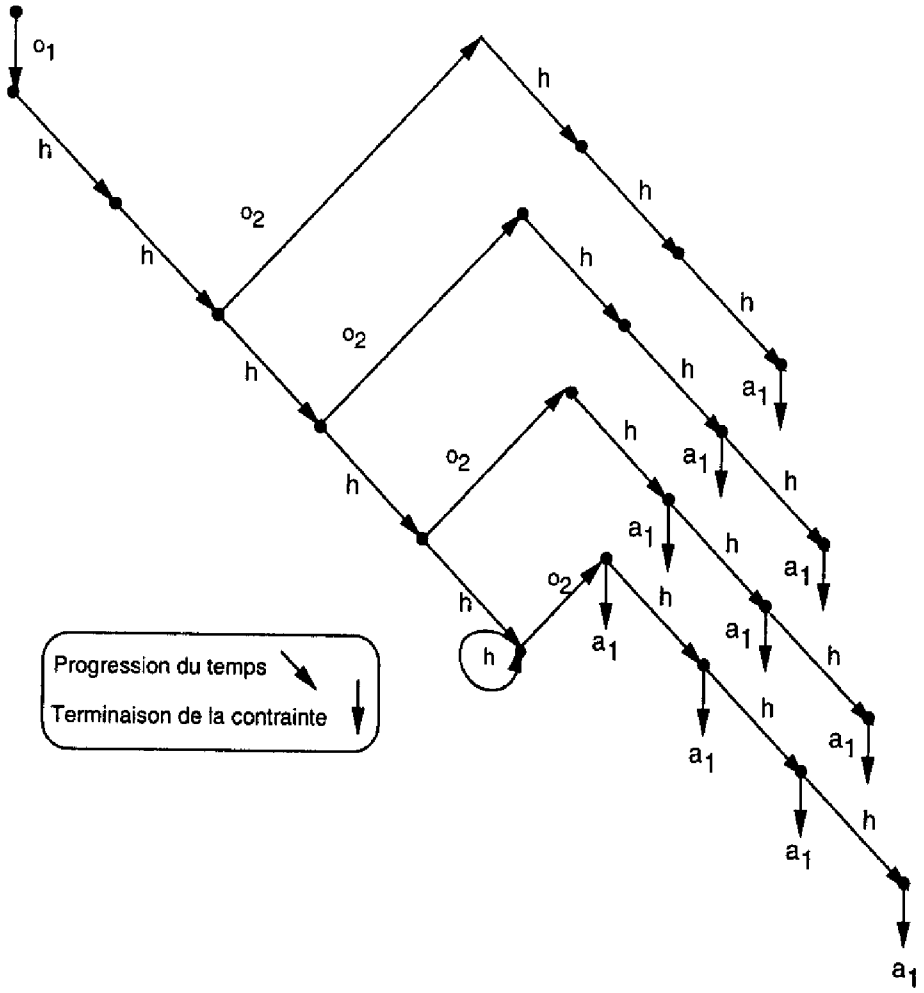


Figure V-8 : événement attendu identique, min-Max, $d_1 > d_2$

$$T \rightarrow hT + o_2(\sum_{i=0}^{d_2} h^i a_1)$$

- $d_1 < d_2$
  Soient deux contraintes C_1 et C_2 de type minimum et maximum et de même événement attendu :

$$C_1 = (o_1, a_1, 3, min)$$
$$C_2 = (o_2, a_1, 5, max)$$

La figure des traces compatibles avec C_1 et C_2 est donnée dans le schéma V-9 :
En développant on obtient :

$$\begin{array}{llllll}
S \rightarrow o_1U + o_2V & & & & & \\
U \rightarrow o_2T + hU_1 & T \rightarrow hT_1 & & & & \\
U_1 \rightarrow o_2T^1 + hU_2 & T_1 \rightarrow hT_2 & T^1 \rightarrow hT_2^1 & & & \\
U_2 \rightarrow o_2T^2 + hU_3 & T_2 \rightarrow hT_3 & T_2^1 \rightarrow hT_3^1 & T^2 \rightarrow hT_3^2 & & \\
U_3 \rightarrow o_2T^3 + hU_3 & T_3 \rightarrow hT_4 + a_1 & T_3^1 \rightarrow hT_4^1 + a_1 & T_3^2 \rightarrow hT_4^2 + a_1 & T_4^3 \rightarrow hT_5^3 + a_1 & \\
 & T_4 \rightarrow hT_5 + a_1 & T_4^1 \rightarrow hT_5^1 + a_1 & T_4^2 \rightarrow hT_5^2 + a_1 & T^3 \rightarrow hT_4^3 + a_1 & \\
 & T_5 \rightarrow a_1 & T_5^1 \rightarrow hT_6^1 + a_1 & T_5^2 \rightarrow hT_6^2 + a_1 & T_5^3 \rightarrow hT_6^3 + a_1 & \\
 & & T_6^1 \rightarrow a_1 & T_6^2 \rightarrow hT_7^2 + a_1 & T_6^3 \rightarrow hT_7^3 + a_1 & \\
 & & & T_7^2 \rightarrow a_1 & T_7^3 \rightarrow hT_8^3 + a_1 & \\
 & & & & T_8^3 \rightarrow a_1 &
\end{array}$$

En réduisant :

$$\begin{aligned}
S \rightarrow\ & o_1[o_2h^3(a_1 + ha_1 + h^2a_1) + ho_2h^3(a_1 + ha_1 + h^2a_1 + h^3a_1) + \\
& h^2o_2h^3(a_1 + ha_1 + h^2a_1 + h^4a_1) + h^3T] + \\
& o_2[o_1h^3(a_1 + ha_1 + h^2a_1) + ho_1h^3(a_1 + ha_1) + h^2o_1h^3a_1] \\
T \rightarrow\ & hT + o_2(a_1 + ha_1 + h^2a_1 + h^3a_1 + h^4a_1 + h^4a_1)
\end{aligned}$$

En généralisant :

$$\begin{aligned}
S \rightarrow\ & o_1[h^{d_1}T + \sum_{i=0}^{d_1-1} h^i o_2 h^{d_1-i}(\sum_{j=0}^{d_2-d_1+i} h^j a_1)] \\
& o_2[\sum_{i=0}^{d_2-d_1} h^i o_1 h^{d_1}(\sum_{j=0}^{d_2-d_1+i} h^j a_1)] \\
T \rightarrow\ & hT + o_2(\sum_{i=0}^{d_2} h^i a_1)
\end{aligned}$$

Cette formule reste valable pour $d_1 = d_2$.

2.2.3. Un événement attendu correspondant à un événement origine

Ce type de composition revient à "enchaîner" des grammaires. C'est à dire que la définition de l'une commencera quand l'autre se terminera.

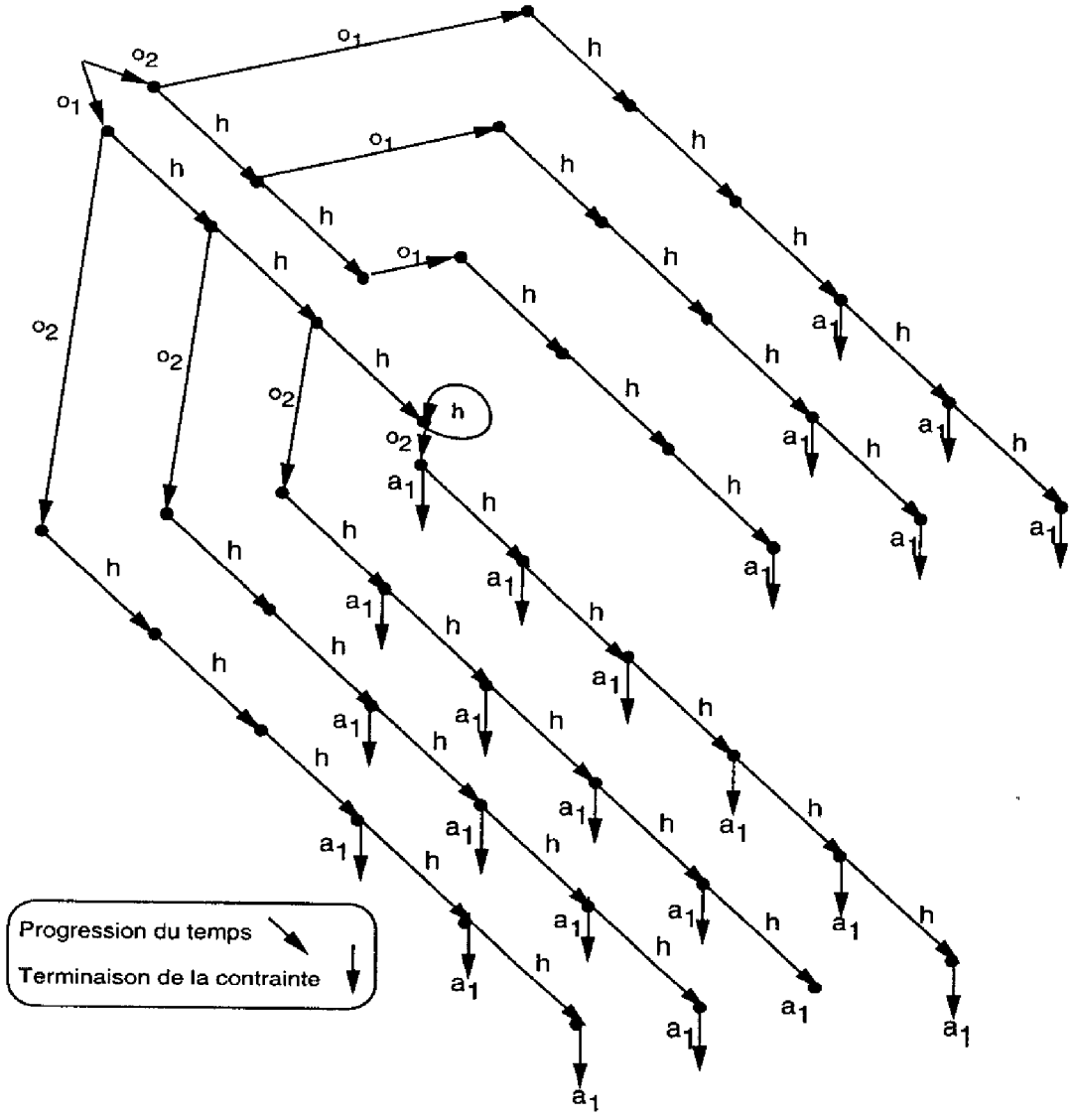


Figure V-9 : événement attendu identique, min-Max, $d_1 < d_2$

- Soient deux contraintes C_1 et C_2 de type minimum telles que :

$$C_1 = (o_1, a_1, 3, min)$$

$$C_2 = (a_1, a_2, 5, min)$$

$$\begin{aligned} S &\rightarrow o_1 h^{d_1} T \\ T &\rightarrow a_1 h^{d_2} U + hT \\ U &\rightarrow hU + a_2 \end{aligned}$$

- Soient deux contraintes C_1 et C_2 telles que :

$$C_1 = (o_1, a_1, d_1, min)$$

$$C_2 = (a_1, a_2, d_2, max)$$

$$\begin{aligned} S &\rightarrow o_1 h^{d_1} T \\ T &\rightarrow a_1 (\sum_{i=0}^{d_2} h^i a_2) + hT \end{aligned}$$

- Soient deux contraintes C_1 et C_2 telles que :

$$C_1 = (o_1, a_1, d_1, max)$$

$$C_2 = (a_1, a_2, d_2, min)$$

$$\begin{aligned} S &\rightarrow o_1 (\sum_{i=0}^{d_1} h^i a_1) h^{d_2} T \\ T &\rightarrow a_2 + hT \end{aligned}$$

- Soient deux contraintes C_1 et C_2 telles que :

$$C_1 = (o_1, a_1, d_1, max)$$

$$C_2 = (a_1, a_2, d_2, max)$$

$$S \rightarrow o_1 [\sum_{i=0}^{d_1} (h^i a_1 \sum_{j=0} d_2 h^{d_2} a_2)]$$

2.3. Même origine et même attendu

C'est le cas le plus favorable. Il s'agit ici d'un resserrement de contraintes.

- Soient deux contraintes C_1 et C_2 telles que :

$$C_1 = (o_1, a_1, d_1, min)$$

$$C_2 = (o_1, a_1, d_2, min)$$

La grammaire résultante est celle correspondant à la contrainte de plus grande durée.

- Soient deux contraintes C_1 et C_2 telles que :

$$C_1 = (o_1, a_1, d_1, max)$$

$$C_2 = (o_1, a_1, d_2, max)$$

La grammaire résultante est celle correspondant à la contrainte de plus petite durée.

- Soient deux contraintes C_1 et C_2 telles que :

$$C_1 = (o_1, a_1, d_1, min)$$

$$C_2 = (o_1, a_1, d_2, max)$$

Pour $d_1 > d_2$, les contraintes sont incompatibles. Par contre avec $d_1 \leq d_2$ on a la grammaire suivante :

$$S \rightarrow o_1 h^{d_2} (\sum_{i=0}^{d_1 - d_2} h^i) a_1$$

remarque

Toutes les formules présentées sont baties à l'aide de trois formes :

- $\sum_{i=0}^{m} h^i a$ qui une forme de type "maximum".
- $\sum_{i=n}^{m} h^i a$ pour $n \neq 0$ qui est aussi une forme de type maximum avec borne inférieure.
- les formes récursives qui sont des formes de type "minimum".

$$S \rightarrow \cdots h^d T$$
$$T \rightarrow hT + a$$

3. Procédure générale

Les expressions de conjonction de contraintes que nous avons donnés précédemment avaient pour but de réaliser la conjonction de deux contraintes de bases. Nous allons maintenant définir l'algorithme permettant de croiser une contrainte de base à un ensemble de contraintes (ensemble résultant déjà de conjonction de contraintes). Cette étape représente l'étape de récurrence de l'algorithme.

3.1. Graphe de contraintes

Nous utilisons une représentation graphique qui évite les expressions sous forme de grammaire. Ce mode de représentation permet aussi de représenter les différents cas de manière plus compréhensible. Un ensemble de contraintes peut se représenter de manière simple par un graphe de contraintes, où chaque arc $\overset{o_1}{\downarrow}{}_{a_1}$ représente une contrainte temporelle. Le sommet o_1 représente l'origine relative des temps de la contrainte et a_1 l'événement attendu. Les compositions de contraintes se représente alors simplement :

Même attendu : o_1 o_2 a_1 Même origine : a_1 o_1 o_2

Origine = attendu : o_1 a_1 a_2 Même origine et même attendu : o_1 a_1

Un ensemble conjonctif de contraintes peut donc se représenter par une combinaison des éléments décrits ci-dessus, combinaison qui peut se représenter par un graphe orienté, comme dans la figure V-10 où chaque arc représente une contrainte (min ou max). Le type de la composition est indiqué par la façon dont l'arc est attaché au graphe, c'est à dire s'il a son origine ou son attendu pendant.

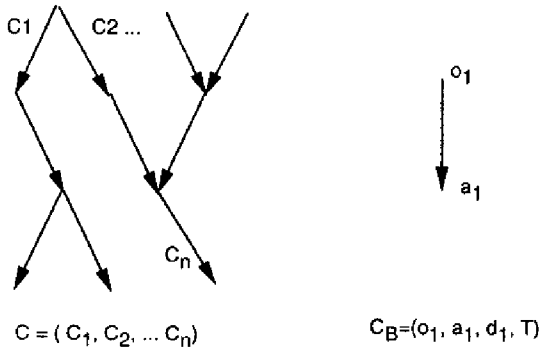
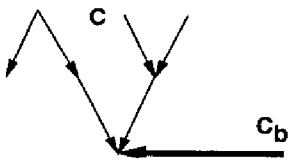


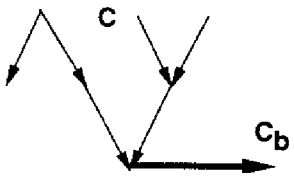
Figure V-10 : Graphes de contraintes et contraintes de base

Composer une contrainte de base c'est attacher un arc à un graphe. On retrouve donc cinq cas de figures :

- Rajout d'un arc "pendant" vers un nœud du graphe de contraintes, ce qui peut s'interpréter comme si on rajoutait une contrainte temporelle sur un événement dont l'origine des temps n'était pas encore apparue sur le graphe.

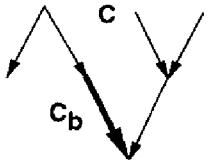


- Rajout d'un arc "pendant" à partir d'un nœud du graphe de contraintes. Cela peut s'interpréter comme si un événement du graphe devenait l'origine des temps d'une nouvelle contrainte temporelle portant sur un nouvel événement.

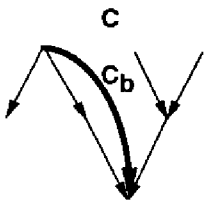


- Enfin le rajout d'un arc entre deux nœuds existant. On note alors trois sous-cas :

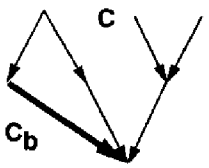
1. La superposition d'un arc sur un ancien arc. Comme dans le cas précédent, il s'agit de faire porter une nouvelle contrainte temporelle plus restrictive sur un événement à partir de la même origine temporelle.



2. Rajout d'un arc entre deux nœuds existants déjà reliés par un chemin. Cet arc constitue une nouvelle information normalement plus restrictive sur deux événements du graphe (dans le cas contraire, cette contrainte est redondante dans le cahier des charge).



3. Rajout d'un arc entre deux nœuds existants, qui ne sont pas déjà reliés par un chemin (séquence d'arcs). Cela constitue une nouvelle information d'antériorité sur deux événements du graphe qui n'étaient pas ordonnés.



3.2. Description de l'algorithme

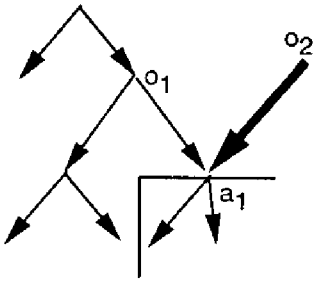
Dans cette section nous présentons l'algorithme de conjonction et nous l'illustrons sur un exemple. Cet algorithme comporte deux étapes :

3.2.1. Calcul des entrelaçages et de la faisabilité

Le calcul de la fenêtre où s'applique la nouvelle contrainte, et de l'ensemble des contraintes temporelles qui ont soit une relation de dépendances avec la nouvelle contrainte, soit dont la fenêtre se superpose avec celle de la nouvelle fenêtre. Cette étape permet aussi une analyse de la faisabilité de la composition. Pour le calcul de ces entrelaçages, on retrouve plusieurs cas de figure :

- Pendant par l'origine :

Dans ce cas, on rajoute une origine temporelle o_2 qui n'est pas ordonnée par rapport aux événements déjà exprimés sur le graphe. Ici l'ajout d'un arc a donc des répercussions sur le "passé" de a_1, c'est à dire sur la partie supérieure de la coupe.

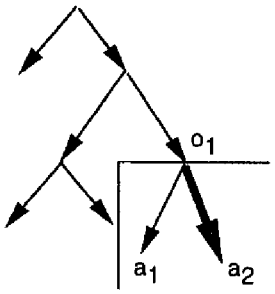


Le nœud sur lequel s'attache la contrainte définit deux parties disjointes dans le graphe. Le sous-graphe dont la racine est précisément le noeud recevant la contrainte, que l'on appelle sous-graphe inférieur, et le graphe privé de ce sous-graphe, appelé graphe supérieur. Dans le cas d'un pendant par l'origine, l'ensemble des contraintes susceptibles d'être modifiées se trouve dans le graphe supérieur.

On détermine l'ensemble des contraintes dont la fenêtre se superpose à celle de la nouvelle contrainte, dans le graphe supérieur. Entre les origines o_1 et o_2, on choisit l'origine temporelles la plus antérieure, ou à défaut l'une d'entre elles (en effet dans le cas de contraintes de type minimum, o_1 et o_2 ne peuvent être ordonnées). On calcule alors les nouvelle fenêtres des contraintes temporelles. susceptibles d'être concernées.

- Pendant par l'attendu

Les répercussions de la nouvelle contrainte vont avoir lieu sur la partie inférieure de la coupe horizontale

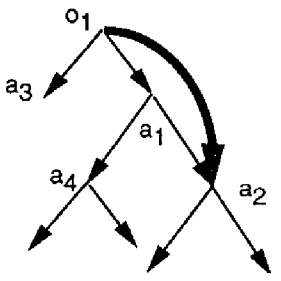


- Entre des nœuds du graphe

Ce cas nécessite une analyse préalable à la composition. En effet, la nouvelle contrainte peut être incompatible avec le graphe de contraintes.

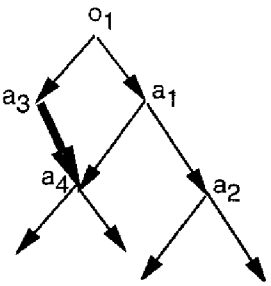
Analyse dans le cas d'un arc transitif

Il faudra dans un premier temps calculer la fenêtre de a_2, par rapport au chemin (o_1, a_1, a_2). Ensuite on calculera la fenêtre de a_2 par rapport au nouvel arc. Si ces fenêtres n'ont aucune intersection, la contrainte est rejetée.



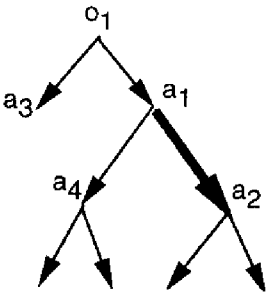
Analyse dans le cas d'un nouvel arc non transitif

On calcule les fenêtres de a_4 selon l'ensemble des chemins possibles (dans cet exemple, deux chemins possibles). Si aucune intersection n'est trouvée, la nouvelle contrainte est incompatible.



Analyse dans le cas de la superposition d'un arc sur un arc existant

En général, cela réduit la fenêtre de a_2. Cependant, un cas d'impossibilité peut se produire dans le cas où l'arc existant et le nouvel arc sont de type différent. Lorsque l'on a deux arcs correspondants à deux contraintes de type opposés min et max et lorsque la durée de la contrainte minimum est supérieure à celle de la durée de la contrainte maximum la conjonction est impossible.



3.2.1.a. Decscription de la procédure sous forme algorithmique

Nous donnons dans cette section une description un peu plus formelle (sous forme d'un algorithme), la procédure que nous venons d'évoquer. Nous donnons d'abord la description de la fonction principale puis détaillons les sous-programmes et les variables.

```
Debut
  Dependances ← Analyse_Dep(Arbre, N_Cont);
  Si (Dependance.nbdep > 1) ALORS
    Si (Dependance.type > attendu) Alors
      Fenetres ← Cacul_Fen(Arbre, N_CT, 1_dep, "sup");
    Sinon (dependance par l'origine)
      Fenetres ← Cacul_Fen(Arbre, N_CT, 1_dep, "inf");
    Finsi
  Sinon (la nouvelle contrainte portent sur deux nœuds existant dans le graphe)
    Chemin ← Analyse_Chem(Arbre, N_CT, Dependance);
    Si (Chemin.nbarc > 1) Alors
      Si (Chemin.nouveau) Alors
        Fenetre ← Calcul_Fen(Arbre,N_CT,2_dep,"Nouv_arc");
        Si Fenetre.cpt = ECHEC Alors
          Retourner "Contrainte incompatible avec l'arbre courant"
        Finsi
      Sinon (il s'agit d'un arc transitif)
        Fenetre ← Calcul_Fen(Arbre,N_CT,2_dep,"arc_trans");
        Si Fenetre.cpt = ECHEC Alors
```

```
        Retourner "Contrainte incompatible avec un chemin existant"
      Finsi
    Finsi
  Sinon (Il s'agit ici d'une superposition sur un arc existant)
    Fenetre ← Calcul_Fen(Arbre,N_CT,2_dep,"Superposition");
      Si Fenetre.cpt = ECHEC Alors
        Retourner "Resserement de contrainte impossible"
      Finsi
  Finsi
Finsi
Fin
```

Indications sur l'algorithme

- *N_CT* : Structure contenant les informations concernant la contrainte, qu'il faut composer à l'arbre courant (*Arbre*).
- *Arbre* : Structure contenant les informations l'arbre courant, notamment une liste des contraintes déjà composées.
- *Analyse_Dep* : Fonction qui analyse les dépendances de la nouvelle contrainte *N_CT* avec la liste des contraintes déjà composées qui forment la structure *Arbre*. Cette fonction renseigne la structure *dependance* qui contient notamment les champs *nbdep* : nombre de dépendances trouvé, *type* : qui indique si la contrainte est pendante par l'origine ou l'attendu.
- *Analyse_Chem* : Permet de recenser les chemins existant entre les deux nœuds sur lesquels portent la nouvelle contrainte.
- *Calcul_Fen* : Après l'examen combinatoire des cas que nous avons décrit, cette fonction calcule dans un premier temps l'entrelaçage des fenêtres impliquées dans la composition et dans un second temps génére la nouvelle grammaire.

3.2.2. Génération de grammaire

Cette génération, faite par la fonction *Calcul_Fen*, est basée sur des remarques que nous avons faites à la fin du chapitre précédent : dans toute grammaire représentant un ensemble conjonctif de grammaire on retrouve :
- Des sommes : $\sum_{i=n}^{m} \cdots$ qui sont des formes analogues à celle d'une contrainte de type maximum.
- Des règles récursives ($T \rightarrow hT + \cdots$), qui sont des formes analogues à celle d'une contrainte minimum.

Les formules présentées au chapitre IV sont réutilisées dans la génération de la nouvelle grammaire: appliquer une nouvelle contrainte temporelle de base à un ensemble déterminé de contraintes, revient à identifier des formes de bases et à leur appliquer les formules précédement établies. Cette utilisation "itérative" des formules de composition est illustrée dans l'exemple de la prochaine section.

3.2.3. Optimisation

On vient de l'évoquer, l'opérateur de conjonction est un processus itératif de composition de contraintes. On peut optimiser le fonctionnement de l'algorithme en suivant plusieurs critères sur le choix de la contrainte à composer. Le premier, est de choisir la contrainte ayant le maximum de dépendances avec l'ensemble courant de contraintes. On réduit ainsi les "degrés de liberté" de la nouvelle contrainte par rapport à la grammaire suivante, c'est à dire que cette nouvelle contrainte devrait permettre de réduire le nombre de traces admissibles par la grammaire résultante. On peut, comme deuxième critère, choisir les contraintes présentant les fenêtres les plus petites. Enfin on privilégiera d'abord les contraintes pendante par l'attendu, qui introduisent moins de complexité. En dernier critère, on choisira d'abord les contraintes de types maximum.

4. Exemple

L'exemple choisi a pour but d'illustrer l'aspect itératif de la composition. Ainsi, les choix que nous allons faire en matière de composition de contraintes simplifient le calcul des entrelaçages et nous permettent de présenter le processus de composition par reconnaissance de formes de contraintes de base.

On donne maintenant l'énoncé d'un cahier des charges comportant un ensemble de contraintes temporelles, pour lesquelles on veut construire l'analyseur syntaxique.

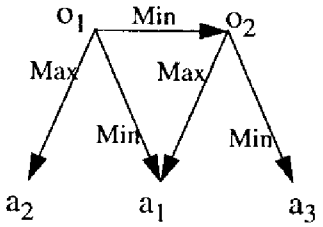
"Une personne A, désirant joindre un interlocuteur B, ne peut effectivement l'avoir en ligne qu'au plus tôt 7 secondes après avoir décroché le combiné téléphonique. B, une fois la sonnerie entendue, a 5 secondes pour décrocher. A détient n unités téléphoniques sur sa carte et devra interrompre la communication avant 60 secondes. B par ailleurs, après avoir entendu la sonnerie, ne peut transférer l'appel en moins de 4 secondes."

On relève deux origines des temps. Une sur les actions de A, et une sur celles de B. Les contraintes exprimées sur les actions de A, ont comme origine des temps l'instant où A décroche le combiné. On suppose que cette date fixe le début du décompte des unités sur la carte téléphonique. De plus, cette date marque le début de l'écoulement du délai de jonction de l'interlocuteur B. Pour B, l'origine des temps est matérialisée par la sonnerie du téléphonne, à partir de laquelle il ne doit pas laisser s'écouler plus d'un certain délai, au risque de voir A raccrocher. Par ailleurs, B ne peut transférer cet appel avant un certain délai.

On note
o_1 : "A décroche son combiné"
o_2 : "B entend la sonnerie"
a_1 : "B décroche"
a_2 : "A raccroche"
a_3 : "B transfère l'appel"

Après avoir identifié ces événements, on relève cinq contraintes :

$C_1 = (o_1, a_1, 7, min)$
$C'_1 = (o_1, o_2, 2, min)$
$C_2 = (o_1, a_2, 60, max)$
$C_3 = (o_2, a_1, 5, max)$
$C_4 = (o_2, a_3, 4, min)$



(Où C'_1 est une contrainte induite, par les autres contraintes)
Pour simplifier le calcul des entrelaçages (cela permet de fixer l'origine temporelle o_2 par rapport à o_1 et donc de calculer les entrelaçages une fois pour toutes), on débute l'algorithme à partir des contraintes suivantes :
$C'_1 = (o_1, o_2, 2, min)$
$C_2 = (o_1, a_2, 60, max)$
On peut maintenant calculer l'entrelaçage figure V-11 :

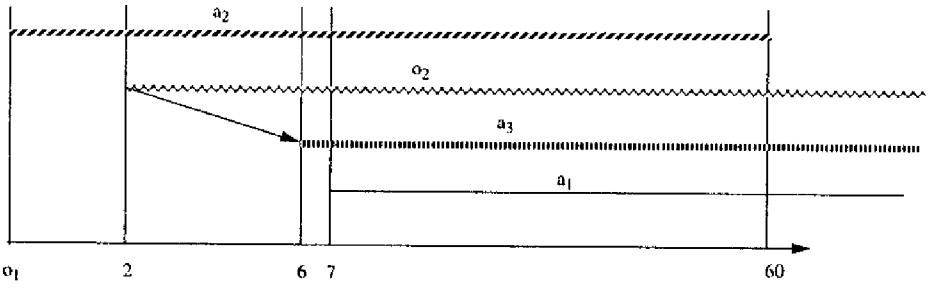


Figure V-11 : calcul des entrelaçages

Appliquons notre algorithme :

- ETAPE 1 composition à deux contraintes : $C_2 * C'_1$
  $(o_1, a_2, 60, max) * (o_1, o_2, 2, min)$
  Composition d'une contrainte minimum et d'une maximum, dont les origines sont identiques, et dont $d_1 < d_2$
  La formule générale nous donne :

$$\begin{aligned} S &\rightarrow o_1[h^2[\sum_{i=0}^{58} h^i o_2(\sum_{j=0}^{58-i} h^j a_2)] + \\ &\quad [(\sum_{i=0}^{2} h^i a_2 h^{2-i})([\sum_{j=0}^{57} h^j o_2) + h^{58}T]] \\ T &\rightarrow hT + o_2 \end{aligned}$$

Notation simplifiée
Nous introduisons cette notation pour alléger l'écriture et faciliter la compréhension de la suite du calcul.

- $date(o_2, o_1)$ représente le nombre d'occurrences de temps écoulé entre o_1 et o_2 (qui se traduira dans les formules suivantes par l'indice i).

- $Max(\epsilon, o_2, 58)$ représente, une contrainte temporelle de type maximum n'ayant pas d'origine temporelle. Son origine est l'événement placé juste avant cette expression.
- $Max^*(o, a, bi, bf)$ qui représente une contrainte maximum avec borne inférieure.
- $miMa(\epsilon, a_1, 7-i-2)(\epsilon, a_1, 5)$ représente la composition d'une contrainte minimum $(\epsilon, a_1, 7-i-2, min)$ et d'une contrainte maximum $(\epsilon, a_1, 5, max)$, dont la forme a été établi précédement (composition de contraintes de base).
- $h^n \parallel a = \sum_{i=0}^{n} h^i a h^{n-i}$ qui représente l'ensemble des séquences ou a apparaît dans l'intervalle $[0..n]$.

- ETAPE 2 : Composition à trois contraintes (introduction de la fenêtre de a_1). Cette fenêtre est déterminée par les contraintes C_1 et C_3, cependant l'utilisation à la première étape de C_1' permet de les rassembler en une seule contrainte C_3' :

  Le schéma de l'entrelacement des contraintes correpond au schéma V-11 pour lequel on a rajouté le segment concernant l'événement a_1.

$$\begin{aligned} C_1' * C_2 * C_3' &= (o_1, a_2, 60, max) * (o_1, o_2, 2, min) * \\ &\quad (o_2, a_1, 7 - date(o_2, o_1), 5) \end{aligned}$$

En développant :

$$\begin{aligned} S \rightarrow\ & o_1[h^2 Max(\epsilon, o_2, 58)[miMa(\epsilon, a_1, 7-i-2)(\epsilon, a_1, 5) * Max(\epsilon, a_2, 58-i)] + \\ & (h^2 \parallel a_2)[Max(\epsilon, o_2, 57) miMa(\epsilon, a_1, 7-i-2)(\epsilon, a_1, 5) \\ & min(\epsilon, o_2, 58) min(\epsilon, a_1, 0)]] \end{aligned}$$

$miMa(\epsilon, a_1, 5, Max)(\epsilon, a_1, 7-i-2)$ est une forme particulière de Max, c'est une somme partant d'une borne inférieure non nulle que l'on note :
$max^* = (\epsilon, a_1, 7-2-i, 5) = \sum_{j=7-i-2}^{5} h^j a_1$.

On retrouve donc dans :
$miMa(\epsilon, a_1, 5, Max)(\epsilon, a_1, 7-i-2) * Max(\epsilon, a_2, 58-i)]$ une forme de type $Max^* Max$.

Forme développée :

$$\begin{aligned} S \rightarrow\ & o_1[\sum_{i=0}^{58} h^i o_2[\sum_{j=0}^{min(58-i,5)} h^j a_2(\sum_{k=7-i-2}^{min(58-i,5)-j} h^k a_1) + \sum_{j=7-i-2}^{min(58-i,5)} h^j a_1(\sum_{k=0}^{max(58-i,5)-j} h^k a_2)] \\ & (a_2 \parallel h^2)[\sum_{i=0}^{57} h^i o_2(\sum_{j=7-i-2}^{5} h^i a_1 + h^{58} T]] \\ T \rightarrow\ & hT + o_2 V \\ V \rightarrow\ & hV + a_1 \end{aligned}$$

- ETAPE 3 : Composition à qautre contraintes (contrainte C_4) :

  Le schéma de l'entrelacement des contraintes correpond au schéma V-11 pour lequel on a rajouté le segment concernant l'événement a_3.
  $C_1'*C_2*C_3'*C_4 = (o_1, a_2, 60, max)*(o_1, o_2, 2, min)*(o_2, a_1, 7-date(o_2, o_1), 5)*(o_2, a_1, 3, min)$

  En notation simplifiée :

$$\begin{aligned} S \rightarrow\ & o_1[h^2 Max(\epsilon, o_2, 58)[Max^* Max(\epsilon, a_1, 7-i-2, 5)(\epsilon, a_2, 58-i) * Min(\epsilon, a_3, 4)] + \\ & (h^2 \parallel a_2)[Max(\epsilon, o_2, 57) Max^*(\epsilon, a_1, 7-i-2, 5) * min(\epsilon, a_3, 4) + \\ & min(\epsilon, o_2, 58) min(\epsilon, a_1, 0) * min(\epsilon a_3, 4)]] \end{aligned}$$

  Calcul de $Max^* Max(\epsilon, a_1, 7-i-2, 5)(\epsilon, a_2, 58-i) * Min(\epsilon, a_3, 4)$:
  On développe $Max^* Max$ en min et max :

$$Max^* Max(\epsilon, a_1, 7-i-2, 5)(\epsilon, a_2, 58-i) =$$

$$\sum_{j=7-i-2}^{min(5,58-i)} h^j a_1 \sum_{k=0}^{min(5,58-i)-j} h^k a_2 + \sum_{j=0}^{min(5,58-i)} h^j a_2 \sum_{k=7-i-2}^{max(5,58-i)} h^k a_1$$

$$\begin{aligned} = & Max^*(\epsilon, a_1, 7-i-2, min(5, 58-i)) Max(\epsilon, a_2, min(5, 58-i) - date(a_1, \epsilon)) + \\ & Max(\epsilon, a_2, min(5, 58-i)) Max^*(\epsilon, a_1, 7-i-2, max(5, 58-i) - date(a_2)) \end{aligned}$$

  D'où le résultat symbolique :

$$Max^* Max * min = (Max^* Ma + MaxMax^*) * min = minMax^* Max + minMaxMax^*$$

  En développant cette forme :

$$\begin{aligned} S \rightarrow\ & o_1[h^2 Max(\epsilon, o_2, 58)[Max^* min(\epsilon, a_1, 7-i-2, min(5, 58-i))(\epsilon, a_3, 4) \\ & Max(\epsilon, a_2, min(5, 58-i) - date(a_1)) + \\ & Maxmin(\epsilon, a_2, min(5, 58-i))(\epsilon, a_3, 4) \\ & Max^*(\epsilon, set, a_1, 7-i-2, min(5, 58-i) - date(a_2))] + \\ & (h^2 \parallel a_2)[Max(\epsilon, o_2, 57) Max^* min * (\epsilon, a_1, 7-i-2, 5)(\epsilon, a_3, 4) + \\ & min(\epsilon, o_2, 58) minmin(\epsilon, a_1, 0)(\epsilon, a_3, 4)]] \end{aligned}$$

  En développant la forme finale :

$$\begin{aligned} S \rightarrow\ & o_1[h^2 \sum i = 0^{58} h^i o_2[[(\sum_{j=7-i-2}^{4} h^j a_1 h^{4-j})(\sum k = 0^{min(5,58-i)-1} h^k a_3 + h^{min(5,58-i)} T_{a_3}) + \\ & h^4(\sum_{k=0}^{d=min(5,58-i)-4} h^k a_3 h^{d-k})(\sum_{j=7-i-6}^{d-k} h^j a_1)](\sum_{l=0}^{min(5,58-i)-j} h^l a_2) + \end{aligned}$$

$$[(\sum_{j=0}^{4} h^j a_2 h^{4-j})(\sum k = 0^{min(5,58-i)-j-1} h^k a_3 + h^{min(5,58-i)-j} T_{a_3}) +$$
$$h^4(\sum_{k=0}^{d=min(5,58-i)-j-4} h^k a_3 h^{d-k})(\sum_{j=0}^{d-k} h^j a_2)](\sum_{l=0}^{min(5,58-i)-j} h^l a_2)]] +$$
$$(h^2 \parallel a_2)[\sum_{i=0}^{57} h^i o_2[(\sum_{j=7-i-2}^{4} h^j a_1 h^{4-j})(\sum_{k=0}^{4} h^k a_3 + h^5 T_{a_3}) +$$
$$h^4(\sum_{j=0}^{1} h^j a_3 h^{1-j})(\sum_{k=7-i-2}^{1-j} h^k a_1) + h^{58} T_{o_2}]]$$

$$T_{o_2} \rightarrow hT_{o_2} + o_2[h^4 T + \sum i = 0^1 h^i a_1 h^{4-i} T_{a_3}]$$
$$T \rightarrow a_3 T_{a_1} + a_1 T_{a_3} + hT$$
$$T_{a_3} \rightarrow hT_{a_3} + a_3$$
$$T_{a_1} \rightarrow hT_{a_1} + a_1$$

5. Réduction de la complexité

La composition de toutes les contraintes temporelles d'un cahier des charges d'un système temps réel produirait une grammaire complexe et ne serait pas très utile, car on serait tenté par la suite de l'utiliser relativement à une fenêtre temporelle donnée, ou à un type d'événements donné. Aussi, pour augmenter l'efficacité de cette méthode nous préconisons de suivre les consignes suivantes :

- Ne considérer que les événements pertinents de la contrainte.

  Exemple :
  Si l'on doit composer les 7 chiffres d'un numéro téléphonique en moins de 30 secondes, il suffit de faire porter la contrainte entre le premier et le dernier chiffre. Si $1, 2, \cdots, 7$ sont les événements associés à la composition des 7 numéros, la contrainte finale pourrait s'écrire : $C = 1 \sum_{i=1}^{30} h^i 7$. L'examen de la trace finale comportera tous les événements $1, 2, \cdots 7$ et h, mais ceux-ci seront filtrés pour ne laisser apparaître que $1, 7, h$.

- Raisonner par fenêtre temporelle.
  On examine un ensemble de contraintes dans une fenêtre temporelle donnée.

  Exemple :
  Entre les événements *décrochage du combiné* et *obtention du correspondant*, on peut consacrer une grammaire dédiée à l'ensemble des contraintes intervenant dans cet intervalle.

- Raisonner par scénarios :
  On n'examine que des contraintes traitant d'un même sujet.

  Exemple :
  Consacrer une grammaire par type de communication téléphonique :

- une conversation à trois personnes
- protocole d'échange de données

- Examiner les contraintes temporelles par rapport à un niveau de détail donné.

  Exemple :
  On peut dans l'exemple de l'échange de données par le réseau, s'intéresser dans un premier temps aux contraintes temporelles d'échange d'un message complet, puis raffiner aux niveau des paquets, au niveau des trames, ...

6. Conclusion du chapitre

On a présenté dans ce chapitre, l'ensemble des algorithmes permettant de composer des contraintes temporelles. Cette composition nous permet d'obtenir sous forme statique, une expression générale du comportement temporel d'un ensemble de contraintes. Cette expression générale nous permet :

- De rejeter toute contrainte temporelle inconsistante avec l'ensemble courant et donc de produire un ensemble consistant de contraintes temporelles.
- De vérifier s'il n'existe pas de redondances entre des contraintes, en vérifiant par exemple, qu'un ensemble de traces construites selon un schéma syntaxique particulier appartient ou non à la grammaire en courante.
- De produire un analyseur syntaxique. Une grammaire peut se transformer aisément en analyseur syntaxique par le biais de l'outil Unix Lex & Yacc [1] [Aho 1974]. Cet analyseur permet de vérifier qu'une trace vérifie bien toutes les contraintes temporelles du cahier des charges.

[1] Lex pour des grammaires régulières, Yacc pour des grammaires indépendantes du contexte

Conclusion générale

Nous avons proposé dans ce mémoire une nouvelle représentation du temps et des contraintes temporelles. L'écoulement du temps est modélisé par l'occurrence d'un événement. Cette conception nous permet de considérer les contraintes temporelles comme des propriétés de séquences. En relation avec une notation simple, basée sur la notion de grammaire, les propriétés temporelles deviennent syntaxiques.

Ces considérations nous ont amenés à proposer une nouvelle méthode d'analyse des contraintes temporelles. L'approche que nous proposons se veut complémentaire aux méthodes de spécification existantes. Elle a pour objet de vérifier que toutes les contraintes temporelles du cahier des charges ont bien été prises en compte. Pour cela, on construit un analyseur de traces temporisées. La construction de cet analyseur se fait en deux étapes. Il faut tout d'abord extraire les contraintes temporelles en suivant une **décomposition structurée**, qui nous permet d'identifier un ensemble d'opérateurs liant les contraintes (periodicité, sporadicité, disjonction et conjonction). La seconde étape est la recomposition de ces contraintes, par l'ensemble des opérateurs précédemment identifiés, le résultat final de cette recomposition devant générer un analyseur syntaxique.

Les travaux que nous avons effectués ont défriché un nouveau domaine de recherche : l'analyse et la représentation des contraintes temporelles pour la validation de spécification. Ils n'introduisent pas de nouvelles notations mais au contraire se basent sur un formalisme universel et général, celui des grammaires. En cela notre objectif était de ramener l'analyse temporelle vers une approche syntaxique.

Des prolongements à moyen ou à long terme peuvent être envisagés à ces travaux. A moyen terme, il faudrait développer de nouveaux algorithmes pour prendre en compte les contraintes temporelles linéaires et les contraintes temporelles polynomiales. A long terme, en attachant des attributs à nos grammaires on peut envisager de transformer l'analyseur syntaxique obtenu en compilateur. Celui-ci pourrait permettre de synthétiser des informations (de type statistique par exemple) à partir d'un jeu de traces d'exécution donné. Une autre extension pourrait être de combiner notre approche à la logique temporelle. On pourrait par exemple vérifier une propriétée sur notre analyseur. En effet, le modèle d'une propriété constitue un ensemble de séquences d'événements qui peut constituer un point d'entrée pour notre analyseur.

On envisage également l'intégration de ces travaux [Delfieu 1994b] à des méthodes de spécification semi-formelles (type SA-RT) ou à objets, pour lesquelles des outils formels de traitement du temps font défaut.

ANNEXE A

DÉFINITION FORMELLE DES STATECHARTS

A.1 Définition de base

Un Statechart est un diagramme à états qui peut se définir de la façon suivante :

- DÉFINITION 0 "STATECHARTS"
  Un SC est un quintuplet (Π, S, Γ, r, V) où :
  - Π : Ensemble (cf $\sum$ dans l'automate) d'événements qui comprend aussi bien ceux qui viennent de l'environnement du système que ceux qui servent à la synchronisation interne ou ceux qui sont émis vers l'environnement.
  - S : ensemble d'états (cf Q dans l'automate)
  - Γ : Ensemble de transitions. Ces transitions sont du même type que celle des Machines de Mealy.
  - r : État racine.
  - V : Ensemble de variables.

A.2 Définitions formelles des principales caractéristiques des SC

- **Parallélisme et abstraction**
  Les notions d'abstraction et de parallélisme se traduisent au niveau de la syntaxe formelle par les deux fonctions suivantes :
  * fils(s) : $S \rightarrow 2^S$
    Cette fonction associe à un état un ensemble de sous états. L'état est dit basique si fils(s)=$\{\emptyset\}$; sinon l'état est dit composite. Cette fonction introduit la notion d'état abstrait et donc la notion de décomposition d'un état en sous-états.
  * type(s) : $S \rightarrow \{ET, OU\}$
    Cette fonction donne un type à chaque état. Si type(s) = "ET" alors l'ensemble des fils de s sont des composants parallèles. Le contrôle se trouve distribué dans plusieurs états à la fois : il y a la notion de conjonction.
    Si type(s) = "OU" alors l'ensemble des fils de s sont des états séquentiels, c'est à dire que le contrôle ne peut se trouver que dans un seul état à la fois. Ce qui

amène à la définition suivante :

DÉFINITION 1 "ORTHOGONALITÉ DE COMPOSANTS"

x ⊥ y ssi
- x et y ne sont pas reliés par une relation d'ancestralité (cette relation peut être formalisée comme la fermeture transitive de la fonction fils)
- l'état le plus plus immédiat les contenant est un état "ET".

Cette notion peut se comprendre facilement si l'on interprète un SC comme un arbre orienté "ET/OU" de racine r où chaque sous-arbre est un état composite, et chaque feuille un état basique. x ⊥ y ssi il n'existe pas de chemin entre x et y. Par exemple, dans l'arbre orienté V-12, il n'y a pas de chemin entre x et y.

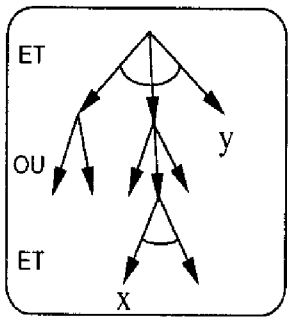


Figure V-12 : Arbre et/ou

On définit une configuration comme un sous-arbre du SC, où les états composites "ET" ont tous leurs fils, tandis que les états "OU" n'ont qu'un seul fils.

La différence des SC avec les automates porte essentiellement sur le calcul des prochaines transitions et des nouveaux états.

- **Les transitions**

  Elles se définissent comme dans les Réseaux de Petri par des pré-conditions et des post-conditions. Dans les SC, les pré-conditions sont représentées par une fonction source(t) (états de départ) et cible(t) pour les post-conditions (états d'arrivées).

  Comme les RP, ces transitions sont n-aires (voir figure V-13). Alors que dans les RP cette arité permet d'introduire le concept de synchronisation, dans les SC elle est utilisée pour traduire l'abstraction.

  On définit plage(t) comme le plus petit état "OU" englobant t qui contient les ensembles source(t) ∪ cible(t).

A.3 Définitions concernant le tir de transitions

On donne un ensemble de définitions qui nous permet ensuite de définir la procédure de calcul de tir de transitions.

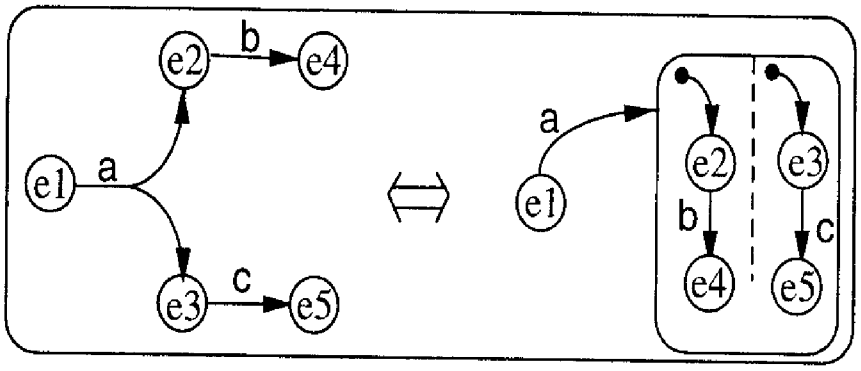


Figure V-13 : Équivalent d'une flèche n_aire en Statecharts

DÉFINITION 2 "CONSISTANCE"

Deux transitions sont *consistantes* si
Soit t1=t2.
Soit plage(t1) ⊥ plage(t2)

On note *consitant*(T), l'ensemble des transitions du SC qui sont consistantes avec toutes les transitions de T.

Deux transitions sont en conflit si elles ne sont pas consistantes. Cette définition rejoint celle de conflit structurel que l'on trouve dans les RP. On pourrait introduire la notion de conflit effectif en analysant si les transitions ont la possibilité d'être sensibilisées en même temps, par exemple si l'on admet que leurs événements déclencheurs puissent apparaître en même temps.

Aux transitions sont attachées des expressions de la forme : **E[C]/A**

* E ($\in \Pi$) est un ensemble d'événements,
* C l'ensemble des conditions,
* A l'ensemble des actions. Soit e et f des événements, s un état, u,v des variables, R une opération $R \in \{=, <, >, \neq, \leq, \geq\}$
* exit(s), entered(s), $e \wedge f$, $e \vee f$, written(v), true(c), ... respectivement la sortie ou l'entrée dans un état, la conjonction ou la disjonction d'événements, l'écriture dans une variable, ou le passage d'une condition à "vraie".
* in(s), not_yet(e), u R v, sont des conditions que l'on peut retrouver dans C. On appelle *action*(t), l'ensemble des événements générés par une transition, c'est à dire ceux qui sont produits par la partie gauche de l'étiquette des transitions.

– **Le tir de transition**

La fonction de tir n'est pas réalisée comme dans les RP comme une soustraction de matrices, mais à l'aide d'une procédure de type itératif, dont on ne connaît pas à priori le nombre d'itérations. Nous définissons les ensembles suivants avant de décrire l'algorithme de cette procédure :

* generes(T) = $\cup_{t \in T}$ action(t)

* triggered(E) l'ensemble des transitions déclenchées par l'ensemble d'événements E.
* compatible(C) : l'ensemble des transitions dont les états sources sont dans la configuration C.
* On définit d'abord une fonction qui détermine une itération de la recherche. Pour une configuration donnée C, et un ensemble d'événement $I \in \Pi$,

$$En_C(T) = compatible(C) \cap consistant(T) \cap sensibilise(I \cup generes(T))$$

Cette ensemble permet de trouver les transitions justes sensibilisables après l'examen d'un pas. L'algorithme complet peut alors se décrire de la façon suivante :

Procedure TIR

* 1 Initialement on a un ensemble $T = \emptyset$
* 2 Comparaison de En(T) à T
  1. Si T = En(T) alors l'algorithme s'arrête avec succès, et T représente l'ensemble des transitions qui seront tirées en même temps.
  2. Si T ⊂ En(T) : de nouvelles transitions ont été détectées soit $t \in En(T) - T$, on relance l'algorithme sur $T \cup t$
  3. Sinon rapport d'erreur.

Pneuli à montré dans [Pneuli 1989] que si cet algorithme est appliqué sur un SC bien formé il se termine alors sans erreur. Cette démonstration est réalisée en montrant au préalable que la fonction En est décroissante monotone.

ANNEXE B

DÉFINITIONS CONCERNANT LE LANGAGE LOTOS

Présentation du langage LOTOS

Ce langage est fondé sur les principes de CCS. Sa syntaxe est basé sur les opérateurs suivants :

- l'opérateur de séquence ";"
  $a; B \rightarrow B$ s'interprète de la façon suivante : lorsque l'événement a est observé, le processus (que l'on appelle aussi comportement) $a; B$ se dérive (on dit aussi se transforme) en B.

  Exemple :
  Le comportement $a; b; stop$ peut se transformer en $b; stop$ après l'observation de a, et sous la forme du processus "puits" *stop* (symbolisant un échec) après l'observation de a suivi de b.

- l'opérateur de choix noté []
  $B_1[]B_2$ signifie : "dès qu'une action de B_i (i=1 ou 2) alors ce comportement se transforme en B_i"

  Exemple :
  $(a; b; c; stop)[](c; d; e; exit) \rightarrow^c d; e; stop$
  ou exit est un processus terminal indiquant que le processus se termine avec succès.

- l'opérateur d'interruption $P[> Q$
  il s'interprète de la façon suivante : "dès qu'une action de Q, se produit, ce comportement se dérive en Q, sinon il se déroule comme P (tout en restant sensible à une action de Q).

  Exemple :
  $(a; b; c; exit)[> (c; stop) \rightarrow^c stop$
  $(a; b; c; exit)[> (c; stop) \rightarrow^a (b; c; exit)[> (c; stop)$

- l'opérateur de phase : $P[> Q$: il s'interprète de la façon suivante : "si P se termine en $exit$, alors Q est exécuté".

  Exemple :
  $(a; b; c; exit)[](c; d; stop) >> c; exit$ se dérive en $c; exit$

si et seulement si on peut observer les événements a suivi de b suivi de c.

- l'opérateur de synchronisation gardée $P[a,b]Q$ il s'interprète de la façon suivante : "si P veut évoluer en a ou b, alors il ne peut le faire que si Q peut le faire".

  Exemple :
  $(a;c;a;exit)[c](d;c;stop) \rightarrow^{a} (c;a;exit)[c](d;c;stop)$
  A ce moment seul l'événement d est possible et donc
  $(c;a;exit)[c](d;c;stop) \rightarrow^{d} (c;a;exit)[c](c;stop)$
  enfin on a :
  $(c;a;exit)[c](c;stop) \rightarrow^{c} stop$

- D'autres opérateurs existent : la synchronisation pure, qui est une extension de la synchronisation gardée à tous les événements apparaissant dans les comportements ; l'entrelacement "pur" qui correspond à un opérateur de parallèlisme, ou l'opérateur de récursion.

Définition formelle de l'équivalence observationnelle

Lotos est un langage de spécification qui permet une description à plusieurs niveaux. Les notions d'abstraction et d'événement interne ont conduit à interpréter un haut niveau par rapport à un bas niveau de description, comme la spécification et l'implémentation d'un problème. Ainsi, la partie gauche de la figure III-26 représente une implémentation de la figure III-25.

L'usage que l'on fait de l'opérateur parallèle peut conduire à obtenir de multiples versions de la même spécification. Ainsi l'équivalence observationnelle permet d'établir qu'une implémentation est conforme à un cahier des charges, mais aussi de montrer que deux programmes répondent au même cahier des charges.

La notion d'équivalence observationnelle est l'idée qu'un observateur externe voyant se dérouler deux processus, ne peut les distinguer. La définition formelle de la notion d'équivalence observationnelle passe par la définition d'une nouvelle relation notée $\Rightarrow$, et d'une relation de bissimulation notée $\mathcal{R}$.

DÉFINITION 3 "RELATION DE BISSIMULATION"

Soit $s = k_0 g_0 \cdots g_n k_n$, où g_i est une action observable, et s une séquence d'actions observables comprenant un nombre quelconque de transitions internes.

(i) Si s est une séquence acceptable par le système, c'est à dire telle que $B - s \rightarrow B'$ Alors $B = s \Rightarrow B'$

(ii) Dans le cas où $B - k \rightarrow B'$ alors $B = \epsilon \Rightarrow B'$ et $B = \epsilon \Rightarrow B$ (o'u ϵ désigne la chaîne vide).

Exemple :
Soit la séquence *iab* appliqué sur le comportement B :
$B_0 - i \rightarrow B_1 - a \rightarrow B_2 - i \rightarrow B_3 - b \rightarrow B_4$
En appliquant cette définition, on obtient pour la relation $\Rightarrow$:

$B_0 = \epsilon \Rightarrow B_0, B_0 = c \Rightarrow B_1, B_0 = a \Rightarrow B_2, B_0 = a \Rightarrow B_3, B_0 = ab \Rightarrow B_4$
$B_1 = \epsilon \Rightarrow B_1, B_1 = a \Rightarrow B_2, B_1 = a \Rightarrow B_3, B_1 = ab \Rightarrow B_4$
$B_2 = \epsilon \Rightarrow B_2, B_2 = \epsilon \Rightarrow B_3, B_2 = b \Rightarrow B_4$
$B_3 = \epsilon \Rightarrow B_3, B_3 = b \Rightarrow B_4$
$B_4 = \epsilon \Rightarrow B_4$

Cette définition transforme les séquences de transitions, en absorbant toute transition interne comprise entre deux actions observables. Elle conserve, toutefois, les transitions purement internes (fermeture transitive des chaînes de "i-transitions" par la règle ii)

La définition suivante formalise le processus d'observation du système. Elle est basée sur la relation précédente ($\Rightarrow$) qui sert de base pour définir la notion d'*Équivalence Observationnelle*. Cette notion peut se définir de façon informelle : si toutes les expériences (séquences définies par la relation $\Rightarrow$) réussissent sur deux comportements, alors ils sont observationnellement équivalents.

DÉFINITION 4 "ÉQUIVALENCE OBSERVATIONNELLE : $=_{obs}$"
Il existe une relation $\mathcal{R}$ de bissimulation entre deux arbres B_1 et B_2 ssi :
$\forall s$ séquence d'actions observables :
i) si $B_1 = s \Rightarrow B_1'$ alors $\exists B_2'$ tel que $B_2 = s \Rightarrow B_2'$ et $B_1' \mathcal{R} B_2'$
ii) si $B_2 = s \Rightarrow B_2'$ alors $\exists B_1'$ tel que $B_1 = s \Rightarrow B_1'$ et $B_1' \mathcal{R} B_2'$.

Cette méthode peut s'avérer lourde, et exige une double exploration des deux comportements sur la relation $\Rightarrow$, qui contient la fermeture transitive des "i-transitions" et qui fait donc exploser, en général, le nombre de séquences observables, comme on a pu le constater sur l'exemple précédent (on passe de 4 séquences pour $\rightarrow$ à 14 pour $\Rightarrow$)

Une autre alternative, pour montrer l'équivalence entre deux comportements, est d'utiliser une méthode axiomatique. L'objectif est d'appliquer des règles de réécriture sur des expressions, pour obtenir une forme canonique identique. La définition de ces règles impose toutefois de définir des congruences[2](notée $\approx_c$). Si dans l'expression E, figure une sous-expression F, et s'il existe une règle de transformation de F en F', cette réécriture n'est possible dans E que si F est congruente observationnellement à F'.

En effet, il ne suffit pas que ces deux expressions soient observationnellement équivalentes. Ainsi, si $a =_{obs} i;a$ (si a est équivalent observationellement à $i;a$) par contre, a et $i;a$ ne sont pas substitutifs. En effet $b + a \neq_{obs} b + ia$ ($b + a$ n'est pas équivalent observationellement à $b + i;a$). Il y a trois axiomes de réécriture respectant la congruance observationnelle :

- $\mu iB \approx_c \mu B$ où μ est une action observable. Ceci traduit l'absorption des transitions internes intercalées, déjà évoquée pour la définition de la relation de Bissimulation ($\mathcal{R}$).
- $B[]iB \approx_c iB$
- $\mu(B[]iC) + \mu C \approx_c \mu(B[]iC)$

de la commande

[2]La congruence est plus forte que l'équivalence, dans la mesure où si deux expressions sont congruentes alors, quelque soit le contexte ou apparait l'une, on peut la remplacer par l'autre

ANNEXE C

DÉFINITIONS CONCERNANT LES APPROCHES LOGIQUES

A.4 Fondements Théoriques des systèmes axiomatiques

On appelle **formule bien formée** ou plus simplement formule, toute construction syntaxique construite à l'aide du langage. Les axiomes sont des formules bien formées qui sont considérées vraies sans démonstration. Les théories que nous allons évoquer, se distinguent essentiellement par les axiomes sur lesquels elles sont basées.

Exemple de formule bien formée : $succ(succ(0))$ et $succ(0) < 0$ (cet exemple illustre le fait qu'il ne faut pas confondre la construction d'une formule et son interprétation).

Exemple d'axiomes, concernant la théorie des nombres entiers :
$A0 : 0 < succ(0)$
$A1 : m < n \rightarrow succ(m) < succ(n)$

Les **règles** permettent de produire de nouvelles connaissances. Elles sont en général de la forme :

$$\frac{f_1, f_2, \cdots f_n}{f_0}$$

ce qui signifie, si $f_1, f_2, \cdots f_n$ sont des théorèmes, alors f_0 est aussi un théorème.

Exemple :
La plupart des théories contiennent la règle du Modus Ponens

$$MP : \frac{P, P \rightarrow Q}{Q}$$

Les **théorèmes** sont des formules bien formées, dérivées des axiomes et des règles. Ils sont notés $\vdash T$

Exemple :
$\vdash succ(0) < succ(succ(0))$
Ce théorème s'obtient par A_0 et A_1 où l'on a substitué m par 0, et n par $succ(0)$, puis appliqué MP.
Une **interprétation** est l'association à chaque élément (formule construite sans utiliser de connecteurs logiques) de la théorie, d'une valeur appartenant à un domaine et d'une valeur de vérité pour les formules théorèmes ou les axiomes.

L'interprétation est "l'application de la théorie à la réalité". Un **modèle**, est une interprétation qui associent aux théorèmes et aux axiomes la valeur booléenne vraie.

Exemple :

$$\mathcal{I} = \begin{array}{ll} 0 & \longrightarrow \ 0 \\ succ(0) & \longrightarrow \ 1 \\ & \vdots \\ succ((\cdots_n(succ(0)\cdots_n) & \longrightarrow \ n \\ > & \longrightarrow \ > \\ & \vdots \end{array}$$

$A_0 : 0 < succ(0) \equiv 0 < 1 = vrai$

$A_1 : m < n \rightarrow succ(m) < succ(n) \equiv m < n \rightarrow m + 1 > n + 1 = vrai$

L'ensemble $\mathcal{I}$ de correspondance ci-dessus, est un modèle de la théorie des entiers naturels. Il peut exister plusieurs modèles pour une théorie. Ainsi si "0" représente la date de naissance de Jésus Christ, "succ" étant la fonction qui calcule la date du jour suivant, on obtient un nouveau modèle de la théorie.

Une théorie est **complète** si pour toute formule bien formée, on peut dériver f ou $\neg f$, autrement dit, si toute propriété vraie peut être démontrée. Un problème se présente lorsqu'il n'existe pas de procédure qui au bout d'un temps fini permette de démontrer qu'une formule admet un modèle ou non. On dit alors que la théorie est **indécidable**.

L'enrichissement des théories s'est fait en rajoutant la quantification, (logique des prédicats ou logique du premier ordre), ou des opérateurs modaux (logique modale). Cependant ces théories perdent alors leur pouvoir de décidabilité. C'est pourquoi, il est important de montrer qu'elles restent complètes, c'est à dire que l'on peut par l'axiomatique montrer qu'une formule est valide ou non. L'introduction de nouvelles théories est intéressante si on peut établir une procédure de décidabilité, ou à défaut, si on peut montrer qu'elles sont complètes.

A.5 La preuve de programme

Avant d'aborder les systèmes axiomatiques traitant les systèmes temps réels, on va évoquer les travaux de Floyd [Floyd 1967] et de Hoare [Hoare 1969], qui furent les premiers à développer des théories appliquées à la preuve de programme.

Floyd présenta un ensemble de techniques de vérification axiomatiques sur des programmes exprimés à l'aide d'organigramme. Ces travaux furent ensuite repris par Hoare, qui les appliqua aux formes textuelles. Le langage basé sur une logique du premier ordre, permettait d'écrire des formules sous la forme "Pré-Post" : $P\ a\ Q$

Pour chaque instruction de base : affectation, composition d'instruction, boucle, ... Hoare définissait une formule de ce type, où P représentait les pré-conditions (expression booléene) et Q les post-conditions d'exécution de l'instruction.

Hoare introduisit aussi des règles et un ensemble d'axiomes, qui lui permettait de prouver certaines propriétés de programme. On peut noter, par exemple, que les axiomes d'arithmétiques prenaient en compte, l'implantation réelle (hardware), notamment en exprimant que le domaine des entiers était borné, ou en définissant une addition de type "modulo" selon la taille des registres.

Il y a deux types de propriétés afférentes aux preuves de programmes :

- **la correction partielle**
  Un fragment de programme "a" est partiellement correct pour la pré-condition P et la post-condition Q si et seulement si a est exécuté dans un état où P est vérifié. Si cette exécution se termine alors l'état résultant vérifie Q.
- **la correction totale**
  Un fragment de programme "a" est partiellement correct pour la pré-condition P et la post-condition Q si et seulement si a est exécuté dans un état où P est vérifié; alors l'exécution se termine et vérifie Q.

Cependant, la théorie de Hoare ne permettait de prouver que des propriétés de correction partielle. D'autres théories seront développées pour satisfaire ces deux classes.

A.6 Logique des prédicats

La logique des prédicats offre un pouvoir d'expression supérieur à celui de la logique classique (logique des propositions). Elle peut se définir par la donnée des éléments de base de son langage :

- les variables (ex : m et n sont des variables dans la théorie des entiers naturels)
- les constantes (ex : "0" élément sur lequel on avait construit la théorie des entiers)
- les symboles fonctionnels d'arité quelconque qui permettent de construire des objets mais qui ne peuvent pas être interprétés (ex fonction "succ" ou "+")
- les symboles de prédicats (ex : $<$, $>$, $=$, ...)
- les symboles de quantifications $\forall$, $\exists$

Pour définir la notion de formule, on définit d'abord celle de terme :

DÉFINITION 5 "TERME"

- L'ensemble des variables et des constantes sont des termes
- Soit f, un symbole de fonction n-aire, et $t_1, \cdots t_n$ des termes alors $f(t_1, \cdots t_n)$ est un terme.

Toute entité obtenue par application des deux règles ci-dessus, un nombre quelconque de fois, est un terme. Un terme est une formule non interprétée, c'est à dire que l'on ne peut pas lui associer une valeur de vérité.

DÉFINITION 6 "FORMULE"

- Si P est un prédicat, et $t_1, \cdots t_n$ des termes, alors $P(t_1, t_2, \cdots t_n)$ est une formule.
- Si F et G sont des formules, $\neg F, (F \vee G), (F \wedge G), (F \rightarrow G), (F \leftrightarrow G)$ sont des formules.
- Si F est une formule et x une variable, $\forall x F, \exists x F$ sont des formules.

Toute entité obtenue par application des règles ci-dessus, un nombre quelconque de fois, est une formule.

Ces systèmes logiques ont un pouvoir d'expression beaucoup plus étendu que la logique des propositions, car il existe d'une part, des fonctions non interprétées qui permettent d'exprimer des relations entre les objets (ex appartenance, attribut, ...), et d'autre part, des quantificateurs qui donnent le pouvoir de qualifier tous les objets d'un domaine, ou d'en distinguer un. (ex : $\forall x\ cube(x) \rightarrow couleur(x, rouge)$, exprime que tous les cubes sont rouges). Cependant si ces logiques sont complètes elles ne sont pas en revanches décidables.

A.7 Logique modale

La logique modale représente un bon compromis entre la logique des propositions et celle des prédicats, car elle apporte un pouvoir expressif important, et elle reste complète et décidable. On s'intéressera plus particulièrement à l'une de ces théories, la logique temporelle, qui est adaptée à la spécification des systèmes temps réels.

Le langage de la logique modale est construit de la même façon que celui du calcul des propositions en y ajoutant la règle :
Si A est une formule alors $\Box A$ et $\Diamond A$ sont aussi des formules.

A.7 1. Définition informelle des opérateurs modaux

Si la définition mathématique d'un opérateur s'admet facilement, en revanche, la raison de son introduction et sa sémantique sont parfois obscures. Nous allons par la suite présenter les opérateurs modaux de façon un peu intuitive, en résumant et en interprétant des notions tirées d'un ouvrage d'Audureau, Enjalbert et Del Cerro [Audurau *et al.* 1990] qui présentent ces opérateurs de façon simple et pédagogique.

Selon ces derniers, l'introduction de ces nouveaux opérateurs remonte dès l'antiquité chez Aristote puis est reprise au Moyen-âge. Mais c'est Lewis [Lewis 1912], qui dans l'époque moderne sera le premier à leur donner un fondement formel. Selon Lewis, l'implication traditionnelle, dénotée par le symbole "$\rightarrow$", était insatisfaisante notamment à cause des paradoxes qu'elle introduisait comme par exemple le paradoxe de l'implication matérielle :

$$A \rightarrow (B \rightarrow A)$$

qui signifie "Si A est vrai, alors A peut être déduit de n'importe quoi". Lewis introduit alors un nouveau type d'implication appelé implication stricte dénotée ">", pour lequel, ce paradoxe disparaît, c'est à dire pour lequel la formule $A > (B > A)$ n'est plus valide.

$$A > B$$

signifie alors, "Lorsque A est vraie, alors B découle de A". La sémantique de cet opérateur (à travers l'adverbe lorsque) fait alors apparaître une notion nouvelle, celle de temporalité, ou encore appellée notion de "monde".

Aujourd'hui, la notation de Lewis a disparu au profit de deux opérateurs dual, $\Box$ et $\Diamond$ qui sont tel que $\Box A =_{def} \neg\Diamond\neg A$. $\Box$ signifiant "toujours : dans tous les mondes possibles", et $\Diamond$ signifiant "possible : il existe un monde où ...".

Ces nouveaux opérateurs sont définis par rapport à ">" :

$$(A > B) =_{def} \neg\Diamond(A \vee \neg B) =_{def} \Box(A \rightarrow B)$$

Cette définition, se lit alors "Il est impossible que A soit vrai et B faux" ou bien "toujours (dans tous les cas), A implique B". Cette notion de temporalité, introduite dans la syntaxe par des adverbes temporels, s'est traduite au niveau de la syntaxe formelle, par l'introduction de la notion de "mondes possibles" et de relation d'accessibilité entre les mondes qui est en fait, un graphe d'états, où chaque état (monde) représente une interprétation classique, c'est à dire une fonction d'évaluation des variables et des fonctions logiques.

Ainsi, considérons une triste journée d'hiver, où la formule "Il pleut" s'avère exacte, que nous noterons p. Il est clair, d'après le sens de $\Box$ que nous venons de décrire que $\Box p$ est heureusement une formule fausse, par contre $\Box(p \wedge \neg p)$ est vraie; c'est en effet une vérité logique.

Par contre, si l'on considère un ensemble de situations météorologiques (mondes) reliés par un graphe, si l'on se place dans un de ces mondes (correspondant à une situation météoroloique typiquement Bretonne par exemple), on peut imaginer que la formule $\Box p$ soit vraie, c'est à dire que dans l'ensemble des mondes accessibles à partir de cet état, p soit vraie. C'est à dire que dans le graphe reliant les conditions météorologiques, il n'existe pas d'état accessible (pour la Bretagne) où la propriété "jour sans pluie" soit vraie, sachant par ailleurs que des états, représentant les situations météorologiques du sud de la France, bien que non accessibles, à partir de l'état choisi peuvent satisfaire cette propriété.

Cet exemple met en lumière le sens profond des opérateurs modaux dont l'interprétation, est liée à un ensemble de mondes accessibles et à la notion de monde. Ils introduisent en fait, une notion intermédiaire entre vérité de fait (vérité liée à une interprétation) et vérité logique.

Abandonnons l'interprétation de p et considérons maintenant que p est une vérité logique, que l'on note par $\vdash p$ alors on a $\Box p$, c'est à dire que quelque soit un graphe représentant l'ensemble des interprétations, p est vraie dans tous les états accessibles, car elle est vraie dans tous les états. Par contre l'inverse n'est pas vraie comme nous venons de l'illustrer (si $\Box p$ est vraie, p n'est pas forcément valide, et il peut exister un monde où cette propriété soit fausse). Ce qui fait que l'on aura dans ces logiques une nouvelle règle d'inférence : "Si A est une vérité de fait, ce qui se note par $\vdash A$ Alors $\Box A$ est aussi une vérité de fait, ce qui se note $\vdash \Box A$. Cette expression se note conventionnellement de la manie're suivante :

$$\frac{\vdash A}{\vdash \Box A}$$

(l'inverse n'est ni une règle d'inférence ni un théorème)

A.7 2. Définition formelle

Un modèle d'une théorie de la logique modale est un graphe, c'est à dire un triplet

$$M =< W, R, m >$$

où

- W est l'ensemble des états du graphe

- R une relation binaire entre les états de W donnant tous les arcs du graphe,
- m la fonction d'interprétation, qui décrit pour un état donné l'ensemble des propositions vraies.

DÉFINITION 7 "MODÈLE"
On dit qu'un état ou qu'un monde w du modèle M, satisfait A, et on le note

$$M, w \models A$$

Cette relation de satisfaisabilité peut être définie par les clauses :
$M, w \models p$ **ssi** $p \in m(w)$
$M, w \models \neg A$ **ssi non** $M, w \models A$
$M, w \models A \vee B$ **ssi** $M, w \models A$ **ou** $M, w \models B$
$M, w \models \Box A$ **ssi** $\forall w'/wRw' \rightarrow M, w' \models A$
$M, w \models \Diamond A$ **ssi** $\exists w'/wRw' \wedge M, w' \models A$

En logique modale, il existe plusieurs théories identifiées par leur ensemble d'axiomes. Ceux-ci ont une influence sur les propriétés du graphe, et notamment sur la relation d'accessibilité. Par exemple, la théorie S_4, se définit par les axiomes suivant :

- les axiomes de la logique propositionnelle
- $\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$
- $\Box A \rightarrow A$
- $\Box A \rightarrow \Box\Box A$

Examinons les propriétés que ces axiomes imposent sur la relation d'accessibilité R du graphe du modèle : le troisième axiome impose à R d'être réflexive (si A est vraie dans tous les mondes accessibles alors elle est vraie dans l'état de départ), tandis que le quatrième lui impose la transitivité (si A est vraie dans tous les états directement accessibles, alors elle est vraie dans tous les états, accessibles depuis les états accédés).

S_4 est une théorie complète et décidable, et la logique temporelle que nous allons présenter comme une des techniques de spécification des systèmes temps réel est basée sur les mêmes axiomes.

A.7 2.1. Logique Temporelle

Dans la logique temporelle, la notion de monde correspond à celle de date, et la fonction d'accessibilité à celle de succession de dates. Cette relation est donc réflexive (si l'on considère que le présent appartient au passé et au futur), transitive, et antisymétrique comme S_4.

En général on y ajoute des axiomes pour donner au temps des caractéristiques, et on peut ainsi exprimer à l'aide du langage, que le temps a une origine, que le temps est ramifié ou linéaire, ou enfin, que le temps est discret ou continu.

BIBLIOGRAPHIE

[Aho 1974] A.V. Aho et S.C. Johnson. "*Programming Utilities and Libraries LR parsing*". Computing Survey, 1974.

[Alembert 1800] Alembert. "*Oeuvres d'Alembert, tome XIV*". 1800.

[Andre 1991] C. Andre et L. Fancelli. "*Etude d'une re'alisation mixte (Asyncrone/Synchrone) d'un syste'me temps-re'el*". APII, vol. 25, pp. 109–140, 1991.

[Antoine 1994] J. Antoine. "*Pardoxes et représentation du temps dans les Statecharts*". Probatoire C.N.A.M., 1994.

[Audurau *et al.* 1990] E. Audurau, P. Enjalbert et L. Fari n as Del Cerro. "*Logique temporelle sémantique et validation de programmes parallèles*". Masson, 1990.

[Bar-Hillel *et al.* 1961] Bar-Hillel, Y.M. Perles et E. Shamir. "*On formal properties of simple phrase structure grammars*". Z. Phonetik. Sprachwiss. Kommunikationsforsch, vol. 14, pp. 143–172, 1961.

[Benveniste 1991] A. Benveniste et G.Berry. "*The Synchronous Approach to Reactive and Real-Time Systems*". Rapport 581, IRISA, 1991.

[Bernstein 1981] A. Bernstein et P.K. Harter. "*Proving real-time properties of programs with temporal logic*". ACM, éditeur, 8th ACM Operating System Principles, pp. 1–11, 1981.

[Berry *et al.* 1987] G. Berry, P. Couronne et G. Gonthier. "*Synchronous Programming of Reactive Systems : an Introduction to Esterel*". Rapport, INRIA, 1987.

[Berry 1989] G. Berry. "*Real Time Programming: Special Purpose or General Purpose Languages*". IFIP 89 World Computer Congress, San Fransisco, 1989.

[Billaut 1993] Jean Charles Billaut. "*Prise en compte des ressources multiples et des temps de préparation dans les problèmes d'ordonnancement temps réel*". Thèse de Doctorat, LAAS, 1993.

[Bolognesi 1988] T. Bolognesi et E. Brinksma. "*Introduction to the ISO Specification Language LOTOS*". Computer Network and ISDN Systems, vol. 14, pp. 25–59, 1988.

[Bourey *et al.* 1989] J.P. Bourey, E. Castelain, J.C Gentina et M. Kapusta. "*CASPAIM : A computer aided design of the control system of FMS*". IMACS, éditeur, Annals on Computing and Applied Mathematics, Paris, 1989.

[Brams 1982] G. W. Brams. "*Réseaux de Petri : Théorique et Pratique. Tome 1 théorie et analyse*". Editions Masson, 1982.

[Brams 1983] G. W. Brams. "*Réseaux de Petri : Théorique et Pratique, Tome 2 Modélisation applications*". Editions Masson, 1983.

[Chomsky 1956] N. Chomsky. "*Three models for the description of language*". IRE transaction on Information Theory, vol. 2, n°. 3, pp. 113–124, 1956.

[Chomsky 1959] N. Chomsky. "*On certain formal properties of grammar*". Information and control, vol. 2, n°. 2, pp. 137–167, 1959.

[Courtiat *et al.* 1993] J.P. Courtiat, M.S. De Camargo et D.E. Saïdouni. "*RT-LOTOS : A Time extension of LOTOS for the Specification for Real-Time Systems*". Rapport 93158, LAAS-CNRS, May 1993.

[Dasarathy 1985] B. Dasarathy. "*Timing constraints of real-time systems: construct for expressing them, methods of validating them*". IEEE transaction Software Engineering, vol. 11, pp. 80–86, 1985.

[de Bonneval 1993] Agnan de Bonneval. "*Mécanismes de reprise dans les systèmes de commande à événements discrets*". Thèse de Doctorat, LAAS, 1993.

[de Michiel 1994] M. de Michiel. "*Recherche de la Configuration Optimisée d'une Architecture Cible pour une Application Temps Réel*". Thèse de Doctorat, UPS-IRIT, 1994.

[Delfieu *et al.* 1993] D. Delfieu, J.C Gentina, L. KERMAD, J.P. Maik, R. MOISAND et A.E.K. Sahraoui. "*Integration of Operating Modes in the Control of Flexible Manufacturing Systems Combining Synchronous and Asynchronous Approaches*". IEEE, éditeur, IEEE/SMC'93, 1993.

[Delfieu 1990] D. Delfieu et F. Abarca del Rio. "*Conception et realisation d'une interface graphique pour une application ARMOR*". Rapport, Irit, 1990.

[Delfieu 1993] D. Delfieu et A.E.K. Sahraoui. "*Expression and verification of temporal constraints for real-time systems*". IEEE, éditeur, 7th Annual European Computer Conference, 1993.

[Delfieu 1994a] D. Delfieu et A.E.K. Sahraoui. "*Automata timming specification*". INRIA, éditeur, 11 th International Conference on Analysis and Optimization of Systems, 1994.

[Delfieu 1994b] D. Delfieu et A.E.K. Sahraoui. "*Expression and validation of timing constraints and integration in software specification methods*". IEEE, éditeur, 2nd IEEE Workshop on Real-Time Applications, 1994.

[Dennis 1979] J.B. Dennis. "*The variety of data flow computers*". IEEE, éditeur, IEEE int. conf. on Distributed Systems, 1979.

[Diderot 1800] Diderot. "*Opinion des anciens philosophes*". 1800.

[Dours *et al.* 1992] D. Dours, M. De Michiel, L. Carcagno, R. Facca et P. Magnaud. "*Design method of Real Time design Systems*". IFAC-IFIP, éditeur, 18th Real-time Programming WRTP'92, Bruges, 1992.

[Ehrig 1985] H. Ehrig et B. Mahr. "*ACT ONE*". Springer Verlag, éditeur, Fundamentals of Algebraic Specification, Part 1, 1985.

[Elkhattabi 1993] S. Elkhattabi. "*Inte'gration de la surveillance de bas niveau dans la conception des syste'mes a' e've'nements discrets : application aux syste'mes de production flexibles*". Thèse de Doctorat, LAIL, 1993.

[Floyd 1967] R. W. Floyd. "*Assigning Meanings to Programs*". American Mathematical Society Symposium in Applied Mathematics, volume 19, pp. 19–31, 1967.

[Genrich 1981] H.J. Genrich et K. Lautenbach. "*System modelling with high level Petri nets*". Theorical Computer Science, vol. 13, pp. 109–136, 1981.

[Genrich 1987] H.J. Genrich. "*Predicate/transition nets*". Advances in Petri Nets, 1987.

[Harel *et al.* 1987] D. Harel, A. Pnueli, J.P. Schmidt et R. Sherman. "*On the formal Semantics of Statecharts*". New York: IEEE press, éditeur, 2nd IEEE symp. Logic in Computer Science, pp. 54–64, 1987.

[Harel *et al.* 1990] D. Harel, H. Lachover, A. Naamad abd A. Pneuli, M. Politi, R. Sherman, A. Shtull-Trauring et M. Trackhtenbrot. "*STATEMATE: A Working Environment for the Development of Complex Reactive Systems*". IEEE transaction on software engineering, vol. 16, n°. 4, pp. 403–414, Avril 1990.

[Harel 1985] D. Harel et A. Pnueli. "*On the development of Reactive Systems, in logic and Models of Concurrent systems*". NATO ASI Series in Computer Science, pp. 447–498, 1985.

[Harel 1987] Harel. "*Statecharts: a visual formalism for complex systems*". Science of Computer Programming, pp. 231–274, 1987.

[Harel 1988] Harel. "*On visual formalisms*". ACM, vol. 31, pp. 514–530, 1988.

[Hoare 1969] C.A.R. Hoare. "*An axiomatic basis for computer programming*". Communications of the ACM, vol. 12, pp. 576-580, 10 1969.

[Hopcroft 1979] J. E. Hopcroft et J. D. Ullman. "*Introduction to automata theory, languages and computation*", chapitre 1-2. Addison Wesley, 1979.

[Huising 1991] C. Huising et W.P. de Roever. "*Introduction to design choices in the semantics of Statecharts*". Information Processing Letters, vol. 37, n°. 4, pp. 205-213, 1991.

[Ingenierie 1988] Cisi Ingenierie. "*Esterel V3 Documentation*". Cisi, 1988.

[Jahanian 1986] F. Jahanian et A. K. Mok. "*Safety analysis of timing properties in real-time systems*". IEEE trans. software eng., vol. 12, pp. 890-904, 1986.

[Jensen 1987] K. Jensen. "*Coloured Petri nets*". Advances in Petri Nets, 1987.

[Larousse 1983] P. Larousse. "*Le petit Larousse illustré*". Larousse, 1983.

[Leduc 1992] G. Leduc et L. Léonard. "*A Timed LOTOS Supporting a Dense Time Domain and Including New Timed Operators*". 5th International Conference on Formal Description Techniques FORTE'92, 1992.

[Lewis 1912] C.I. Lewis. "*Implication and the algebra of logic*". Mind, pp. 522-523, 1912.

[Magnaud 1990] P. Magnaud. "*Me'thodologie et outil de conception d'une architecture paralle'le temps-re'el*". Thèse de Doctorat, IRIT, 1990.

[Marty 1994] J.C. Marty. "*Utilisation des Statecharts pour une spécification structurée du contrôle des cellules flexibles*". Thèse de Doctorat, LAAS, 1994.

[Merlin 1975] Merlin. "*Temporal petri net*". Rapport, Formal aspects of Computing, 1975.

[Milner 1980] R. Milner. "*a Calculus for Communicating Systems*". Computer Science, 1980.

[Pascal 1700] Pascal. "*Esprit Géométrique I*". 1700.

[Peraldi 1993] M.A. Peraldi. "*Conception et Re'alisation de Syste'mes Temps-Re'el par une Approche Synchrone*". Thèse de Doctorat, Sophia Antipolis, 1993.

[Pneuli 1989] A. Pneuli et M. Shalev. "*What is in a step ?*". J.W. De Backer, Liber Amicorum, pp. 373–400, 1989.

[Quemada *et al.* 1992] J. Quemada, D. de Frutos et Arturo Azcorra. "*TIC: A TImed Calculus*". Formal Aspects of Computing, 1992.

[Razouk 1989] R. Razouk et M. Gorlick. "*A Real-Time Interval Logic for Reasoning About Executions of Real-Time Programs* ". ACM, éditeur, 3th Symposium on Software Testing Analysing and Verification, pp. 10-19, 1989.

[Sahraoui 1988] A.E.K. Sahraoui. "*Timing constraints problems in real-time systems : a survey* ". Rapport 88346, LAAS, 1988.

[Sahraoui 1989] A.E.K. Sahraoui. "*What realness about time in real-time programming* ". IFIP/IFAC, éditeur, Workshop on Real-Time Programming, Berlin, 1989.

[Sahraoui 1992] A.E.K. Sahraoui et D. Delfieu. "*Zaman, a Simple Language For Expressing Timming Constraints* ". IBRA-BIRA, éditeur, 18th IFIP-IFAC workshop on Real-Time Programming : WRTP'92, pp. 19-24, 1992.

[Sahraoui 1994] A.E.K. Sahraoui. "*Some Timing Aspectsof Software Devclopment for Reactive Systems* ". Systems Software, vol. 25, pp. 51-57, 1994.

[Shostack 1979] R.E. Shostack. "*A practical procedure for arithmetic with function symbol*". ACM, vol. 26, n°. 2, pp. 351-360, 1979.

[Sibertin-Blanc 1988] C. Sibertin-Blanc. "*Le prototypage des applications interactives a' l'aide des re'seaux de Petri* ". Journe'es internationales : Le ge'nie logiciel et ses applications, Toulouse, Decembre 1988.

[Turski 1987] W. M. Turski et T. S. E. Mailbaum. "*The Specification of Computer Programs* ". Addison Wesley, 1987.

[Valette *et al.* 1982] R. Valette, M. Courvoisiser et D. Mayeux. "*Control of flexible production systems and Petri nets* ". Application and Theory of Petri nets, vol. 66, pp. 264-277, 1982.

[Valette 1990] R. Valette et M. Silva. "*Petri nets and Flexible Manufacturing* ". Lecture Notes in Computer Science, vol. 424, pp. 374-417, 1990.

[Vegdahl 1984] S. R. Vegdahl. "*A survey of proposed architectures fot the execution of functional languages* ". IEEE transaction on computers, vol. c-33, n°. 12, pp. 1050-1070, Decembre 1984.

Thèse de Monsieur David DELFIEU

Expression et validation de contraintes temporelles pour la spécification des systèmes réactifs

Résumé : La spécification des systèmes temps réel pose le problème de l'expression du temps et des contraintes temporelles pour lesquels nous proposons dans ce mémoire, une nouvelle représentation. Cette représentation est basée sur l'hypothèse que l'écoulement du temps est modélisé par l'occurrence d'un événement spécifique. Cette conception nous permet de considérer les contraintes temporelles comme des propriétés de séquences d'événements observables. En relation avec une notation simple, basée sur la notion de grammaire, les propriétés temporelles deviennent des propriétés syntaxiques. Ces considérations nous ont amenés à proposer une nouvelle méthode d'analyse des contraintes temporelles qui a pour objet de vérifier que toutes les contraintes temporelles d'un cahier des charges ont bien été prises en compte dans l'étape de spécification. Pour réaliser cette vérification, on élabore un analyseur de traces temporisées dont la construction se fait en deux étapes. On extrait d'abord les contraintes temporelles en suivant une décomposition structurée. Cette décomposition permet d'identifier un ensemble d'opérateurs (périodique, sporadique, de disjonction ou de conjonction) liant les contraintes temporelles. On exprime ensuite ces contraintes, sous la forme de grammaires "types". La seconde étape est la recomposition de ces grammaires, par l'ensemble des opérateurs précédemment identifiés. Pour cela, on a redéfini chacun de ces opérateurs pour qu'ils puissent s'appliquer sur des grammaires. Le résultat final de cette recomposition produit une grammaire globale qui constitue un analyseur syntaxique, capable de vérifier si une trace temporisée vérifie ou non, toutes les contraintes temporelles du cahier des charges.

Mots-clés : systèmes temps réel, spécification, contraintes temporelles, composition de contraintes temporelles, théorie des langages, grammaire, système à événements discrets.

Expression and validation of temporal constraints for the specification of reactive systems

Abstract : The specification of real time system raises problems of the expression of time and temporal constraints for which we suggest in this thesis a new representation. This representation is based on the hypothesis that time passing is modelized by the occurrence of a special event. This point of view allows us to consider temporal constraints as properties on observable event sequences. According to a simple notation based on grammar formalism, temporal properties become syntactic properties. These considerations have led us to propose a new approach for the analysis of temporal constraints which aim is to verify that all the temporal constraints of the requirements have been taken into account in the specification step. To realize this verification, we elaborate an analyzor of temporal sequences which building is composed of two steps. Fisrt, the extraction of temporal constraints is obtained following a structured decomposition, which allow to identify a set of operators (periodic, sporadic, of conjunction and disjunction) binding those constraints. We express then these constraints under grammar basical forms. The second step is the recomposition of all these grammars by the set previously identified. For this step we have redefined each of these operators, for their application on grammars. The final result of this composition will give a syntactic analyzor able to check if a temporized sequence verify all the temporal constraints of the requirements.

Keywords : real-time systems, specification, temporal constraints, composition of temporal constraints, language theory, grammar, discrete event systems.