De la composition de systèmes temporisés

Sébastien Bornot

▶ **To cite this version:**

Sébastien Bornot. De la composition de systèmes temporisés. Autre [cs.OH]. Université Joseph-Fourier - Grenoble I, 1998. Français. NNT : . tel-00004871

HAL Id: tel-00004871
https://theses.hal.science/tel-00004871

Submitted on 19 Feb 2004

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ JOSEPH FOURIER – GRENOBLE 1
SCIENCES ET GÉOGRAPHIE

THÈSE

pour obtenir le grade de
DOCTEUR DE L'UNIVERSITÉ JOSEPH FOURIER

DISCIPLINE : INFORMATIQUE

présentée et soutenue publiquement par

Sébastien BORNOT

le 15 décembre 1998

De la Composition des Systèmes Temporisés

Composition du jury :

André ARNOLD,	Rapporteur
Antoine PETIT,	Rapporteur
Jacques VOIRON,	Examinateur
Patrick COUSOT,	Examinateur
Joseph SIFAKIS,	Directeur de thèse

Table des matières

1 Introduction **5**
1.1 Systèmes de transitions 6
1.2 Systèmes temporisés et systèmes hybrides 6
1.3 Composition parallèle 9
1.4 Exposé du problème 10
1.5 Plan de la thèse 13

2 Spécification de l'urgence **15**
2.1 Présentation du problème 15
2.1.1 Le problème lui-même 15
2.1.2 Modélisation du temps 15
2.2 Les échéances 17
2.3 Les fonctions de délais possibles 19
2.4 La correspondance de Galois entre E et FDP 25
2.5 Traduction des échéances en FDP 30
2.5.1 Échéances bien définies 30
2.5.2 Traduction des échéances de forme modale 33
2.6 Conclusion de cette étude 39

3 Des systèmes hybrides : les SHEC **41**
3.1 Systèmes séquentiels 41
3.1.1 Une première temporisation 41
3.1.2 Opérateurs de choix 45
3.1.3 Choix consensuel 55
3.2 Compositions parallèles 59
3.2.1 Introduction 59
3.2.2 Composition de systèmes discrets 60
3.2.3 Composition des SHEC 61
3.2.4 Garde de la synchronisation 63
3.2.5 Synchronisation des échéances 64
3.2.6 Composition de plus de deux actions 66
3.3 Critique du modèle 67

4 Un modèle simplifié : les ATTT **69**
4.1 Pourquoi un autre modèle? 69
4.2 Transitions typées 70

4.3 Choix 72
4.4 Compositions parallèles 73
4.4.1 Modes de synchronisation 74
4.4.2 À propos des échéances 76
4.4.3 Réalisation de la composition 79
4.5 Synthèse 80
4.6 Présentation sous forme d'algèbre de processus 82
4.6.1 Équivalence sur les actions temporisées 83
4.6.2 Équivalence sur les processus 83

5 Conclusions et perspectives **85**
5.1 Objectifs 85
5.2 Résultats 85
5.3 Perspectives 86

Chapitre 1

Introduction

Si au cours de ces dernières années les programmes informatiques sont devenus de plus en plus sûrs, on est encore loin de la perfection. S'il est bien connu que cette perfection nous est inaccessible, nous chercherons à nous en rapprocher. Et ce d'autant plus que les systèmes sur lesquels nous travaillons sont de plus en plus complexes, et que les sources d'erreurs sont donc multipliées.

Cette complexité peut venir de différents facteurs. Tout d'abord, on peut vouloir augmenter la qualité des contrôleurs de certains systèmes extrêmement critiques nécessitant une réponse très rapide à certains événements. On regardera entre autre du côté des centrales nucléaires, ou du domaine spatial. Ensuite, les systèmes répartis se multiplient. Ainsi, dans le domaine de la téléphonie mobile, on gère un très grand nombre d'appareils qui peuvent se déplacer, se connecter ou se déconnecter du réseau. Parfois, les deux problèmes sont présents, comme dans les réseaux de transport. De nombreux appareils circulent et communiquent pour assurer un haut débit de circulation tout en évitant les collisions, mais il faut aussi pouvoir réagir aux accidents tels que des problèmes mécanique sur un véhicule, une voie impraticable, ou un problème de communication.

Pour valider ces programmes, et plus généralement, tous les programmes dits réactifs (qui réagissent à leur environnement), il faut être capable de modéliser aussi leur environnement. En effet il est nécessaire de confronter un programme de ce genre à son environnement pour évaluer son comportement, mais dans de nombreux cas on ne peut tester le programme sur un environnement réel, pour des raisons de coût ou de sécurité. Il reste alors à confronter le programme à un modèle de l'environnement. De plus en plus, on a alors besoin de modéliser le temps, qui est un paramètre important de cet environnement surtout s'il y a interactions entre plusieurs systèmes. Les interactions sont en général modélisées par des compositions parallèles, qui posent un certain nombre de problèmes lorsqu'elles sont utilisées avec des systèmes temporisés. Cette thèse propose un modèle de temporisation permettant de résoudre une partie de ces problèmes, les propriétés dites de progrès maximal et de réactivité temporelle étant notamment assurées par construction.

Avant de rentrer plus en détail dans le sujet, voyons quelques notions qui permettront de cerner les problèmes que nous chercherons à résoudre.

1.1 Systèmes de transitions

L'objet de base pour modéliser un système en informatique est le système de transition. Il s'agit d'un graphe orienté (au sens mathématique du terme), dont les sommets représentent les états du système et les arcs représentent les actions, ou transitions. Si l'on isole un état initial dans ce graphe, les différents chemins du graphe partant de cet état représentent les différentes évolutions possibles pour le système modélisé.

Ce modèle, s'il est extrêmement simple, est aussi extrêmement puissant puisqu'il permet de représenter un très grand nombre de systèmes. Des modèles plus évolués ont été définis, qui permettent de structurer les systèmes de transitions. Tous ces modèles ont en commun le fait que leur sémantique est définie sur les systèmes de transitions, et qu'ils permettent donc de représenter des sous-classes de ces systèmes de transitions. Parmi ces modèles, on remarquera entre autres, les réseaux de Petri, les automates et les algèbres de processus.

Pour des raisons de généralité nous travaillerons donc sur les systèmes de transitions (à part dans le chapitre 3, où nous nous placerons dans le cadre plus restreint des automates temporisés). Les résultats qui seront donnés ici pourront alors être appliqués à toute la gamme des modèles informatiques que nous venons de citer.

1.2 Systèmes temporisés et systèmes hybrides

L'un des principaux problèmes de ces modèles apparaît lorsque l'on veut spécifier des systèmes où le temps intervient. Cette situation est de plus en plus courante avec le développement des systèmes répartis, où les différentes parties du système sont obligées de se synchroniser pour fonctionner correctement. On la retrouve aussi lorsque le système nécessite une réponse rapide à certains événements.

Tout d'abord nous remarquerons que les problèmes temporels peuvent être vus différemment suivant que l'on cherche à programmer un système ou à le spécifier. En effet, lors de la programmation, on n'appréhendera le temps et les actions de l'environnement que par l'ordre dans lequel ils arrivent. On tiendra alors compte de la nature de ces événement pour fournir des données au système. En aucun cas on ne pourra poser de contrainte sur l'environnement, en particulier sur l'écoulement du temps.

À l'opposé, lors de la spécification de systèmes, nous avons un certain nombre d'informations sur l'environnement, en particulier une relation de causalité entre les événements est donnée. De plus, pour modéliser l'urgence de certaines actions on peut se permettre de stopper le temps, forçant ainsi les actions à s'exécuter. Pour introduire les incertitudes qui apparaîtraient dans l'environnement, on utilisera alors le non déterminisme, à savoir que si dans certains états plusieurs événements sont possibles, un seul sera effectué, mais cet événement est susceptible de changer d'une exécution du système sur l'autre. On aura ainsi un moyen de représenter différentes situations possibles (ou dif-

férentes évolutions de l'environnement) en un seul système.

D'autre part, deux approches sont possibles pour intégrer le temps dans un système. Lorsque l'on programme, on sait très bien que toute action a un certain délai d'exécution non nul. Mais lorsque l'on spécifie, on peut très bien considérer que les actions sont instantanées. Cette approche peut surprendre, mais elle est équivalente à la première en ce sens qu'une action est alors représentée comme un début d'exécution, une attente, puis une fin d'exécution. Si seul le résultat de l'action compte, on pourra représenter une action par sa durée et sa terminaison, ainsi $s_0 \xrightarrow{3} s_1 \xrightarrow{a} s_2$ peut représenter une action a de durée 3. Une autre vision de cette seconde approche est de considérer que le passage du temps représente une évolution du système suivant un comportement donné (par exemple pour un contrôleur de chaudière, le chauffage est en marche) et que les actions sont des changements de comportement (on allume ou éteint le chauffage). Quelle que soit l'optique envisagée, cette seconde approche est très pratique pour la spécification puisqu'elle permet de séparer très clairement les actions qui seront considérées comme instantanées et le passage du temps. Dans le cadre de cette étude, cela permettra aussi de se focaliser sur la gestion du temps dans les modèles. C'est pour cette raison que j'ai choisi cette approche.

Comme les systèmes non temporisés sont étudiés depuis plus longtemps, et que l'on sait mieux les décrire, l'habitude a été prise de baser les systèmes temporisés sur un squelette non temporisé que l'on habille ensuite selon des techniques de temporisations qui peuvent varier.

De nombreuses temporisations ont été effectuées sur les algèbres de processus et les automates. On parle habituellement de systèmes hybrides pour le cas général, et de systèmes temporisés dans le cas plus restreint où les seules grandeurs continues sont des horloges permettant de mesurer des laps de temps. Le terme temporisation sera néanmoins employé dans le sens le plus général dans la suite.

La plupart des temporisation est réalisée par l'adjonction de variables qui sont modifiées par le passage du temps et que l'on peut tester. Ceci permet de savoir quand on peut prendre une transition et quand on doit la prendre. Il y a donc deux types de conditions :

- des conditions, souvent appelées *gardes*, qui disent quand une action est possible. Les gardes sont donc des conditions qui restreignent les moments où les actions sont possibles.

- des conditions sur la progression du temps qui interdisent au temps de progresser à certains moments. Cela permet de simuler l'urgence d'une ou plusieurs actions, car en empêchant le temps de progresser on force les actions à être effectuées. Après avoir effectué une action, on change d'état, et le temps peut alors progresser à nouveau. Ces conditions n'ont donc pas pour but de bloquer le temps, mais de forcer des actions à être réalisées à certains moment, le temps devant pouvoir progresser en permanence.

Ce type de temporisation est celui adopté par les différents automates temporisés [NSY92, MP91, AD90], automates hybrides [NSY92, PV94, MMP92],

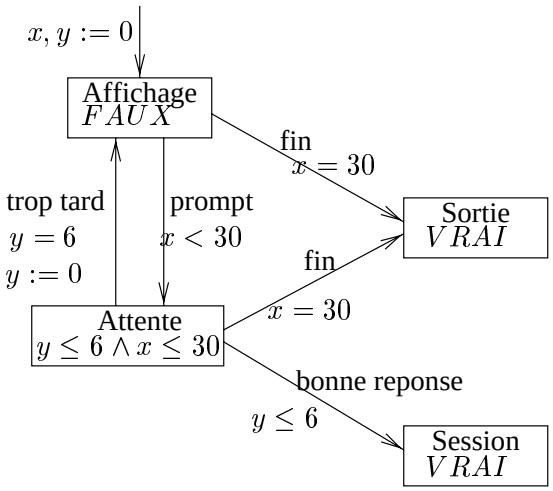


FIG. 1.1 – *Procédure de login*

réseaux de Petri temporisés [Ram74] et de nombreuses algèbres de processus temporisés (telles que ATP [NRSV90], ou ET-LOTOS [LL95]).

La figure 1.1 montre un système temporisé donné sous forme d'un automate temporisé. Il s'agit d'une procédure de login correspondant à la situation suivante: la procédure ne peut durer plus de 30 secondes, passé ce délai on sort et l'utilisateur ne peut se loger. Pendant les 30 secondes qui lui sont imparties, l'utilisateur a droit à un nombre d'essais indéfini, mais chaque essai est lui aussi limité dans le temps et ne peut durer plus de 6 secondes. Chaque essai se décompose en deux étapes: l'affichage d'un prompt, et la lecture de la réponse. La figure se lit comme suit: on a un graphe sur lequel on peut reconnaître la boucle représentant la succession des essais, et les deux sorties possibles. Les sommets du graphe sont appelés états de contrôle. Sur ce graphe, on a ajouté des conditions portant sur les horloges (ici x et y, dont nous verrons la signification ci-après) et des remises à zéro de ces horloges (notées $x := 0$ et $y := 0$). On reconnaîtra les gardes associées aux transitions et les conditions de progression du temps associées aux états. Dans cet exemple, x mesure le temps depuis le début de la procédure, pour savoir si les 30 secondes imparties ne sont pas écoulées, auquel cas on effectue l'action nommée fin. De même y mesure le temps écoulé pour chaque essai. On remarquera que la condition de progression du temps associée à l'état Sortie est $VRAI$, c'est-à-dire que l'on peut rester indéfiniment dans cet état, de même pour Session. Au contraire, l'état affichage ne correspond qu'à un point de contrôle où l'on ne peut attendre, la condition associée est alors $FAUX$. Par contre dans cet état, suivant la valeur de x on prendra soit la transition fin soit la transition prompt.

Nous reviendrons ultérieurement sur cette manière de décrire des systèmes réactifs.

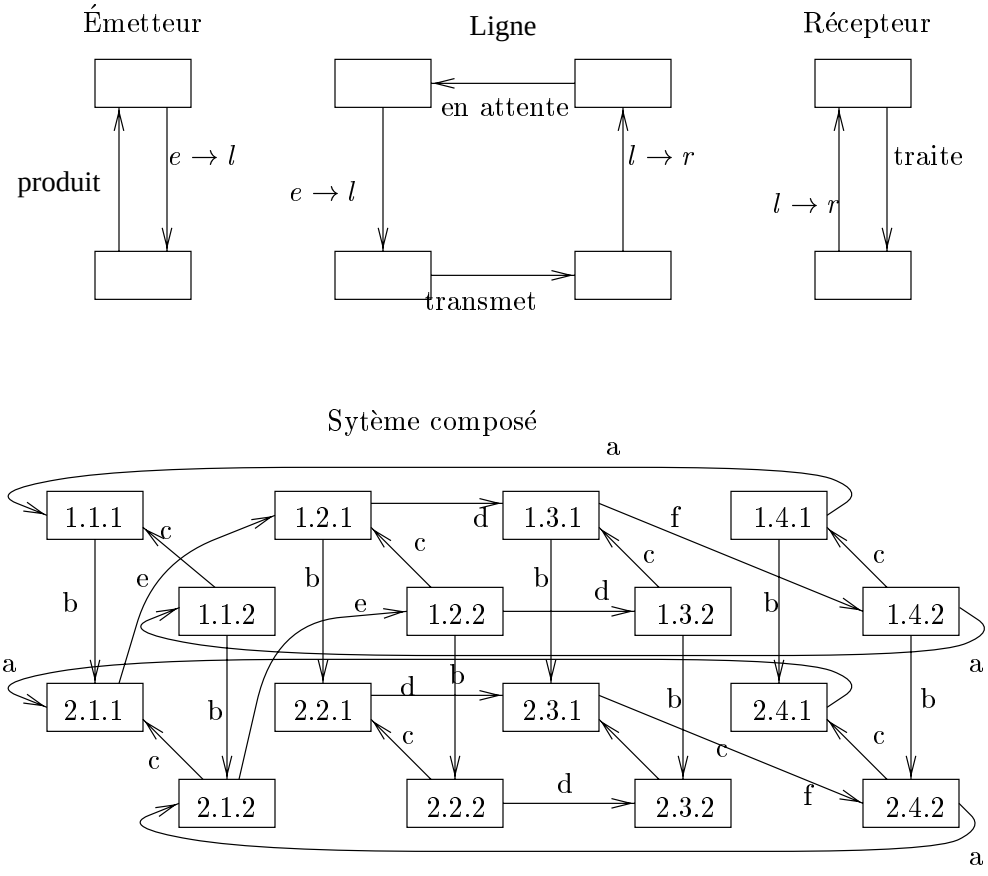


FIG. 1.2 – *Un exemple simple : émetteur - récepteur*

1.3 Composition parallèle

On assiste au développement des systèmes distribués, c'est-à-dire formé d'un certain nombre de composants, lesquels sont répartis sur plusieurs sites et évoluent en parallèle tout en communicant entre eux pour se synchroniser ou transmettre des données. Pour représenter cela, on utilise en général des compositions parallèles. L'opérateur de composition parallèle [Mil80] permet de regrouper plusieurs systèmes simples pour donner un seul système qui représente l'ensemble des comportements des petits systèmes mis en présence les uns des autres.

Cet opérateur présente plusieurs avantages. Tout d'abord, il est plus facile de représenter deux petits systèmes qu'un gros. Ceci est illustré sur l'exemple suivant.

La figure 1.2 nous montre un exemple composé de trois systèmes, un émetteur, une ligne de transmission et un récepteur. Ce système est très simplifié. Le comportement de la ligne est modélisé comme suit. Dans l'état 1 la ligne attend un message de l'émetteur qui la fera passer dans l'état 2. Le message est

alors transmis à l'autre bout de la ligne, celle-ci arrive dans l'état 3. Elle délivre alors le message au récepteur et passe dans l'état 4. Finalement, elle émet un signal indiquant qu'elle est à nouveau prête à transmettre. L'émetteur est plus simple, il produit un message et le donne à la ligne. De même le récepteur reçoit le message de la ligne et effectue ensuite un certain traitement sur ce message. Pour compléter le système, nous imposerons des contraintes de synchronisation : l'émetteur délivre le message à la ligne au moment où celle-ci le reçoit, de même que la ligne délivre le message au récepteur au moment où celui-ci le reçoit. Le système composé est alors formé de 16 états et 28 transitions. Ce système, pourtant relativement simple, commence à devenir illisible.

Si la composition parallèle est apparue pour représenter des systèmes distribués, elle s'est développée pour d'autres applications. Il existe aujourd'hui des langages de programmation, tel ET-LOTOS ([LL95] permettant de décrire un système complexe comme composition parallèle de systèmes plus simples, même dans le cas de systèmes non distribués. Outre la facilité apportée par une telle description, on gagne en place mémoire. En effet, la description par composition parallèle n'oblige pas à calculer le système composé (qui est proche du produit cartésien des composants), mais permet de travailler directement sur les composants et leurs interactions. Cette approche est aussi utilisée lorsque l'on veut effectuer certains traitements à ces systèmes complexes, par exemple les vérifier. On procède alors avec les techniques dites *à la volée* [VW86].

L'opérateur de composition parallèle apparaît alors comme un outil indispensable pour construire ou vérifier des systèmes.

1.4 Exposé du problème

Étant donnés deux systèmes S_1 et S_2 non temporisés, on sait effectuer leur composition parallèle $(S_1 \parallel S_2)$ et l'on sait temporiser chacun des trois systèmes (pour obtenir S_1^t, S_2^t et $(S_1 \parallel S_2)^t$). Ce que l'on aimerait c'est avoir une composition parallèle sur les systèmes temporisés ($\parallel^t$) qui rende le diagramme de la figure 1.3 commutatif.

Or ceci n'est pas réalisable avec la composition parallèle classique. En effet, la composition parallèle sur les systèmes non temporisés correspond à une spécification à base de processus communicants. Il s'agit de processus complètement séparés qui ne communiquent qu'au travers de certaines synchronisations, pour échanger des données ou coordonner leurs actions. Lorsque l'on cherche à étendre cette composition aux systèmes temporisés, on se heurte au fait que le temps (physique) est une donnée partagée, même si cette donnée n'apparaît que pour indiquer comment évoluent les autres données, en particulier les horloges. Il est alors facile de faire apparaître des blocages : le système ne peut ni effectuer d'action, ni laisser passer le temps, comme c'est le cas sur l'exemple suivant issu de [SY96].

Cet exemple (figure 1.4), qui représente un système formé d'un producteur qui met à disposition le produit pendant une durée limitée avant de recommencer à produire, et d'un consommateur qui cherche un produit pendant un

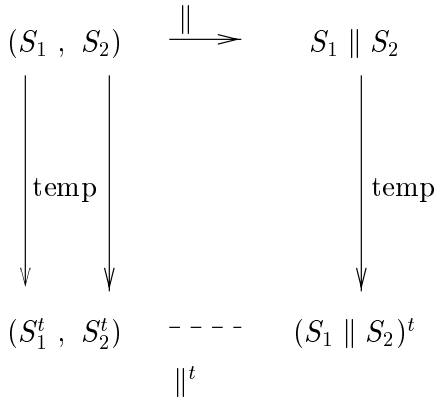


FIG. 1.3 – *Composition parallèle des systèmes temporisés*

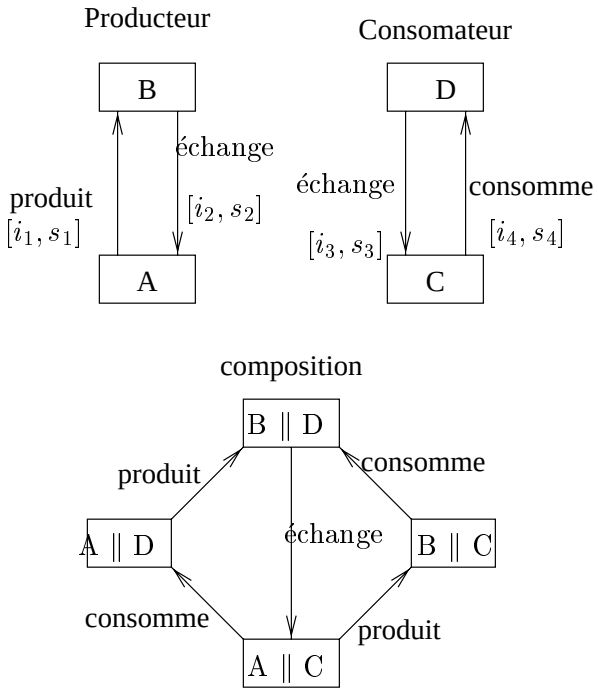


FIG. 1.4 – *Producteur - Consommateur*

certain temps puis le consomme. Les intervalles associés aux transitions indiquent à quels moments les transitions peuvent être prises, le temps écoulé dans l'état de contrôle devant appartenir à l'intervalle donné. Ainsi, pour le producteur, en partant de l'état A, on peut effectuer la transition *produit* en ayant attendu au moins i_1 unités de temps, on devra avoir pris cette transition avant s_1. Après avoir effectué cette transition on arrive dans l'état B, où il faudra attendre entre i_2 et s_2 pour effectuer la transition *échange*.

Le producteur et le consommateur doivent se synchroniser sur l'échange du produit (transition *échange*), les actions de production et de consommation étant indépendantes. Le squelette de la composition est donné figure 1.4. Le calcul de l'intersection des comportements peut introduire des blocages, c'est à dire que le système est dans un état ou non seulement il ne peut plus effectuer de transition, mais en plus il ne peut pas laisser passer le temps. Pour certaines valeurs des intervalles, il est même impossible d'éviter les blocages, car les deux systèmes seront complètement désynchronisés.

Regardons ce qui se passe pour les valeurs suivantes : $i_1 = 1$, $s_1 = 3$, $i_2 = 2$, $s_2 = 5$, $i_3 = 2$, $s_3 = 4$, $i_4 = 3$ et $s_4 = 4$. Dans certaines conditions le système peut fonctionner. Par exemple, en partant de l'état $A \parallel C$, les transitions suivantes peuvent être prises :

$$\xrightarrow{2} \xrightarrow{produit} \xrightarrow{1} \xrightarrow{consomme} \xrightarrow{3} \xrightarrow{echange} \ldots$$

En effet, ayant attendu entre 1 et 3 on peut effectuer la transition *produit* après quoi le système se trouve dans l'état de contrôle $B \parallel C$, le consommateur n'a pas changé d'état de contrôle. Le temps de référence pour savoir si l'on peut prendre la transition *consomme* est donc le délai écoulé depuis le début de l'exécution. En attendant encore 1 unité de temps on aura donc un délai total de 3 qui est compris entre i_3 et s_3, on peut donc effectuer *consomme* pour se trouver dans l'état $B \parallel D$. Ensuite si l'on attend 3 unités de temps supplémentaires, on sera depuis 4 unités de temps dans B, ce qui permet d'effectuer *échange*, et depuis 3 dans D, ce qui permet également d'effectuer *échange*. Ces transitions *échange* étant autorisée simultanément dans les deux composants, leur composition est autorisée, on peut ainsi continuer avec les mêmes délais.

Par contre, les mêmes valeurs pour les intervalles induisent un blocage si l'on commence par la séquence suivante, toujours à partir de $A|C$:

$$\xrightarrow{1} \xrightarrow{produit} \xrightarrow{3} \xrightarrow{consomme} \xrightarrow{2}$$

Cette séquence est valide au sens où chaque transition peut être effectuée suivant les contraintes définies ci-dessus. Par contre, elle conduit à un état où le temps ne peut plus progresser : on est dans B depuis 5 unités de temps, ce qui est le maximum autorisé avant de prendre *échange*, et dans D depuis 2, ce qui est inférieur à la limite inférieure i_4 pour effectuer *échange*. Les deux transitions *échange* ne pouvant se réaliser simultanément, leur synchronisation ne peut se faire en ce point. On est donc dans un état où ni action, ni passage du temps ne sont possibles.

Avec d'autres valeurs pour les intervalles, on réalise facilement des systèmes se bloquant forcément. Or le blocage est quelque chose qui nous semble irréaliste, puisqu'en aucun cas dans la réalité on ne peut empêcher le temps de passer. On cherchera donc à produire des systèmes qui peuvent toujours laisser passer le temps, si besoin après avoir effectué des transitions discrètes (actions).

Une solution possible serait d'autoriser les deux transitions *échange* à s'effectuer séparément, mais alors un autre problème se pose : ces transitions entrent en concurrence avec la synchronisation. La propriété que nous appellerons de *progrès maximal* dit que dans ce cas la synchronisation doit être favorisée. Pour réaliser le progrès maximal on ajoute souvent un opérateur de masquage, permettant de supprimer les transitions indésirables. Nous chercherons à nous passer de cet opérateur, ou plus exactement à l'intégrer à la composition parallèle. Le but de cette thèse est de réaliser un système satisfaisant la condition de progrès maximal, et temporellement réactif (c'est-à-dire ne bloquant jamais totalement le système).

1.5 Plan de la thèse

Pour résoudre le problème envisagé, nous commencerons par une étude de la temporisation dans le chapitre 2. Nous en tirerons des conclusions quant à la forme que peuvent prendre les contraintes temporelles. Habituellement les contraintes régissant la progression du temps sont liées aux états de contrôle. Nous verrons que pour des raisons de spécification, et pour des raisons de flexibilité dans la composition parallèle, nous attacherons ces contraintes aux transitions. Ceci semble naturel aussi par le fait que le but premier de ces contraintes est de caractériser l'urgence des transitions. L'étude portera donc sur ces contraintes, que nous appellerons *échéances*.

Nous verrons au chapitre 3 un modèle général (les SHEC) satisfaisant les propriétés de réactivité temporelle et de progrès maximal. La réactivité temporelle est une version affaiblie de la propriété de non-blocage, mais elle est associée à chaque transition séparément; la propriété de non-blocage exigerait au contraire d'examiner simultanément toutes les transitions. Le progrès maximal est assuré par un opérateur de choix particulier : le choix avec priorités qui permet de favoriser certaines transitions par rapport à d'autres. Ce modèle est très puissant, mais nous verrons qu'il n'est pas utilisable dans la pratique : une grande gamme d'opérateurs aussi bien pour le choix que pour la composition parallèle sont disponibles, mais les interactions entre différents opérateurs sont parfois difficiles à prévoir. Par contre, si l'on se restreint à quelques opérateurs, on peut avoir une puissance d'expression très grande et simultanément une grande facilité d'utilisation.

Le chapitre 4 présente l'un de ces modèles, que nous appellerons les Automates Temporisés à Transitions Typées. Dans ce modèle, nous ne garderons qu'un seul opérateur de choix (le choix avec priorités) et nous utiliserons trois modes de synchronisation ET (synchronisation classique), MAX (synchronisation avec attente) et MIN (interruption). De plus nous verrons que nous n'utiliserons pas directement les échéances, mais des *types* permettant de spécifier

l'urgence d'une transition en calculant son échéance à partir de sa garde.

Enfin, nous ferons le point sur le travail effectué au chapitre 5.

Chapitre 2

Spécification de l'urgence

2.1 Présentation du problème

2.1.1 Le problème lui-même

Jusqu'à présent la manière de gérer le passage du temps dans les systèmes temporisés et hybrides a été très peu étudiée de façon théorique. En effet, dans ces systèmes il est nécessaire d'interdire au système d'évoluer continûment, ce qui revient à arrêter la progression du temps c'est-à-dire rendre urgentes certaines actions. A cet effet, l'utilisation de conditions liées aux états et spécifiant si la progression du temps est autorisée semble naturelle, mais elle devient insuffisante si l'on veut simuler le système. En effet un prédicat sur les états du système permet de savoir si le temps peut s'écouler dans un état donné mais ne permet pas de quantifier directement le laps de temps qui peut s'écouler. Pour passer du qualitatif au quantitatif on est obligé de calculer une fonction permettant de savoir si l'on peut attendre dans un état donné pendant une durée donnée. Cette fonction, que nous appellerons "fonction de délais possibles", est plus générale et pourrait être utilisée pour spécifier le comportement temporel d'un système si elle n'avait ce grave défaut d'être complexe à manipuler. Nous en sommes donc réduits à utiliser deux entités, l'une simple (les prédicats sur les états) et l'autre puissante (les fonction de délais possibles) et à passer de l'une à l'autre. Nous étudierons donc dans cette partie ce que sont ces deux entités et comment elles sont liées.

2.1.2 Modélisation du temps

Nous allons étudier une méthode classique pour spécifier la progression du temps et l'urgence dans les systèmes hybrides et temporisés. Il s'agit de la méthode consistant à ajouter des variables à un système discret, ces variables évoluant avec le passage du temps, le comportement discret du système étant alors limité par des contraintes sur les variables.

Les états du système sont donc définis par un couple (état discret, valuation des variables). Nous décrirons le temps comme une grandeur réelle, c'est-à-dire que toute progression du temps sera quantifiée par un nombre réel. C'est cette représentation du temps par les réels qui nous permettra de modéliser des phé-

nomènes continus.

Nous étudierons certains aspects de la spécification des comportements temporels d'un système hybride. Ces systèmes peuvent évoluer de façon continue et changer de comportement par des actions discrètes. Il est donc important de savoir quand le système peut évoluer et dans ce cas comment il évolue. Les deux aspects de l'évolution (quand, comment) peuvent être rassemblés en une seule fonction ou séparés en deux. C'est cette deuxième approche que nous utiliserons; en effet, nous porterons le plus gros de notre étude sur la spécification de l'urgence.

L'évolution du système avec le passage du temps est généralement donnée par l'utilisation d'équations différentielles avec des dérivées par rapport au temps. Nous supposerons ici que les équations sont résolues et que la solution est unique, c'est-à-dire que l'on peut prédire l'état qui sera atteint en laissant passer le temps. Ceci nous permet de définir les fonctions d'évolution.

Définition 1 *Fonction d'évolution*
L'évolution du système sera donnée par une fonction $\triangleright$ *qui à un état* (s, v) *et une durée* t *associe l'état atteint à partir de* (s, v) *en laissant s'écouler* t *unités de temps.* $\triangleright$ *est donc une fonction de* $(S \times V) \times \mathbf{R}_+$ *dans* $(S \times V)$*, où* V *est l'ensemble des valuations et* S *l'ensemble des états.*

Notation 1 *Pour plus de simplicité on notera* $(s, v) \triangleright t$ *l'image de* $((s, v), t)$ *par* $\triangleright$*. De plus, on séparera la loi d'évolution en lois d'évolution locales à chaque état de contrôle,* $\triangleright_s : V \times \mathbf{R}_+ \to V$ *qui donne l'évolution dans chaque état de contrôle.*

Remarque 1
Cette séparation en lois d'évolutions locales est justifiée par le fait que le passage du temps n'autorise que des évolutions continues du système et donc interdit de changer d'état de contrôle.

On supposera de plus que pour tout état (s, v), la fonction $t \mapsto v \triangleright_s t$ est définie sur $\mathbf{R}_+$. Cela signifie que l'on donne une valeur à tous les successeurs temporels d'un état donné, indépendamment du fait que ces successeurs soient accessibles ou non. Ceci peut surprendre, mais cette technique ne change rien par rapport à celle consistant à ne donner de valeurs qu'aux successeurs accessibles, puisque celles-ci ne seront pas prises en compte lors de l'exécution du système. Par contre nous verrons par la suite qu'il peut être très utile de connaître l'évolution "spontanée" du système lorsque l'on oblige le temps à s'écouler en interdisant toute action discrète.

Pour clore le sujet des fonctions d'évolution nous remarquerons que du fait du déterminisme du temps elles doivent satisfaire la propriété suivante :

Définition 2 *Déterminisme temporel*
On dit qu'une fonction $f : V \times \mathbf{R}_+ \to V$ *est déterministe temporellement lorsqu'elle satisfait l'égalité suivante :*

$$\forall v \in V, \forall t_1, t_2 \in \mathbf{R}_+ \ . \ f(f(v, t_1), t_2) = f(v, t_1 + t_2)$$

Pour une loi d'évolution cela signifie simplement que l'évolution est la même en laissant passer une durée t_1 puis une durée t_2 ou en laissant passer une seule durée $t_1 + t_2$.

> **Remarque 2** Nous utiliserons des lois d'évolutions $\triangleright_s$ déterministe temporellement. On en déduit que $(v \triangleright_s t) \triangleright_s 0 = v \triangleright t$, pour tout v et t. Nous généraliserons cette propriété en exigeant que $v \triangleright_s 0 = v$ même si v n'est successeur temporel d'aucune valuation. Cela revient à dire qu'attendre une durée 0 ne change pas l'état, ce qui est normal puisque attendre pendant une durée 0 revient à ne pas évoluer.

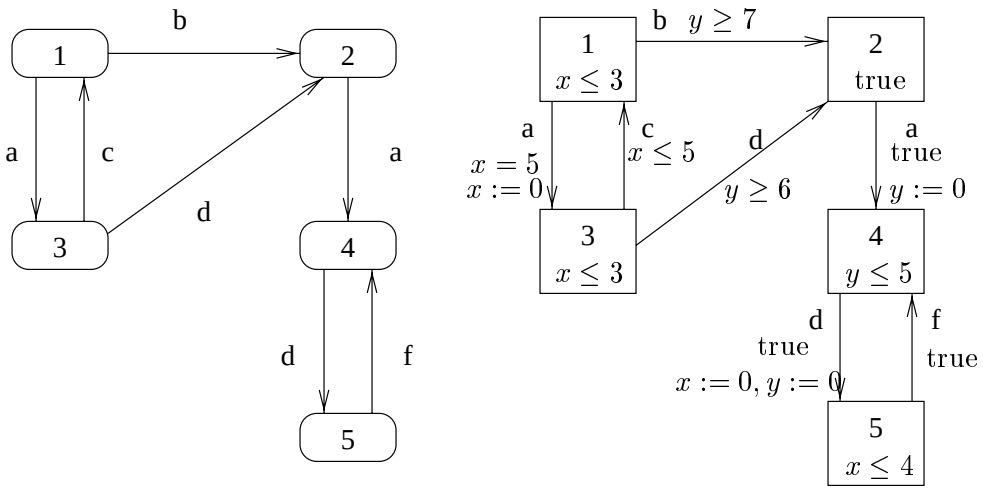
Pour simplifier les notations par la suite, nous utiliserons souvent le symbole $\triangleright$ pour une fonction d'évolution locale à un état de contrôle lorsqu'il n'y a pas d'ambiguïté sur cet état de contrôle ou si cet état de contrôle est insignifiant dans le contexte étudié. Par abus nous parlerons aussi de fonction d'évolution dans les mêmes conditions, sans préciser qu'elle est locale. La différence entre une fonction d'évolution globale et une fonction d'évolution locale est facile à remarquer puisque la première s'applique à un état complet alors que la seconde ne s'applique qu'à la partie continue de l'état, la valuation.

2.2 Les échéances

Classiquement on utilise des conditions de progression du temps liées aux états pour spécifier si l'attente est possible dans un système. Les conditions de progression du temps spécifient les états accessibles par passage du temps, on doit donc prohiber toute attente menant dans un état qui ne satisfait pas la condition. Ceci est bien sûr lié à la volonté de spécifier l'urgence d'une action.

Les automates temporisés, qui sont des automates étendus par des horloges, sont ainsi définis de la manière suivante. Partant d'un automate discret, on ajoutera des contraintes de temps (cf figure 2.1). Les variables x et y sont des horloges qui mesurent le temps écoulé depuis leur dernière remise à zéro (indiquée par $x := 0$ et $y := 0$), ou depuis le début de l'exécution du système. Un état du système est alors défini par un état de contrôle (un état de l'automate discret), et un état des horloges. La contrainte sur un état de contrôle, appelée condition de progression du temps, indique les états des horloges possibles dans cet état de contrôle. Les contraintes sur les transitions, appelées gardes, indiquent les états des horloges pour lesquels ces transitions sont autorisées. Cela permet de limiter les moments où les transitions peuvent être effectuées.

On remarquera que les conditions de progression du temps, qui limitent le passage du temps dans cet état de contrôle, peuvent être données indépendamment des actions possibles depuis cet état de contrôle. Il est alors facile de bloquer le temps quand aucune action n'est possible. On exigera donc de relier les conditions permettant de bloquer le temps aux conditions autorisant les actions. Il nous est alors paru naturel que si la gestion du passage du temps devait être liée aux actions, on pouvait définir deux conditions pour chaque action. La première, la *garde*, dit quand l'action est possible. La seconde, l'*échéance*, dit quand l'action est obligatoire. L'échéance est donc le complément d'une condition de progression du temps, en ce sens que lorsqu'elle est vraie le temps ne



Automate discret Automate temporisé

FIG. 2.1 – *Automates temporisés*

peut progresser. Nous reviendrons sur ce choix dans la partie suivante.

Nous utiliserons donc au cours de cette partie la convention suivante.

Définition 3 *Échéances*

Une échéance est un prédicat sur les valuations d'un système hybride, autrement dit une fonction de V dans $\{vrai, faux\}$. L'échéance spécifie les états à partir desquels le temps ne peut plus progresser, rendant une action urgente. Les états satisfaisant l'échéance ne sont pas tous inaccessibles : le temps ne doit s'arrêter que lorsque l'échéance est devenue vraie. Des échéances seront liées à chaque transition, mais une condition globale à chaque état de contrôle sera induite et permettra donc de connaître la progression du temps dans cet état de contrôle.

Remarque 3 Ce qu'il faut retenir de cette définition est que les échéances, liées aux actions, identifient les états où l'action est urgente et donc où le temps est arrêté. Ceci nous permet de calculer une condition, par exemple une condition de progression du temps, pour chaque état de contrôle. Ceci est illustré sur l'exemple suivant.

Exemple 1

La figure 2.2 nous montre un état s d'où partent deux transitions étiquetées par a et b; on supposera qu'il n'y a que deux variables. Les échéances sont les suivantes : a est urgente si $x > 4 \wedge y > 3$ et b est urgente si $x = 3 \vee x = 5$. Le temps ne peut progresser que si aucune des actions n'est urgente, donc si $\neg((x > 4 \wedge y > 3) \vee (x = 3 \vee x = 5))$. La condition de progression du temps d'un état de contrôle n'est donc que la négation de l'union des échéances des actions issues de cet état de contrôle.

Le choix qui a été fait que certains états satisfaisant l'échéance soient ac-

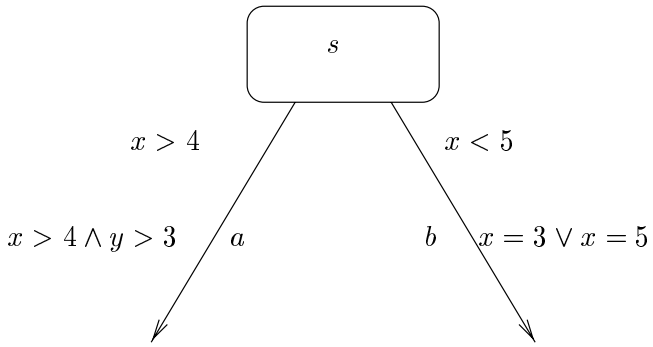


FIG. 2.2 – *Condition de progression du temps et échéances*

cessibles est discutable. Néanmoins, il permet de simplifier considérablement les modèles que nous étudierons par la suite. Nous verrons plus tard que si les échéances étaient inaccessibles, la définition d'un modèle temporellement réactif serait beaucoup plus complexe, et dépasserait le cadre de notre propos.

Dans toute la suite, nous utiliserons la dualité qui existe entre les prédicats sur un ensemble et les parties de cet ensemble : un prédicat peut être représenté par l'ensemble des éléments qui ont *vrai* pour image par ce prédicat et inversement un ensemble peut être représenté par le prédicat dont l'image des éléments de cet ensemble est *vrai* et l'image des autres éléments est $faux$. Ainsi sur $\mathbf{R}$, on utilisera indifféremment le prédicat $x <= 4$ ou l'intervalle $(\Leftrightarrow\infty, 4]$. Nous préférerons cependant travailler sur des prédicats en utilisant des opérateurs logiques, et illustrer les exemples à l'aide d'ensembles, plus faciles à visualiser.

Nous profiterons de plus de la structure de treillis existant sur les prédicats sur V pour induire une telle structure sur les échéances. Cependant, nous inverserons l'ordre : en effet nous considérerons que la plus petite échéance est celle qui ne laisse pas passer le temps, soit $VRAI : v \mapsto vrai$. Au contraire la plus grande sera celle qui laisse toujours passer le temps, soit $FAUX : v \mapsto faux$. Nous verrons une autre raison, technique, de cette inversion de l'ordre en 2.4.

Notation 2 *Nous noterons E le treillis des échéances.*

2.3 Les fonctions de délais possibles

Si les échéances sont faciles d'utilisation, elles sont insuffisantes dès qu'il s'agit de déterminer une durée pendant laquelle le système peut évoluer. Or ce besoin se présente si l'on veut exécuter ou simuler le système, c'est-à-dire si l'on ne considère pas le système comme un objet statique à étudier, mais comme un objet dynamique. Dans notre cas, il est facile de pallier ce problème en considérant une fonction qui, pour une valuation x et une durée t données, dit si l'on rencontre une échéance ou non. Pour un état de contrôle s source d'une unique transition d'échéance c, nous obtenons la fonction suivante :

$$f(v,t) = \forall t' \in [0,t) \ . \ \neg c(v \triangleright_s t')$$

On remarquera que l'on ne tient pas compte de la valeur de c en $x \triangleright t$ puisque, comme indiqué précédemment, c empêche le temps de progresser à partir d'un état et non pas d'atteindre cet état. Par contre tous les états intermédiaires sont examinés pour savoir si le temps n'est pas bloqué avant d'atteindre $v \triangleright_s t'$.

Cette fonction satisfait la propriété d'additivité, définie ci-après, qui signifie que si le temps peut avancer de t_1 puis de t_2, alors il peut avancer de $t_1 + t_2$ directement, et inversement toute progression du temps peut être scindée en deux progressions successives.

Définition 4 *Additivité*
Étant donnée une fonction d'évolution $\triangleright_s$, un prédicat sur $V \times \mathbf{R}_+$ est dit additif s'il satisfait l'équivalence suivante :

$$\forall v \in V, \forall t_1, t_2 \in \mathbf{R}_+ \ . \ p(v, t_1) \wedge p(v \triangleright_s t_1, t_2) \Leftrightarrow p(v, t_1 + t_2)$$

Une conséquence directe de l'additivité est que pour tous réels positifs t et t' tels que t soit supérieur à t' on a l'implication :

$$p(v, t) \Rightarrow p(v, t')$$

Ce qui signifie simplement que si l'on peut attendre plus longtemps qu'une durée t' alors on peut attendre une durée t'.

On a même une propriété encore plus forte.

Propriété 1
Si p est additif, alors pour tout $v \in V$, ou bien $p(v, t)$ est vrai pour tout t, ou bien il existe un réel positif t_0, tel que pour tout $t < t_0$, $p(v, t)$ vaut *vrai* et pour tout $t > t_0$, $p(v, t)$ vaut $faux$.

Preuve 1
Immédiate avec l'implication précédente.
□

On voit aussi que $f(v, 0)$ est vrai pour tout v, c'est-à-dire que l'on autorise toujours d'attendre 0 unité de temps.

Comme nous n'avons pas besoin, a priori, d'autres propriétés sur les prédicats pour qu'ils puissent servir à gérer le temps, nous définirons les fonctions de délais possibles comme des prédicats additifs.

Définition 5 *Fonctions de délais possibles*
Une fonction de délais possibles, ou FDP, pour l'état de contrôle s est un prédicat f sur $V \times \mathbf{R}_+$ additif par rapport à la loi d'évolution $\triangleright_s$, tel que pour tout $v \in V$, $f(v, 0)$ vaut vrai.

Remarquons tout de suite que cette définition ne permet pas de reconnaître aisément si un prédicat est une FDP ou non.

Exemple 2
Voici quelques exemples de prédicats que l'on étudiera dans le cas très simple où l'on n'a qu'une seule variable x et où l'évolution est linéaire : $x \triangleright t = v + t$. Sur la figure 2.3, f_1 et f_2 sont des FDP (cela est laissé au lecteur). Par contre, f_3, qui

est la disjonction de f_1 et f_2, n'en est pas une. En effet, on peut attendre 3 unités de temps à partir de 1, et donc atteindre le point 4, à partir duquel on peut attendre 4 unités de temps, mais on ne peut pas attendre directement 7 unités de temps à partir de 1. La propriété d'additivité est donc violée. Inversement f_4 est une FDP car au point 4, le temps est arrêté.

Comme on vient de le voir, l'additivité n'est pas préservée par union. Mais tout espoir n'est pas perdu.

Proposition 2
L'additivité est préservée par conjonction.

Preuve 2
Soient f_1 et f_2 deux FDP, pour tous réels positifs t_1 et t_2 et toute valuation v, on a l'équivalence :

$$\begin{array}{rcl}(f_1 \wedge f_2)(v, t_1 + t_2) & \Leftrightarrow & f_1(v, t_1 + t_2) \wedge f_2(v, t_1 + t_2) \\ & \Leftrightarrow & f_1(v, t_1) \wedge f_1(v \triangleright_s t_1, t_2) \wedge f_2(v, t_1) \wedge f_2(v \triangleright_s t_1, t_2) \\ & \Leftrightarrow & (f_1 \wedge f_2)(v, t_1) \wedge (f_1 \wedge f_2)(v \triangleright_s t_1, t_2)\end{array}$$

□

Ceci nous incite à chercher une structure de treillis sur les FDP, pour l'ordre $\Rightarrow$. Pour cela il nous faut déterminer la borne supérieure de deux FDP, la borne inférieure étant la disjonction. Si l'on note respectivement $\sqcap$ et $\sqcup$ les bornes inférieure et supérieure dans ce treillis, nous obtenons le résultat suivant.

Proposition 3
$(TP, \Rightarrow, \sqcap, \sqcup)$ est un treillis distributif avec :

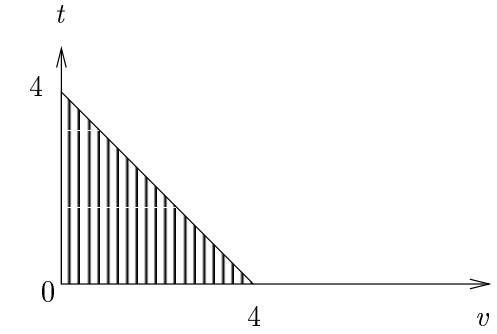
$$f_1 \sqcap f_2 = f_1 \wedge f_2 \quad \text{et} \quad f_1 \sqcup f_2 = \bigvee_{i=1}^{i=\infty} f_1 \vee_i f_2$$

avec :

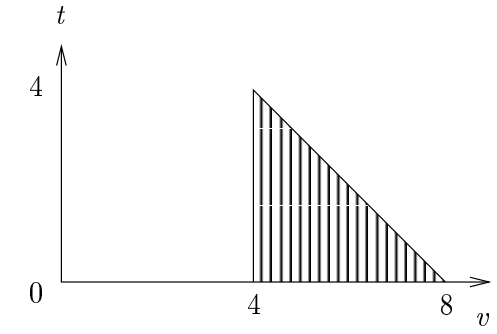
$$\begin{array}{rcl} f_1 \vee_1 f_2 & = & f_1 \vee f_2 \\ (f_1 \vee_{i+1} f_2)(q, t) & = & \exists t'\ 0 \leq t' \leq t\ .\ (f_1 \vee_i f_2)(q, t') \wedge (f_1 \vee f_2)(q \triangleright t', t \Leftrightarrow t') \end{array}$$

Preuve 3
On va d'abord déterminer les bornes supérieure et inférieure et ensuite montrer la distributivité.

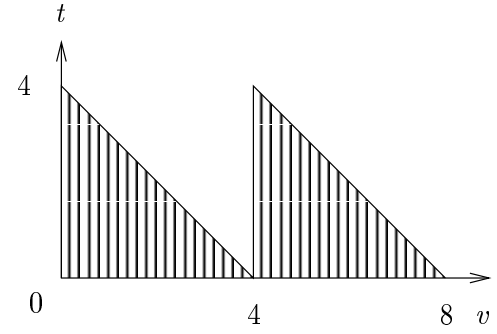
- $\sqcap = \wedge$ *est évident.*
- *Montrons maintenant que $f_1 \sqcup f_2 = \bigvee_{i=1}^{i=\infty} f_1 \vee_i f_2$:*
  f_1 et f_2 impliquent $\bigvee_{i=1}^{\infty} f_1 \vee_i f_2$, en effet : $f_j \Rightarrow f_1 \vee_1 f_2 \Rightarrow \bigvee_{i=1}^{\infty} f_1 \vee_i f_2$ pour j valant 1 ou 2. Montrons que toute fonction d'évolution f impliquée par f_1 et f_2, est impliquée par $f_1 \sqcup f_2 = \bigvee_{i=1}^{i=\infty} f_1 \vee_i f_2$. Pour cela nous allons montrer par induction que $\forall i \in \mathbf{N}\ .\ f_1 \vee_i f_2 \Rightarrow f$:
  $f_1 \vee_1 f_2 \Rightarrow f$ (par choix de f).
  Si $f_1 \vee_{i-1} f_2 \Rightarrow f$, alors pour tous (q, t) tels que $(f_1 \vee_i f_2)(q, t) = vrai$,



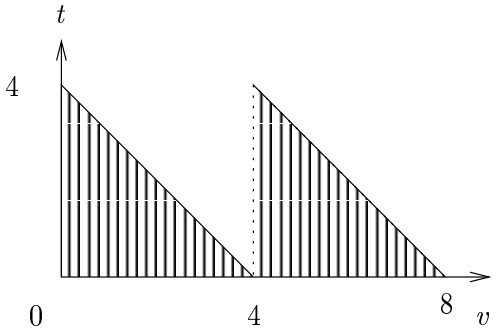
$f_1(v,t) = (0 \leq x \wedge x + t \leq 4) \vee (t = 0)$



$f_2(v,t) = (4 \leq x \wedge x + t \leq 8) \vee (t = 0)$



$f_3 = f_1 \vee f_2$



$f_4(v,t) = (0 \leq x \wedge x + t \leq 4) \vee$
$(4 < x \wedge x + t \leq 8) \vee (t = 0)$

FIG. 2.3 – *Quels prédicats sont des FDP?*

nous avons par définition :
$\exists t'\ 0 \leq t' \leq t\ .\ (f_1 \vee_{i-1} f_2)(q,t') \wedge (f_1 \vee f_2)(q \triangleright t', t \Leftrightarrow t')$
et par conséquent : $\exists t'\ 0 \leq t' \leq t\ .\ f(q,t') \wedge f(q \triangleright t', t \Leftrightarrow t')$
f étant additive, on a donc : $f(q,t) = vrai$.

– *Il reste à montrer la distributivité. Cela nécessite quelques calculs peut-être longs, mais relativement simples.*

Remarquons tout d'abord que $\vee_i$ peut aussi s'exprimer par la formule suivante :

$$(f_1 \vee_i f_2)(v,t) = \left(\begin{array}{l} \exists t_0 = 0,\ \exists t_1 \leq t_0, \ldots, \exists t_k \leq t_{k-1}, \ldots, \exists t_i = t \leq t_{i-1}, \\ \forall j \in \{1, \ldots, i\}\ .\ (f_1 \vee f_2)(v \triangleright t_{j-1}, t_j \Leftrightarrow t_{j-1}) \end{array} \right)$$

qui signifie que $(f_1 \vee_i f_2)(v,t)$ vaut $vrai$ si l'on peut découper l'intervalle $[0,t]$ en i sous-intervalles $[t_j, t_{j+1}]$ sur lesquels soit $f_1(v \triangleright t_j, t_{j+1} \Leftrightarrow t_j)$ soit $f_2(v \triangleright t_j, t_{j+1} \Leftrightarrow t_j)$ vaut $vrai$. Autrement dit, le temps peut passer par $f_1 \vee_i f_2$ s'il peut passer alternativement par f_1 et f_2, en ne changeant que $(i \Leftrightarrow 1)$ fois de fonction pour laisser passer le temps (voir la figure 2.4). Ceci se démontre par récurrence sur i.

  – *Cas de base : pour $i = 1$, comme $\vee_1$ est la disjonction, il est évident que la formule suivante vaut $vrai$:*

$$(f_1 \vee_1 f_2)(v,t) = \{\exists t_0 = 0 \exists t_1 = t \leq t_0 (f_1 \vee f_2)(v \triangleright t_0, t_1 \Leftrightarrow t_0)\}$$

  – *Supposons maintenant que la formule soit égale à $vrai$ pour $i \Leftrightarrow 1$, c'est-à-dire*

$$(f_1 \vee_{i-1} f_2)(v,t) = \left(\begin{array}{l} \exists t_0 = 0,\ \exists t_1 \leq t_0, \ldots, \exists t_k \leq t_{k-1}, \ldots, \exists t_{i-1} = t \leq t_{i-2}, \\ \forall j \in \{1, \ldots, i \Leftrightarrow 1\}\ .\ (f_1 \vee f_2)(v \triangleright t_{j-1}, t_j \Leftrightarrow t_{j-1}) \end{array} \right)$$

  et prouvons la pour i : Par définition de $\vee_i$ on a :

$$(f_1 \vee_i f_2)(v,t) = \exists t' \in [0,t]\ .\ (f_1 \vee_{i-1} f_2)(v,t') \wedge (f_1 \vee f_2)(v \triangleright t', t \Leftrightarrow t')$$

  En appliquant l'hypothèse de récurrence pour développer $(f_1 \vee_{i-1} f_2)(v,t')$ on obtient facilement le résultat :

$$(f_1 \vee_i f_2)(v,t) = \left(\begin{array}{l} \exists t_0 = 0,\ \exists t_1 \leq t_0, \ldots, \exists t_k \leq t_{k-1}, \ldots, \exists t_i = t \leq t_{i-1}, \\ \forall j \in \{1, \ldots, i\}\ .\ (f_1 \vee f_2)(v \triangleright t_{j-1}, t_j \Leftrightarrow t_{j-1}) \end{array} \right)$$

Ensuite il est facile de voir que $f_1 \vee_{i+1} f_2$ implique $f_1 \vee_i f_2$. Ceci implique que pour f_1, f_2, v et t donnés, si $(f_1 \sqcup f_2)(v,t)$ vaut $vrai$ alors il existe $i \in \mathbf{N}$ tel que $(f_1 \vee_i f_2)(v,t)$ vaut $vrai$.

A partir de ces deux remarques, la preuve de la distributivité de $\sqcup$ et $\sqcap$ revient à trouver le bon découpage de l'intervalle $[0,t]$ en sous-intervalles idoines sur lesquels on pourra appliquer la distributivité de $\vee$ et $\wedge$. Cela est laissé au lecteur.

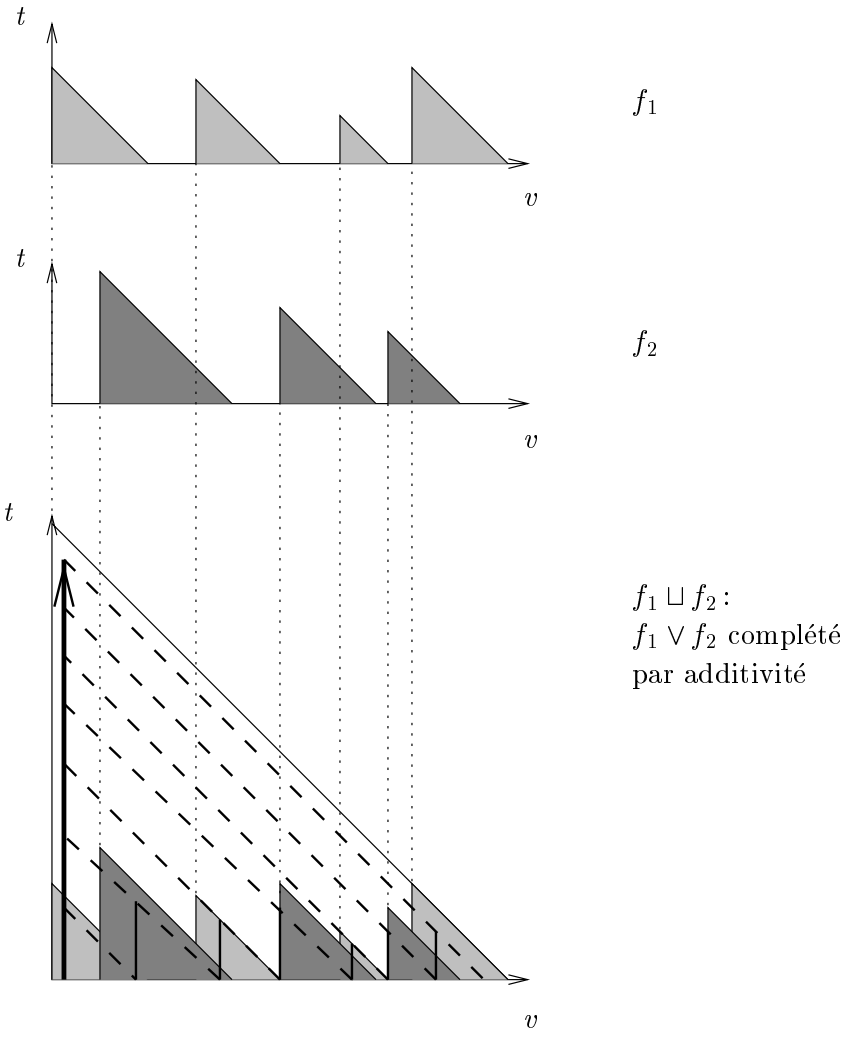


FIG. 2.4 – *Effet de* $\sqcup$

□

Les FDP forment donc un treillis distributif, qui est une structure bien connue. Sur ce treillis le minimum est $\lambda v, t.(t = 0)$ et le maximum est $VRAI$.

Notation 3 *Nous noterons FDP le treillis des FDP.*

2.4 La correspondance de Galois entre E et FDP

Maintenant que nous avons deux treillis distributifs, E (treillis des échéances) et FDP (treillis des fonctions de délais possibles), nous allons chercher à les relier. Ceci nous permettra de passer de l'un à l'autre donc de calculer la FDP correspondant à une échéance et, réciproquement, d'effectuer des opérations sur l'un des treillis en ayant des opérations correspondantes sur l'autre treillis. Ainsi nous pourrons, par exemple, calculer la FDP de l'union de deux échéances à partir des FDP de ces deux échéances.

On peut les définir assez naturellement les fonctions qui permettent de passer de l'une des structures à l'autre. Nous avons déjà la fonction fdp définie par

$$\forall (v,t) \in V \times \mathbf{R}_+ \;.\; fdp(c)(v,t) = \forall t' \in [0,t) \;.\; \neg c(v \triangleright t')$$

qui calcule la FDP associée à une échéance. Il nous reste à trouver une échéance à partir d'une FDP. Si l'on se rappelle que les échéances caractérisent les états à partir desquels le temps ne peut progresser, il suffit de repérer les états à partir desquels la FDP ne laisse progresser le temps par aucun délai positif, à part éventuellement 0 qui ne correspond pas à une attente. Ceci nous donne donc la fonction ec suivante :

$$\forall v \in V \;.\; ec(f)(v) = \forall t > 0 \;.\; \neg f(v,t)$$

La première relation à vérifier est : est-ce que cela nous donne une bijection ? La réponse est évidemment non, comme on peut le voir sur le contre-exemple suivant.

Exemple 3
Pour une seule variable x, et en prenant comme fonction d'évolution $v \triangleright t = x + t$, posons $c(v) = x > 3$. Ce prédicat ne peut être l'image d'une FDP f par ec. Supposons, en effet, que ce soit le cas : il existerait f telle que $ec(f)(v) = x > 3$. Alors nécessairement pour $x > 3$, f ne pourrait laisser progresser le temps. Par contre, pour $x = 3$ (qui n'appartient pas à l'échéance), le temps doit pouvoir progresser d'une durée strictement positive. Notons t_0 cette durée, on aurait alors $f(x = 3, t_0) = vrai$ et (par additivité) $f((x = 3) \triangleright t_0/2, t_0/2) = vrai$. Ce qui contredit le fait que f ne peut laisser le temps progresser pour $x > 3$.

Une autre relation classique est la correspondance de Galois. C'est une version très affaiblie de la bijection où deux ensembles de cardinaux différents peuvent être en correspondance. Par contre la correspondance assure un certain nombre de propriétés qui en font une relation très utile. Elle permet en particulier de dire qu'un ensemble peut être approché par un autre. On trouvera

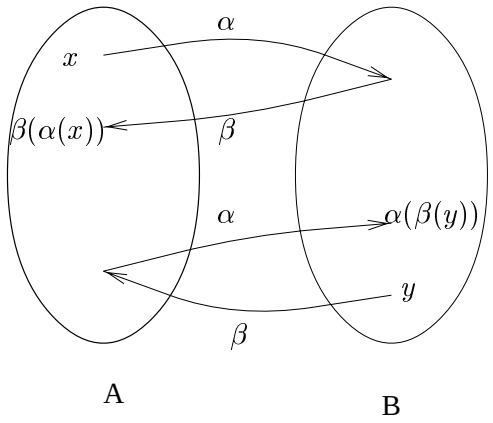


FIG. 2.5 – *Correspondance de Galois*

dans [CC77, CC92] et [LGS+95] un exposé des propriétés de cette correspondance. On retiendra cependant qu'il y a correspondance entre deux ensembles 2^A et 2^B s'il existe deux fonctions croissantes $\alpha : 2^A \to 2^B$ et $\beta : 2^B \to 2^A$ telles que :

$$\forall x \subset A \;.\; x \subset \beta(\alpha(x))$$
$$\forall y \subset B \;.\; \alpha(\beta(y)) \subset y$$

Cette relation est illustrée sur la figure 2.5

Regardons donc si la paire (fdp, ec) forme une correspondance de Galois. Pour commencer, il nous faut des fonctions croissantes, ce qui est le cas car l'ordre sur E est l'ordre inverse de l'ordre habituel (cf 2.2). Ensuite il faut montrer que $ec \circ fdp$ et $fdp \circ ec$ sont impliquées par l'identité. Ceci est donné par les deux propositions suivantes.

Proposition 4
Pour toute échéance c, on a l'implication : $c \Rightarrow ec(fdp(c))$.

Preuve 4
En appliquant directement les définitions de fdp et ec :

$$\begin{aligned} ec(fdp(c))(v) &= \forall t > 0 \;.\; \neg fdp(c)(v,t) \\ &= \forall t > 0 \;.\; \neg \forall t' \in [0,t) \;.\; \neg c(v \triangleright t') \\ &= \forall t > 0 \;.\; \exists t' \in [0,t) \;.\; c(v \triangleright t') \end{aligned}$$

On remarque que si $c(v) = vrai$ alors en prenant $t' = 0$, on a $ec(fdp(c))(v) = vrai$. D'où l'implication recherchée.
□

Proposition 5
Pour toute FDP f, on a l'implication : $f \Rightarrow fdp(ec(f))$.

Preuve 5
Toujours en appliquant les définitions :

$$\begin{aligned} fdp(ec(f))(v,t) &= \forall t' \in [0,t) \;.\; \neg ec(f)(v \triangleright t') \\ &= \forall t' \in [0,t) \;.\; \neg \forall t'' > 0 \;.\; \neg f(v \triangleright t', t'') \\ &= \forall t' \in [0,t) \;.\; \exists t'' > 0 \;.\; f(v \triangleright t', t'') \end{aligned}$$

f étant additive si $f(v,t)$ vaut vrai alors il en est de même pour $f(v \triangleright t', t \Leftrightarrow t')$ pour tout t' dans $[0,t)$. Pour tout t' il suffit alors de prendre $t'' = t \Leftrightarrow t'$, qui est strictement positif. Ceci permet de conclure.
□

On a donc bien une correspondance de Galois entre E et FDP. Si nous avions choisi l'ordre habituel sur les échéances, nous n'aurions pas ce résultat, c'est pour cela que nous avons dû inverser l'ordre. De nombreuses propriétés découlent de cette correspondance. Ce sont des résultats classiques, donc nous n'en donnerons pas de preuve. Dans le cas qui nous intéresse, les preuves sont assez triviales, et des résultats plus généraux peuvent être vus dans [CC77, CC92].

Proposition 6
$fdp \circ ec \circ fdp = fdp$ et $ec \circ fdp \circ ec = ec$
$fdp(c_1 \vee c_2) = fdp(c_1) \sqcap fdp(c_2) = fdp(c_1) \wedge fdp(c_2)$
$ec(f_1 \sqcup f_2) = ec(f_1) \wedge ec(f_2)$

Ceci est un début, mais c'est insuffisant pour les calculs que nous voulons faire. Il serait en effet intéressant d'avoir un homomorphisme pour pouvoir effectuer des opérations sur E ou FDP en sachant quel effet cela aura sur les FDP ou échéances correspondantes.

Néanmoins, il est facile de vérifier que les images de E par fdp et de FDP par ec sont en bijection par ces fonctions, du fait de la correspondance de Galois. Cherchons donc à caractériser ces ensembles.

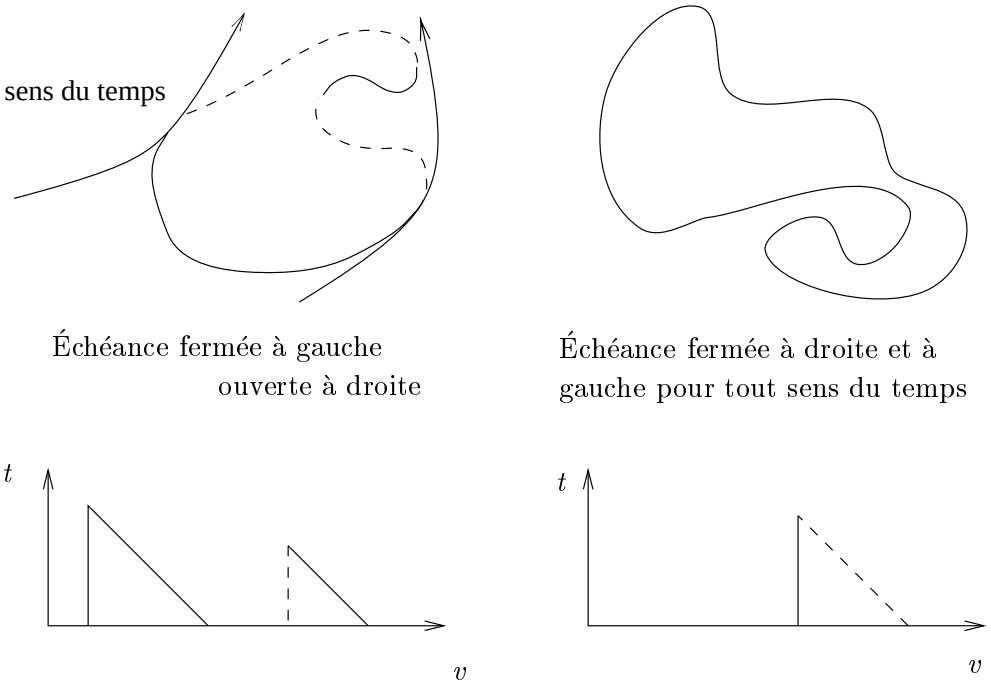
Nous avons vu à l'exemple 3 que $x > 3$ n'est pas dans l'image $Im(ec)$ de FDP par ec. Par contre $x \geq 3$ est l'image de $x + t < 3$. La différence est due à une question de fermeture. Nous allons voir aussi que seule compte la "direction du temps", au sens où les seules propriétés importantes portent sur l'évolution temporelle $t \mapsto v \triangleright t$ pour une valuation v donnée. C'est pourquoi nous allons définir les notions suivantes.

Définition 6 *Fermeture (prédicats)*
Étant donné un prédicat temporel g
($g : \mathbf{R}_+ \to \{vrai, faux\}$), nous dirons que g est fermé à gauche *si*

$$\forall t_0 \;.\; \neg g(t_0) \Rightarrow \exists \epsilon > 0 \;.\; \forall \epsilon' \leq \epsilon \;.\; \neg g(t_0 + \epsilon')$$

Nous dirons que g est fermé à droite *s'il vérifie la formule précédente où l'on a remplacé $g(t_0 + \epsilon')$ par $g(t_0 \Leftrightarrow \epsilon')$.*

Remarque 4
Cette définition est simplement la définition duale de celle des ensembles



Échéance fermée à gauche ouverte à droite

Échéance fermée à droite et à gauche pour tout sens du temps

Fdp fermée à droite.

Fdp ouverte à droite.

FIG. 2.6 – *Échéances et FDP fermées et ouvertes*

fermés à droite ou à gauche sur $\mathbf{R}_+$: c est fermé à gauche s'il est le dual d'un sous-ensemble fermé à gauche de $\mathbf{R}_+$.

Il est alors facile de parler de prédicats ouverts à gauche ou à droite, suivant le même principe.

Définition 7 *Fermeture (échéances, FDP)*
Nous dirons qu'une échéance c est fermée à gauche (resp. à droite) dans le sens du temps *si pour toute valuation v_0 le prédicat temporel $t \mapsto c(v_0 \triangleright t)$ est fermé à gauche (resp. à droite).*

Nous dirons qu'une FDP f est fermée à droite dans le sens du temps *si pour toute valuation v_0 le prédicat temporel $t \mapsto f(v_0, t)$ est fermé à droite.*

On définira de même l'ouverture à gauche et à droite dans le sens du temps.

Remarque 5 De par leur définition, les FDP sont nécessairement fermées à gauche (voir propriété 1), c'est pour cela que nous n'avons pas pris soin de définir la fermeture à gauche sur les FDP.

Quelques exemples d'échéances ouvertes ou fermées sont donnés dans la figure 2.6.

Ceci nous permet maintenant de caractériser $Im(ec)$ et $Im(fdp)$.

Proposition 7

- L'image de FDP par ec est l'ensemble des échéances fermées à gauche dans le sens du temps.
- L'image de E par fdp est l'ensemble des FDP fermées à droite dans le sens du temps.

Preuve 7

- *Montrons d'abord que $Im(ec)$ ne contient que des échéances fermées à gauche. Puis que toute échéance fermée à gauche est dans $Im(ec)$, on aura alors l'égalité cherchée.*

  Pour toute FDP f, on a $ec(f)(v) = \forall t > 0 \ . \ \neg f(v,t)$ et donc $\neg ec(f)(v) = \exists t > 0 \ . \ f(v,t)$. Supposons que $ec(f)(v \triangleright t_0) = faux$ (voir la définition de la fermeture), et posons $v_0 = v \triangleright t_0$. On a alors d'après ce qui précède : $\exists t_1 > 0 \ . \ f(v_0, t_1)$. Pour tout ϵ strictement inférieur à t_1 on a donc, par additivité : $\exists t' = t_1 \Leftrightarrow \epsilon \ . \ f((v_0 \triangleright \epsilon), t')$, c'est-à-dire $ec(f)(v_0 \triangleright \epsilon) = faux$. Ce qui prouve la fermeture de $ec(f)$.

  Soit maintenant une échéance c fermée à gauche. Montrons qu'elle est invariante par $ec \circ fdp$, donc qu'elle est l'image de $fdp(c)$ et qu'elle est dans $Im(ec)$. On a vu $c \Rightarrow ec(fdp(c))$, montrons l'implication inverse. Supposons que $c(v) = faux$, comme c est fermée à gauche :

  $$\exists t > 0, \forall t' \in [0,t) \ . \ \neg c(v \triangleright t')$$

  Or on a calculé que :

  $$ec(fdp(c))(v) = \forall t > 0, \exists t' \in [0,t) \ . \ c(v \triangleright t')$$

  On en déduit que $ec(fdp(c))(v) = faux$, et donc $ec(fdp(c)) \Rightarrow c$. Ce qui nous donne l'égalité $ec(fdp(c)) = c$.

  Ceci suffit pour affirmer que $Im(ec)$ est l'ensemble des échéances fermées à gauche.

- *Montrons de même que $Im(fdp)$ ne contient que des FDP fermées à droite, puis que toutes les FDP fermées à droite sont dans $Im(ec)$.*

  Pour toute échéance c, on a $fdp(c)(v,t) = \forall t' \in [0,t) \ . \ c(v \triangleright t')$ et donc $\neg fdp(c)(v,t) = \exists t' \in [0,t) \ . \ \neg c(v \triangleright t')$. Supposons que $fdp(c)(v,t_0) = faux$. On a d'après ce qui précède : $\exists t' \in [0,t) \ . \ \neg c(v \triangleright t')$. Ce que l'on peut récrire en $\exists t' \in (0,t] \ . \ \neg c(v \triangleright t \Leftrightarrow t')$. Posons alors $t_0 = t \Leftrightarrow t'$. Pour tout t'' strictement inférieur à t' on a $0 \leq t_0 < t \Leftrightarrow t''$ et $c(v \triangleright t_0) = vrai$ ce qui est la même chose que $fdp(c)(v, t \Leftrightarrow t'') = faux$. Ceci prouve la fermeture de $fdp(c)$.

  Soit maintenant f fermée à droite. Montrons qu'elle est invariante par $fdp \circ ec$. On a vu $f \Rightarrow ec(fdp(c))$, montrons l' implication inverse. Supposons que $fdp(ec(f))(v,t)$ vaut $vrai$. On sait que

  $$fdp(ec(f))(v,t) = \forall t' \in [0,t), \ \exists t'' > 0 \ . \ f(v \triangleright t', t'')$$

On en déduit que $f(v,t'')$ est vrai pour au moins une valeur strictement positive de t''. Et par fermeture à droite de f, il existe t_0 maximum (éventuellement infini) tel que $f(v,t_0)$ soit vrai. t_0 étant maximal, il en découle que $\forall t'' > 0 \ . \ \neg f(v, t_0 + t'')$. Par additivité, on a donc: $\forall t'' > 0 \ . \ \neg f(v \triangleright t_0, t'')$. En comparant cela à la valeur de $fdp(ec(f))(v,t)$, on en déduit que $t_0 \geq t$. On a donc nécessairement $f(v,t) = vrai$. Donc, $fdp(ec(f)) \Rightarrow f$, et finalement $fdp(ec(f)) = f$.

$Im(fdp)$ est donc l'ensemble des FDP fermées à droite.

□

La question que l'on peut se poser maintenant est la suivante: serait-il raisonnable de se restreindre à $Im(ec)$ et $Im(fdp)$?

Qu'apportent les échéances qui ne sont pas fermées à gauche? Vu l'approche que nous avons prise, la réponse est simple: elles ne nous apportent que des problèmes. En effet aux points de la frontière gauche d'une échéance, qu'ils soient dans l'échéance ou non, le temps ne peut progresser, car toute attente mène dans l'échéance. Il est donc utile de se restreindre à des échéances fermées à gauche.

Quant aux FDP, on ne les manipulera pas directement. On les calculera à partir d'échéances, donc ce n'est pas un problème de se restreindre à $Im(fdp)$.

2.5 Traduction des échéances en FDP

2.5.1 Échéances bien définies

Les résultats précédents montrent une puissante corrélation entre les échéances et les FDP. Dans la pratique cependant nous utilisons des systèmes complexes qui sont obtenus en combinant des systèmes plus simples. Dès lors, les échéances d'un système hybride, ou de façon équivalente toute forme de conditions de progression du temps, sont obtenues comme combinaison des échéances de ses composants. Nous étudierons dans cette partie les résultats permettant le calcul compositionnel des FDP, c'est-à-dire un calcul de FDP comme combinaisons de FDP plus simples, et nous verrons que les échéances seront obtenues par combinaisons d'échéances plus simples. Nous chercherons d'abord les conditions sous lesquelles nous avons un morphisme de treillis entre E et FDP. Nous verrons ensuite les résultats sur la traduction des échéances de forme modale en FDP. Les échéances de forme modale sont des échéances particulières, bien adaptées aux systèmes hybrides car elles sont définies à partir de gardes, ce qui permet d'avoir facilement les propriétés liant échéances et gardes.

Si nous voulons avoir un morphisme de treillis, il est nécessaire d'avoir une bijection. Nous nous restreindrons donc aux échéances fermées à gauche et aux FDP fermées à droite. De plus, nous avons besoin de la propriété:

$$fdp(c_1 \wedge c_2) = fdp(c_1) \sqcup fdp(c_2)$$

Observons que dans le cas général, cette égalité n'est pas satisfaite.

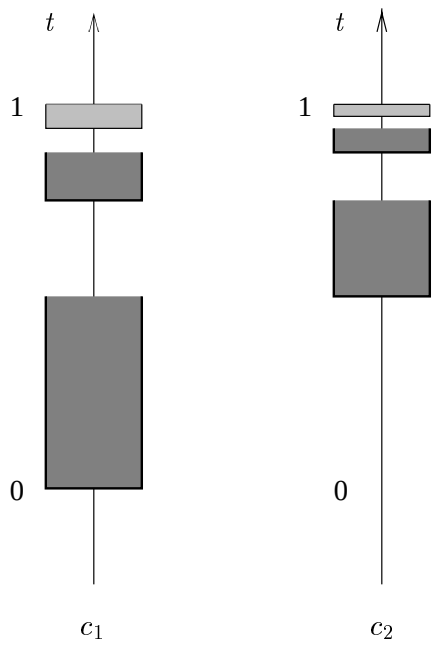


FIG. 2.7 – *Les ensembles de l'exemple 4.*

Exemple 4
Regardons ce qui se passe pour les conditions c_1 et c_2 définies par :

$$\begin{array}{r} c_1 = \bigvee_i p_{2i} \\ c_2 = \bigvee_i p_{2i+1} \end{array} \quad \text{avec} \quad p_i(v) = 1 \Leftrightarrow 2^{-i} \leq x < 1 \Leftrightarrow 2^{-(i+1)}$$

dans le cas où nous n'avons qu'une seule variable, x, et que la fonction d'évolution est $v \triangleright t = x + t$. Ces ensembles sont représentés figure 2.7

Il est facile de voir que $c_1 \wedge c_2 = FALSE$ et par conséquent $fdp(c_1 \wedge c_2) = TRUE$. Mais, par définition de $\sqcup$, on a pour tout réel t supérieur à 1 : $(fdp(c_1) \sqcup fdp(c_2))(0,1) = false$. On a vu l'implication $fdp(c_1) \sqcup fdp(c_2) \Rightarrow fdp(c_1 \wedge c_2)$, on sait donc maintenant que dans le cas général cette implication est stricte. Ici cela provient du fait que chaque FDP permet de faire progresser le temps jusqu'à un point où il peut progresser par l'autre FDP, mais si l'on veut faire progresser le temps de 1 ou plus il faut changer infiniment souvent de FDP or, par définition, $\sqcup$ ne permet qu'un nombre fini de tels changements.

Ceci nous amène à poser la définition suivante.

Définition 8 *Échéances bien définies*
Nous dirons qu'une échéance c est bien définie *si elle est fermée à gauche et si elle est telle que $\lambda t.c(v \triangleright t)$ ne change qu'un nombre fini de fois de valeur sur un intervalle de temps fini, et ce pour toute valuation v.*

Remarque 6 L'ensemble des échéances bien définies est stable par disjonction et conjonction. Il forme de plus un sous-treillis de E de même minimum et même maximum.

Il ne nous reste plus qu'à voir que ce sont là les objets que nous cherchons.

Proposition 8
La restriction de fdp aux échéances bien définies est un morphisme de treillis.

Preuve 8
Par définition de fdp, nous avons immédiatement $fdp(c_1 \vee c_2) = fdp(c_1) \sqcap fdp(c_2)$ et $fdp(c_1) \sqcup fdp(c_2) \Rightarrow fdp(c_1 \wedge c_2)$. Pour avoir un morphisme de treillis, il nous reste donc à montrer l'implication $fdp(c_1 \wedge c_2) \Rightarrow fdp(c_1) \sqcup fdp(c_2)$.

Supposons donc que $fdp(c_1 \wedge c_2)(v,t) = vrai$, et regardons si l'on peut découper $[0,t]$ en sous-intervalles $[0,t_1]$, $[t_1,t_2]$,..., $[t_{n-1},t_n]$ avec $(fdp(c_1) \vee fdp(c_2))(v \triangleright t_i, t_{i+1} \Leftrightarrow t_i) = vrai$ pour chaque $i < n$. On a :

$$\begin{aligned} fdp(c_1 \wedge c_2) &= \forall t' \in [0,t) \ . \ \neg(c_1 \wedge c_2)(v \triangleright t') \\ &= \forall t' \in [0,t) \ . \ (\neg c_1(v \triangleright t')) \vee (\neg c_2(v \triangleright t')) \end{aligned}$$

Or $\lambda\tau.c_1(v \triangleright \tau)$ et $\lambda\tau.c_2(v \triangleright \tau)$ ne changent qu'un nombre fini de fois de valeur sur $[0,t]$. On peut donc diviser cet intervalle en un nombre fini, n, de sous-intervalles $[t_0,t_1]$, $[t_1,t_2]$,..., $[t_{n-1},t_n]$ sur l'intérieur desquels c_1 et c_2 ont une valeur constante. Autrement dit :

$$\begin{aligned} &t_0 = 0 \ , \ \ t_n = t \ \ \text{et} \ \ \forall i < n \ . \ t_{i+1} \leq t_i \\ &\forall i < n \ . \ (\forall t' \in (t_i, t_{i+1}) \ . \ \neg c_1(v \triangleright t')) \vee (\forall t' \in (t_i, t_{i+1}) \ . \ \neg c_2(v \triangleright t')) \end{aligned}$$

Cela ne nous satisfait pas tout à fait, car il faut encore que si l'on ait $\neg c_1$ sur (t_i, t_{i+1}) on ait aussi $\neg c_1$ en t_i, sinon le temps se trouverait bloqué en t_i, et l'on n'aurait pas $fdp(c_1)(v \triangleright t_i, t_{i+1} \Leftrightarrow t_i)$.

On va utiliser ici le fait que les échéances considérées, c_1 et c_2, sont fermées à gauche. Si l'on a $\neg c_1$ sur (t_i, t_{i+1}), avec $t_{i+1} > t_i$, nécessairement on a $\neg c_1$ en t_i. Le cas $t_{i+1} = t_i$ n'est pas gênant. Ceci nous donne donc pour chaque $i < n$:

$$\forall i < n \ . \ (\forall t' \in [t_i, t_{i+1}) \ . \ \neg c_1(v \triangleright t')) \vee (\forall t' \in (t_i, t_{i+1}) \ . \ \neg c_2(v \triangleright t'))$$

Ce qui se traduit en :

$$\forall i < n \ . \ fdp(c_1)(v \triangleright t_i, t_{i+1} \Leftrightarrow t_i) \vee fdp(c_2)(v \triangleright t_i, t_{i+1} \Leftrightarrow t_i)$$

Et finalement nous obtenons : $(fdp(c_1) \sqcup fdp(c_2))(v,t) = vrai$.
□

Nous avons donc défini une classe d'échéances avec lesquelles il est particulièrement pratique de travailler. Nous remarquerons de plus que les restrictions apportées pour les échéances ne sont pas particulièrement contraignantes, puisqu'en pratique les échéances correspondent à des ensembles convexes ou, plus rarement, à des unions finies d'ensembles convexes et que la fermeture à gauche est de toute manière imposée. Les échéances bien définies sont donc bien adaptées à l'utilisation que l'on veut en faire.

2.5.2 Traduction des échéances de forme modale

Nous allons voir ici les résultats sur le calcul compositionnel des FDP lorsque les échéances sont données par des formules modales. Mais commençons par voir ce que sont les formules modales. Intuitivement, ce sont des prédicats sur les états dans lesquels on fait intervenir des événements passés et futurs. Plus formellement, nous utiliserons la définition suivante.

Définition 9 *Échéances de forme modale*
Les échéances de forme modale sont données par le langage dont la syntaxe est la suivante :

$$c ::= g \mid g\downarrow \mid FAUX$$
$$g ::= VRAI \mid FAUX \mid p \mid g \wedge g \mid g \vee g \mid \Box g \mid \Diamond g \mid \boxdot g \mid \diamondsuit g$$

avec $p \in P$. P est un ensemble de prédicats atomiques sur les états.

Ces prédicats, vus en tant qu'échéances, sont bien définis et fermés à droite. Nous définirons la sémantique des échéances de forme modales par la fonction $\mid . \mid$, qui à une formule c associe un prédicat sur V, $\mid c \mid$. Cette fonction est définie par induction sur la structure de c.

$$\begin{array}{rcl}
\mid FAUX \mid & = & FAUX \\
\mid VRAI \mid & = & VRAI \\
\mid g_1 \wedge g_2 \mid & = & \mid g_1 \mid \wedge \mid g_2 \mid \\
\mid g_1 \vee g_2 \mid & = & \mid g_1 \mid \vee \mid g_2 \mid \\
\mid \Diamond g \mid (v) & = & \exists t \geq 0 \ . \ \mid g \mid (v \triangleright t) \\
\mid \Box g \mid (v) & = & \forall t \geq 0 \ . \ \mid g \mid (v \triangleright t) \\
\mid \diamondsuit g \mid (v) & = & \exists t \geq 0 \ . \ \exists v' \ . \ v = v' \triangleright t \ \wedge \mid g(v') \mid \\
\mid \boxdot g \mid (v) & = & \forall t \geq 0 \ . \ \forall v' \ . \ v = v' \triangleright t \ \Rightarrow \mid g(v') \mid \\
\mid g\downarrow \mid (v) & = & \mid g \mid (v) \wedge \exists t > 0 \ . \ \forall t' \in (0, t] \ . \ \mid \neg g \mid (v \triangleright t')
\end{array}$$

Remarque 7
Observons que $\Box$, $\Diamond$, $\boxdot$, $\diamondsuit$ ne correspondent pas aux modalités usuelles de la logique temporelle [Pnu77], signifiant respectivement dorénavant, inévitablement, jusqu'à présent et précédemment. En effet, la logique précédemment citée tient compte des actions. Ici, nous ne tenons compte que de l'évolution virtuelle du système où seul interviendrait le passage du temps, non limité par les conditions de progression du temps. Nous verrons au chapitre suivant que cela permet de définir les propriétés temporelles locales à un état de contrôle donné, dans le cas d'une évolution idéale se déroulant entièrement dans cet état de contrôle, et indépendamment de l'évolution réelle faisant intervenir les actions.

L'opérateur $\downarrow$ est l'opérateur de front descendant, qui associe à une formule c la formule dont la sémantique est la frontière droite de $\mid c \mid$.

Nous définirons en 3.1.1 les échéances à partir de gardes. C'est pour cela que nous définissons la grammaire en deux étapes, g correspondant ici aux gardes. Cependant la grammaire donnée offre un choix plus général, et si effectivement chaque échéance est définie à partir d'une garde, ce ne sera pas nécessairement celle-ci qui sera la garde liée à la transition. La seule contrainte que nous exigeons est que les échéances soient incluses dans les gardes.

C'est pour cette raison, et à cause de l'opérateur $\downarrow$, que nous imposons que toutes les formules précédentes soient fermées à droite. Comme les opérateurs que nous utilisons préservent la fermeture, il suffit d'exiger que les prédicats atomiques de P soient fermés à droite. C'est dans la même optique que nous n'utilisons pas la négation, qui ne préserve pas la fermeture.

Cependant dans les calculs qui suivent nous serons amenés à faire apparaître des formules négatives. Nous utiliserons alors la signification usuelle de $\neg$, ce qui implique les relations suivantes :

$$\begin{array}{ll} \neg VRAI = FAUX & \neg FAUX = VRAI \\ \neg(g_1 \wedge g_2) = \neg g_1 \vee \neg g_2 & \neg(g_1 \vee g_2) = \neg g_1 \wedge \neg g_2 \\ \neg\Box g = \Diamond \neg g & \neg\Diamond g = \Box \neg g \\ \neg \boxdot\, g = \diamondsuit\, \neg g & \neg \diamondsuit\, g = \boxdot\, \neg g \end{array}$$

De plus, nous serons amenés à fermer les formules ouvertes ainsi créées. Nous utiliserons pour cela l'opérateur de fermeture $\mathcal{F}$, qui est formellement défini par la formule suivante :

Définition 10 *Opérateur de fermeture $\mathcal{F}$*

$$\mathcal{F}(g)(v) = g(v) \vee (\quad \neg g(v) \wedge \exists t > 0 \;.\; (\begin{array}{l} (\forall t' \in (0,t] \;.\; g(v \triangleright t')) \vee \\ (\forall t' \in (0,t],\ \exists v' \in V \;.\; v = v' \triangleright t' \wedge g(v')) \end{array}))$$

Voyons maintenant comment l'on s'y prend pour le calcul des FDP.

Théorème 9
Pour toute échéance c, $fdp(c)$ peut être exprimée comme une formule du langage suivant :

$$\begin{array}{rl} f ::= & \lambda v, t.\neg g(v) \vee (t = 0) \mid \lambda v, t.g(v) \vee (t = 0) \mid \lambda v, t.\mathcal{F}(\neg g)(v \triangleright t) \vee (t = 0) \mid \\ & \lambda v, t.g(v \triangleright t) \vee t = 0 \mid fdp(p) \mid fdp(\neg p) \mid f \wedge f \mid f \sqcup f \end{array}$$

où g est encore dans le langage précédent, et p appartient à P.

Preuve 9
Par induction sur la structure de c, dans le langage précédent.

- *$fdp(g)$: le calcul se fait par induction sur la structure de g :*
  - *$fdp(VRAI)$, $fdp(FAUX)$, $fdp(p)$, $fdp(g_1 \wedge g_2)$ et $fdp(g_1 \vee g_2)$ se réduisent facilement. Il reste le cas des quatre opérateurs modaux.*
  - *Si $\Diamond g$ ou $\boxdot g$ sont faux à un état, ils resterons toujours faux. D'après la définition de fdp, il est alors suffisant de tester leur valeur à l'état courant pour savoir si le temps peut passer : $fdp(\Diamond g)(v,t) = \neg\Diamond g(v) \vee (t = 0)$ et $fdp(\boxdot\, g)(v,t) = \neg \boxdot\, g(v) \vee (t = 0)$.*
  - *Si $\Box g$ ou $\diamondsuit g$ sont vrais à un état donné, il le resteront toujours. Pour pouvoir avancer de v à $v \triangleright t$ sans rencontrer $\Box g$, par exemple, il suffit de savoir si $\Box g(v \triangleright t)$ vaut $faux$, ou si $v \triangleright t$ est le premier point où $\Box g$ vaut $vrai$, ce qui revient à savoir si $\mathcal{F}(\neg\Box g)(v)$ vaut $vrai$. On obtient ainsi les formules suivantes : $fdp(\Box g)(v) = \mathcal{F}(\neg\Box g)(v \triangleright t) \vee (t = 0)$ et $fdp(\diamondsuit\, g)(v) = \mathcal{F}(\neg \diamondsuit\, g)(v \triangleright t) \vee (t \Leftrightarrow 0)$.*

– $fdp(g\downarrow)$: *nous allons prouver que pour* g *bien définie et fermée c'est égal à* $fdp(g) \sqcup fdp(\neg g)$.

Commençons par étudier $fdp(g) \sqcup fdp(\neg g)$. *On sait que cette FDP permet au temps de progresser si alternativement* $fdp(g)$ *et* $fdp(\neg g)$ *lui permettent de progresser. Mais voyons ce qu'il se passe si l'on veut enchaîner un passage du temps par* $fdp(\neg g)$ *puis un par* $fdp(g)$, *en d'autres termes que vaut la formule :*

$$\exists t' \in [0,t)\ .\ fdp(\neg g)(v,t') \wedge fdp(g)(v \triangleright t', t \Leftrightarrow t')$$

On peut la récrire en :

$$\exists t' \in [0,t)\ .\ (\forall t'' \in [0,t') g(v \triangleright t'')) \wedge (\forall t'' \in [t',t) \neg g(v \triangleright t))$$

g *étant fermée, cette formule ne peut valoir* $vrai$ *que si* $t = 0$. *On en déduit que l'on ne peut laisser progresser le temps de la manière envisagée ci-dessus. Dès lors, on peut simplifier l'expression de* $fdp(g) \sqcup fdp(\neg g)$:

$$(fdp(g) \sqcup fdp(\neg g))(v,t) = \exists t' \in [0,t]\ .\ fdp(g)(v,t') \wedge fdp(\neg g)(v \triangleright t', t \Leftrightarrow t')$$

Revenons maintenant à $fdp(g\downarrow)$:

$$\begin{array}{rll} fdp(g\downarrow)(v,t) & = \forall t' \in [0,t)\ . & \neg(g\downarrow)(v \triangleright t') \\ & = \forall t' \in [0,t)\ . & \neg(g(v \triangleright t') \wedge \exists t_1 > 0, \\ & & \qquad \forall {t_1}' \in (0,t_1]\ .\ \neg g(v \triangleright t' + {t_1}')) \\ & = \forall t' \in [0,t)\ . & \neg g(v \triangleright t') \vee \\ & & \forall t_1 > 0,\ \exists {t_1}' \in (0,t_1]\ .\ g(v \triangleright t' + {t_1}') \end{array}$$

Cette formule est de la forme $\forall t' \in [0,t)\ .\ A \vee B$. *On va montrer que si* B *est vrai en* t' *alors il le reste sur* $[t',t)$. *Ce qui permettra de conclure que la formule est équivalente à* $\exists t'\ .\ (\forall t'' \in [0,t')\ .\ A \wedge \forall t'' \in [t',t)\ .\ B)$.

g *étant fermée à gauche, on a l'implication :*

$$\forall t_1 > 0,\ \exists {t_1}' \in (0,t_1]\ .\ g(v \triangleright t' + {t_1}') \Rightarrow g(v \triangleright t')$$

Supposons que $fdp(g\downarrow)(v,t) = vrai$, *et que* $\forall t_1 > 0,\ \exists {t_1}' \in (0,t_1]\ .\ g(v \triangleright t' + {t_1}')$ *est vrai en* $t' = t_0$. *Alors, par fermeture à droite de* g, *ou bien* g *reste vraie sur* $[t_0, +\infty)$, *ou bien il existe* $t_1 > t_0$ *maximum tel que* g *reste vraie sur* $[t_0, t_1]$, *et devienne fausse immédiatement après. Comme* t_1 *ne satisfait aucun des deux termes de la disjonction* $\neg g(v \triangleright t') \vee \forall t_1 > 0,\ \exists {t_1}' \in (0,t_1]\ .\ g(v \triangleright t' + {t_1}')$, *on en déduit que* $t_1 > t$. *Dans tous les cas* g *reste vraie sur* $[t_0, t)$. *On a alors l'implication :*

$$fdp(g\downarrow)(v,t) \Rightarrow \exists t' \in [0,t]\ .\ fdp(g)(v,t') \wedge fdp(\neg g)(v \triangleright t', t \Leftrightarrow t')$$

L'implication inverse est immédiate. On a donc l'égalité. Et finalement :

$$fdp(g\downarrow) = fdp(g) \sqcup fdp(\neg g)$$

Ce résultat est illustré dans l'exemple 5.

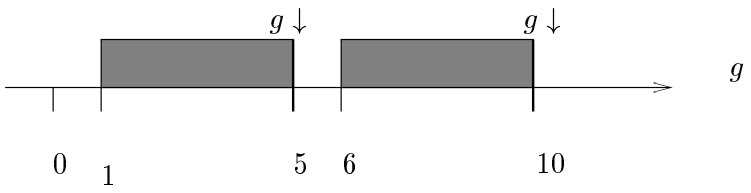


FIG. 2.8 – *FDP et front descendant*

– *Il nous faut donc maintenant calculer* $fdp(\neg g)$*, pour* g *fermée. On obtient facilement que :*

$$
\begin{array}{l}
fdp(\neg VRAI) = VRAI \\
fdp(\neg FAUX)(v,t) = (t=0) \\
fdp(\neg(g_1 \wedge g_2)) = fdp(\neg g_1) \wedge fdp(\neg g_2) \\
fdp(\neg\Box g)(v,t) = \Box g(v) \vee (t=0) \\
fdp(\neg \diamondsuit\, g)(v,t) = \diamondsuit\, g(v) \vee (t=0) \\
fdp(\neg\Diamond g)(v,t) = \Diamond g(v \triangleright t) \vee (t=0) \\
fdp(\neg \boxdot\, g)(v,t) = \boxdot\, g(v \triangleright t) \vee (t=0)
\end{array}
$$

Il reste le cas $fdp(\neg(g_1 \vee g_2))$*, car* $\neg g_1$ *et* $\neg g_2$ *n'étant pas bien définies, on ne peut appliquer le morphisme de treillis. Cependant* $\neg g_1$ *et* $\neg g_2$ *étant toutes les deux ouvertes, on a quand même l'égalité :* $fdp(\neg(g_1 \vee g_2)) = fdp(\neg g_1) \sqcup fdp(\neg g_2)$*. La démonstration pour le cas où l'on a deux prédicats ouverts est tout à fait semblable à celle où l'on a deux prédicats fermés.*

□

Exemple 5
Sur la figure 2.8, on a $g(v) = 1 \leq x \leq 5 \vee 6 \leq x \leq 10$ donc $(g\downarrow)(v) = (x = 5) \vee (x = 10)$ ce qui nous donne :

$$
\begin{array}{l}
fdp(g)(v,t) = x + t \leq 1 \vee (x > 5 \wedge x + t \leq 6) \vee x > 10 \vee (t=0) \\
fdp(\neg g)(v,t) = (1 \leq x \wedge x + t \leq 5) \vee (6 \leq x \wedge x + t \leq 10) \vee (t=0) \\
fdp(g\downarrow)(v,t) = x + t \leq 5 \vee (x > 5 \wedge x + t \leq 10) \vee x > 10 \vee (t=0)
\end{array}
$$

On voit que $fdp(g)$ permet de progresser jusqu'en 1, où l'on peut progresser par $fdp(\neg g)$ jusqu'en 5. De même en 6. Par contre aucune de ces deux FDP ne permet de progresser à partir de 5 ou 10. La dissymétrie est engendrée par le fait que g est fermée à droite, mais pas $\neg g$.

Il est ensuite facile de vérifier $fdp(g) \sqcup fdp(\neg g) = fdp(g\downarrow)$.

Ce théorème permet donc un calcul à la volée des FDP. On remarquera néanmoins que ce calcul va faire intervenir l'opérateur $\sqcup$ qui est très coûteux, voire même dans le cas général incalculable (parce que faisant intervenir un nombre non borné de quantificateurs). C'est pourquoi il vaut mieux essayer de réduire les échéances avant de calculer leur FDP.

Voyons donc comment se distribue l'opérateur $\downarrow$.

Propriété 10
Pour toutes gardes g, g_1, g_2, nous avons les relations :

$$\begin{array}{l} VRAI\downarrow = FAUX = FAUX\downarrow \\ (g_1 \wedge g_2)\downarrow = (g_1\downarrow \wedge g_2) \vee (g_1 \wedge g_2\downarrow) \\ (g_1 \vee g_2)\downarrow = (g_1\downarrow \wedge \neg g_2) \vee (\neg g_1 \wedge g_2\downarrow) \vee (g_1\downarrow \wedge g_2\downarrow) \\ (\Box g)\downarrow = FAUX = (\Diamond g)\downarrow \end{array}$$

Preuve 10
On a par définition :

$$g\downarrow(v) = g(v) \wedge \exists t > 0\ .\ \forall t'\ 0 < t' \leq t\ .\ \neg g(v \triangleright t')$$

Cela implique que pour que $g\downarrow$ vaille vrai en un point, il faut que g vaille vrai en ce point et devienne faux plus tard. Donc pour toutes les formules qui restent indéfiniment à vrai dès qu'il y a un point où elles ont valu vrai, le front descendant donne $FAUX$. On a donc :

$$VRAI\downarrow = FAUX = FAUX\downarrow = (\Box g)\downarrow = (\Diamond d)\downarrow$$

Regardons maintenant ce qui se passe pour la disjonction ou la conjonction. Intuitivement, le front descendant de la conjonction de deux gardes est l'ensemble des points où le front descendant d'une garde est vrai et où l'autre garde est vraie. Le front descendant de la disjonction est l'ensemble des points où le front descendant de l'une des gardes est vrai et où l'autre garde est fausse, additionnés des points communs aux deux fronts descendants. Ceci est illustré par la figure 2.9, et démontré par les calculs suivants (qui peuvent paraître relativement rébarbatifs).

En appliquant les définitions :

$$\begin{array}{rl} (g_1 \wedge g_2)\downarrow(v) & = (g_1 \wedge g_2)(v) \wedge \exists t > 0\ .\ \forall t' \in (0,t]\ .\ \neg(g_1 \wedge g_2)(v \triangleright t') \\ & = (g_1 \wedge g_2)(v) \wedge \exists t > 0\ .\ \forall t' \in (0,t]\ .\ (\neg g_1(v \triangleright t') \vee \neg g_2(v \triangleright t')) \end{array}$$

Utilisons le fait que g_1 et g_2 sont bien définies, et donc que les fonctions $\lambda t'.g_1(v \triangleright t')$ et $\lambda t'.g_2(v \triangleright t')$ ne changent qu'un nombre fini de fois de valeur sur $[0,t]$. On a l'implication suivante :

$$(\forall t' \in (0,t]\ .\ \neg g_1(v \triangleright t') \vee \neg g_2(v \triangleright t')) \Rightarrow \exists t_1 \in (0,t]\ .\ \begin{array}{l} (\forall t' \in (0,t_1]\ .\ \neg g_1(v \triangleright t')) \\ \vee \\ (\forall t' \in (0,t_1]\ .\ \neg g_2(v \triangleright t')) \end{array}$$

La réciproque étant évidente, on a l'égalité et on en déduit :

$$\begin{array}{rl} (g_1 \wedge g_2)\downarrow(v) & = (g_1 \wedge g_2)(v) \wedge \exists t > 0\ .(\forall t' \in (0,t]\ .\ \neg g_1(t')) \vee \\ & \qquad (\forall t' \in (0,t]\ .\ \neg g_2(t')) \\ & = g_1(v) \wedge g_2(v) \wedge ((\exists t > 0\ .\ \forall t' \in (0,t]\ .\ \neg g_1(t')) \vee \\ & \qquad (\exists t > 0\ .\ \forall t' \in (0,t]\ .\ \neg g_2(t'))) \\ & = (g_1\downarrow(v) \wedge g_2(v)) \vee (g_1(v) \wedge g_2\downarrow(v)) \end{array}$$

Voilà pour la conjonction !

Et voici pour la disjonction :

$$\begin{aligned}(g_1 \vee g_2) \downarrow (v) &= (g_1 \vee g_2)(v) \wedge \exists t > 0 \,.\, \forall t' \in (0,t] \,.\, \neg(g_1 \vee g_2)(v \triangleright t') \\ &= (g_1 \vee g_2)(v) \wedge \exists t > 0 \,.\, \forall t' \in (0,t] \,.\, (\neg g_1 \wedge \neg g_2)(v \triangleright t') \\ &= (g_1 \vee g_2)(v) \wedge \exists t > 0 \,.\, (\forall t' \in (0,t] \,.\, \neg g_1(v \triangleright t') \vee \\ &\qquad\qquad \forall t' \in (0,t] \,.\, \neg g_2(v \triangleright t'))\end{aligned}$$

Or on a l'équivalence :

$$\begin{aligned}\exists t > 0 \,.\, (\quad & \forall t' \in (0,t] \,.\, f_1(t') \wedge \forall t' \in (0,t] \,.\, f_2(t')) \\ & \Leftrightarrow (\exists t_1 > 0 \,.\, \forall t' \in (0,t_1] \,.\, f_1(t')) \wedge (\exists t_2 > 0 \,.\, \forall t' \in (0,t_2] \,.\, f_2(t'))\end{aligned}$$

On en déduit :

$$\begin{aligned}(g_1 \vee g_2) \downarrow (v) &= (g_1(v) \vee g_2(v)) \wedge (\exists t > 0 \,.\, \forall t' \in (0,t] \,.\, \neg g_1(v \triangleright t') \vee \\ &\qquad\qquad \exists t > 0 \,.\, \forall t' \in (0,t] \,.\, \neg g_2(v \triangleright t')) \\ &= (g_1(v) \wedge \exists t > 0 \,.\, \forall t' \in (0,t] \,.\, \neg g_1(v \triangleright t') \wedge \\ &\qquad \exists t > 0 \,.\, \forall t' \in (0,t] \,.\, \neg g_2(v \triangleright t')) \vee \\ &\quad (g_2(v) \wedge \exists t > 0 \,.\, \forall t' \in (0,t] \,.\, \neg g_1(v \triangleright t') \wedge \\ &\qquad \exists t > 0 \,.\, \forall t' \in (0,t] \,.\, \neg g_2(v \triangleright t')) \\ &= (g_1 \downarrow (v) \wedge \exists t > 0 \,.\, \forall t' \in (0,t] \neg g_2(v \triangleright t')) \vee \\ &\quad (g_2 \downarrow (v) \wedge \exists t > 0 \,.\, \forall t' \in (0,t] \neg g_1(v \triangleright t')) \\ &= (g_1 \downarrow (v) \wedge (g_2(v) \vee \neg g_2(v)) \wedge \exists t > 0 \,.\, \forall t' \in (0,t] \neg g_2(v \triangleright t')) \vee \\ &\quad (g_2 \downarrow (v) \wedge (g_1(v) \vee \neg g_1(v)) \exists t > 0 \,.\, \forall t' \in (0,t] \neg g_1(v \triangleright t')) \\ &= (g_1 \downarrow (v) \wedge (g_2 \downarrow (v) \vee \\ &\qquad\qquad (\neg g_2(v) \exists t > 0 \,.\, \forall t' \in (0,t] \,.\, \neg g_2(v \triangleright t')))) \vee \\ &\quad (g_2 \downarrow (v) \wedge (g_1 \downarrow (v) \vee \\ &\qquad\qquad (\neg g_1(v) \exists t > 0 \,.\, \forall t' \in (0,t] \,.\, \neg g_1(v \triangleright t'))))\end{aligned}$$

Du fait de la fermeture de g_1 et g_2, on a l'implication : $\neg g_1(v) \Rightarrow \exists t > 0 \,.\, \forall t' \in [0,t] \,.\, \neg g_1(v \triangleright t')$. *On en déduit :*

$$(g_1 \vee g_2) \downarrow (v) = (g_1 \downarrow (v) \wedge \neg g_2(v)) \vee (g_2 \downarrow (v) \wedge \neg g_1(v)) \vee (g_1 \downarrow (v) \wedge g_2 \downarrow (v))$$

□

Cette propriété permet de n'avoir à calculer les fronts descendants que de trois types de gardes :

- les gardes atomiques, $p \in P$, que l'on connaît a priori sans avoir aucune composition à faire,
- les gardes de la forme $\Diamond g$,
- les gardes de la forme $\boxdot\, g$.

Pour ces deux derniers cas, le front descendant est facile à calculer puisque ce sont des gardes ayant une seule composante connexe dans le sens du temps, au maximum : elles ne peuvent en aucun cas être vraies à un moment puis fausses après un certain temps puis à nouveau vraies.

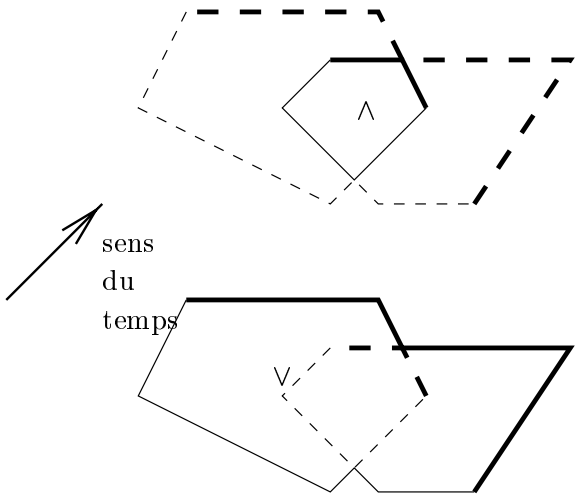


FIG. 2.9 – *Fronts descendants de conjonction et de disjonction*

D'autre part on pourra éviter de calculer certains fronts descendants grâce aux propriétés suivantes.

Propriété 11

$$\Box(g\downarrow) = \boxdot(g\downarrow) = FALSE$$
$$\Diamond(g\downarrow) = \Diamond(g \wedge \neg\Box g)$$
$$\diamondsuit(g\downarrow) = \diamondsuit(\neg g \wedge \diamondsuit g) \vee g\downarrow$$

Ceci nous permettra de limiter au maximum les calculs de fronts descendants au prix, il est vrai, de l'introduction de la négation.

2.6 Conclusion de cette étude

Nous avons vu dans ce chapitre que l'on ne peut prendre n'importe quelle condition sur les valuations pour définir des échéances. Avec la fonction FDP, permettant de calculer les fonctions de progressions du temps à partir des échéances que nous nous sommes données $(fdp(c)(v), t = \forall t' \in [0, t)\ .\ \neg c(v \triangleright t'))$, il faut que les échéances soient fermées à gauche dans le sens du temps.

Nous avons vu que si l'on se limite aux échéances bien définies, alors on peut calculer les fonctions de progression du temps de façon compositionnelle. Cependant ce calcul pose problème, car l'opérateur $\sqcup$ permettant de calculer la FDP d'une disjonction d'échéances coûte cher à calculer, quand il est calculable.

Par contre nous avons vu que le calcul d'échéances de forme modale peut être simplifié, car on peut presque éliminer les occurrences de l'opérateur de front descendant $\downarrow$. En se plaçant dans le cadre des échéances de forme modale, on peut alors espérer réaliser des compositions d'échéances à faible coût.

On remarquera que pour pouvoir avoir des échéances ouvertes à gauche, il aurait fallu prendre une autre fonction pour calculer les FDP :

$$fdp2(c)(v,t) = \forall t' \in [0,t] \ . \ \neg c(v \triangleright t')$$

qui interdit d'atteindre les états où l'échéance est vraie. Or ceci compliquerait le modèle que nous allons voir, où les blocages sont impossibles par le simple fait de n'arrêter le temps que si une action peut être effectuée. En effet, ceci est réalisé en imposant que l'échéance est contenue dans la garde, et cela deviendrait insuffisant si l'on utilise $fdp2$. On pourrait envisager d'imposer une contrainte plus forte, qui permettrait la réactivité temporelle sur ce type de systèmes. Ceci peut être réalisé assez simplement sur le modèle simplifié des ATTT (chapitre 4), mais l'étude complète n'a pas été réalisée dans le cadre du modèle général des SHEC (chapitre 3). Le principe est d'étendre la garde à gauche de l'échéance, de façon à permettre à l'action de s'exécuter avant que le temps ne soit arrêté. Cela interdit donc de spécifier des actions urgentes et ponctuelles (ne pouvant être effectuées qu'à un seul moment précis). Pour cette raison nous avons choisi de ne pas utiliser $fdp2$, mais de rester dans le cadre défini dans ce chapitre.

Chapitre 3

Des systèmes hybrides : les SHEC

Nous venons de constater qu'il y a certaines contraintes à respecter pour spécifier le temps dans les systèmes temporisés. Cela va nous permettre de régler la question de la construction de systèmes temporisés simples en garantissant l'absence de blocage. Par contre, dès qu'il s'agit de composer des systèmes, d'autres problèmes vont surgir, comme nous l'avons évoqué dans l'introduction.

Nous allons voir une méthode pour construire des systèmes hybrides qui satisfont les propriétés de réactivité temporelle et de progrès maximal, avec des opérateurs qui préservent ces propriétés. Nous appellerons ces systèmes les SHEC (Systèmes Hybrides à ÉChéances).

Dans ce chapitre seront exposés des résultats généraux et puissants, mais malheureusement difficilement exploitables tels quels. Nous examinerons au chapitre suivant un cas particulier, qui restreint quelque peu l'expressivité du modèle mais dont l'utilisation est facilitée par une simplification de certains aspects du modèle général.

3.1 Systèmes séquentiels

Nous regarderons dans cette partie comment temporiser un système discret. Nous étudierons d'abord une adaptation de la méthode classique pour étendre le choix non déterministe puis nous nous intéresserons à une généralisation permettant de définir d'autres opérateurs de choix. Nous nous attarderons sur l'opérateur de choix avec priorités dont nous noterons plus tard qu'il est indispensable pour la composition parallèle (voir 3.2.3). Nous verrons aussi l'opérateur de choix consensuel qui ne sera utile que dans certains cas très précis, mais qui prouve aussi que notre modèle offre la possibilité de spécifier par un seul opérateur des comportements relativement complexes.

3.1.1 Une première temporisation

Nous allons temporiser des systèmes discrets en assurant la réactivité temporelle. Grâce à l'utilisation de gardes et d'échéances (cf chapitre précédent),

cela est facile: il suffit de vérifier que les échéances sont contenues dans les gardes. Ainsi si le temps est arrêté c'est qu'une échéance est satisfaite, donc la garde correspondante aussi et une action est nécessairement possible. Le système n'est donc jamais totalement bloqué.

Comment allons nous procéder en pratique? Nous allons définir la temporisation sur un système de transitions étiquetées. Comme les systèmes de transitions servent de modèle sémantique pour nombre de systèmes discrets, cette définition pourra s'appliquer pour définir de nombreuses extensions.

La temporisation se passe en trois étapes: une première étape consiste à ajouter un ensemble de variables X définies sur un ensemble dense B et une fonction d'évolution $\triangleright$, la deuxième étape consiste à modifier les étiquettes du système de transitions et la dernière consiste à donner une signification à ces étiquettes.

Variables et fonction d'évolution

Les variables sont définies classiquement pour mesurer le temps ou des quantités liées au système et évoluant dans le temps. On adoptera les notations suivantes.

Notation 4 *L'ensemble des variables utilisées par le système sera noté X. Le domaine de ces variables sera identique pour toutes les variables et sera noté B. On notera V l'ensemble des valuations, c'est-à-dire l'ensemble des fonctions de X dans B^X.*

Sur ces variables on définira la fonction d'évolution $\triangleright : V \times \mathbf{R}_+ \to V$. Cette fonction d'évolution globale sera divisée en fonctions d'évolution locales à chaque état s, $\triangleright_s : V \to V$.

> **Remarque 8** Le fait d'avoir un seul domaine pour toutes les variables n'est pas discriminant. Cela permet simplement de simplifier les notations.

Modification des étiquettes

Ainsi une transition (s, a, s'), aussi notée $s \xrightarrow{a} s'$, deviendra $(s, (a, g, c, f), s')$ ou bien $s \xrightarrow{a,g,c,f} s'$. Il restera à définir la signification de cette nouvelle étiquette.

> **Définition 11** *Transition temporisée*
> *À toute transition discrète (s, a, s') on associera une transition temporisée $(s, (a, g, c, f), s')$ où g et c sont des prédicats sur V et f une fonction de V dans V. g sera appelée la* garde *de la transition, c sera l'*échéance *de la transition et f la* fonction de transfert *de la transition. On exigera de g et c qu'elles soient fermées à gauche dans le sens du temps (cf définition 6) et formées d'un nombre fini de composantes connexes. De plus on aura l'implication $c \Rightarrow g$.*

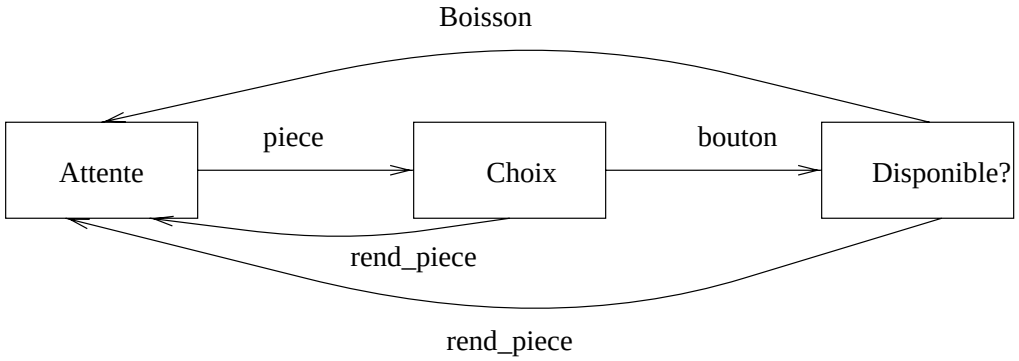


FIG. 3.1 – *Le distributeur de boissons (non temporisé)*

Système de transitions temporisé

Il nous reste maintenant à définir la signification de ces étiquettes. Cela se fera en construisant un système de transitions qui tiendra compte du temps et donc d'un contexte temporel.

Définition 12 *État temporisé*
Pour un système de transitions discret donné, un état temporisé est un couple (s, v) où s est un état du système discret et v est une valuation (v appartient à V défini ci-dessus).

Nous définirons alors un système de transitions pour ces états temporisés grâce aux règles suivantes.

Définition 13 *SHEC - sémantique*
Étant donnés un système discret SD, un ensemble de variables X, un ensemble de valuations V, une fonction d'évolution et une temporisation pour chacune des transitions, on définit le système hybride SH par les deux règles d'induction :

$$\frac{s \overset{a,g,c,f}{\rightarrow} s' \quad et \quad g(v)}{(s,v) \overset{a}{\rightarrow} (s', f(v))} \qquad \frac{\forall t' \in [0,t) \;.\; \neg \bigvee_{s \overset{a,g,c,f}{\rightarrow} s'} c(v \triangleright t')}{(s,v) \overset{t}{\rightarrow} (s, v \triangleright t)}$$

On remarquera que la première règle spécifie la façon dont sont effectuées les actions discrètes et la seconde spécifie le passage du temps.

L'exemple suivant montre la temporisation d'un système rudimentaire.

Exemple 6
Cherchons à spécifier un distributeur de boissons. Comment se comporte-t-il ? Au repos, il attend que l'on introduise une pièce. Ensuite, il attend que l'on choisisse une boisson (en appuyant sur le bouton correspondant) ou alors, si l'on attend trop longtemps, il rend la pièce. Enfin après avoir vérifié si la boisson est disponible il la donne ou il rend la pièce. Ce peut être modélisé par l'automate suivant (figure 3.1).

Si l'on veut temporiser, il faut regarder maintenant chaque transition et voir les conditions qui leur sont appliquées. On remarquera d'abord que l'on

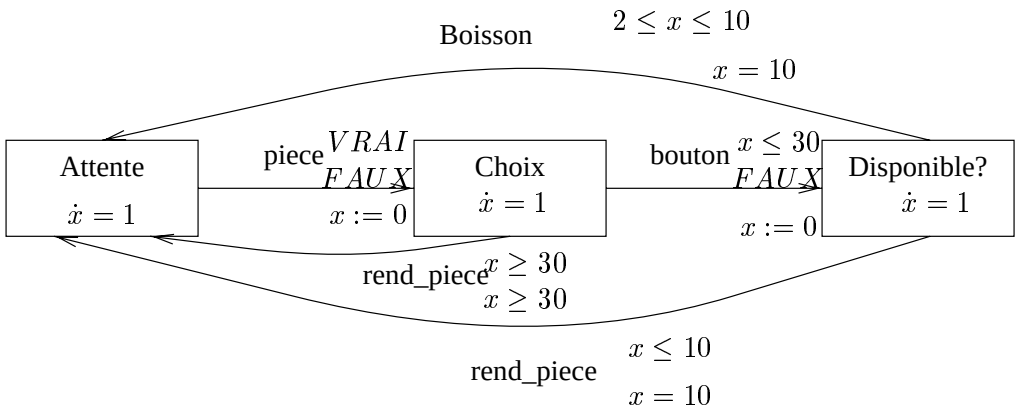


FIG. 3.2 – *Le distributeur de boissons (temporisé)*

ne cherche qu'à mesurer le temps, et uniquement le temps écoulé depuis la dernière action. Une seule horloge est donc nécessaire, nous l'appellerons x. Aucune condition ne peut être appliquée quant au moment où l'on va introduire la pièce (si le distributeur est en attente), donc la garde est $VRAI$ et l'échéance est $FAUX$. Par contre on va mesurer le temps écoulé depuis l'introduction de la pièce, on va donc remettre l'horloge à zéro, ce qui est indiqué par la ligne $x := 0$. Ensuite, on a 30 secondes pour appuyer sur le bouton, sans obligation de le faire. L'horloge est aussi remise à zéro ici pour mesurer le temps de réaction avant de distribuer la boisson ou de rendre la pièce. Rendre la pièce est une action qui doit être effectuée rapidement, c'est pour cela que l'échéance est égale à la garde : dès que l'action est possible il faut réagir. Ceci nous donne l'automate temporisé de la figure 3.2.

On remarquera que dans l'état *choix*, pour $x = 30$, il peut se passer deux choses : ou bien le bouton est enfoncé et l'on regarde si la boisson est disponible, ou bien on rend la pièce. Cette dernière action peut être effectuée même si le bouton est enfoncé. Ceci est imposé par le fait que les gardes sont fermées. On peut voir cela comme une limite du système. Cependant la simultanéité stricte est quelque chose qu'il est difficile d'appréhender en pratique, et l'on peut concevoir que si l'on accepte l'événement qu'est le bouton, c'est que celui-ci a été enfoncé juste avant la date $x = 30$, mais de façon non mesurable, par contre si le distributeur rend la pièce, c'est que les 30 secondes se sont effectivement écoulées. Cela correspond à une certaine façon de voir l'écoulement du temps : on ne peut isoler un instant précis dans le passage du temps, et des erreurs non mesurables sont toujours possibles. On tiendra bien entendu compte du fait de ce qu'on est ici au niveau de la spécification, et que l'implantation d'un tel système pourra par exemple éviter que l'on tienne compte du bouton pour $x = 30$.

En soi ce modèle est pour l'instant identique aux modèles habituels. Il est en effet facile de passer des échéances aux conditions de progression du temps et vice versa. Si l'utilisation des échéances nécessite d'écrire une condition pour chaque transition, elle permet de ne pas avoir à calculer la condition de pro-

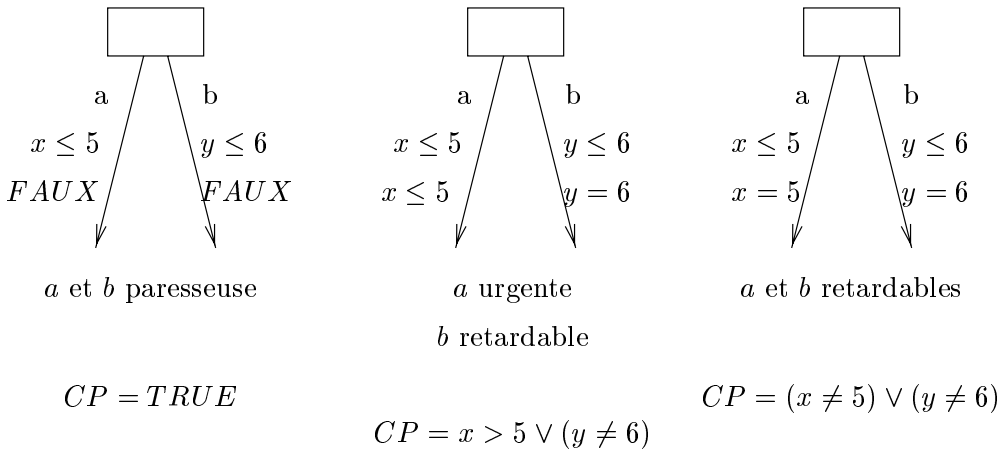


FIG. 3.3 – *Exemple de spécification de l'urgence*

gression du temps lorsque l'on connaît l'urgence de chaque action, comme sur l'exemple de la figure 3.3. Nous allons voir de plus que nous pouvons affiner la temporisation de façon à obtenir différents comportements à partir d'un même ensemble de transitions issues du même état. Il faut pour cela définir différentes manières de temporiser le choix non déterministe, ce qui revient à modifier les règles d'inférence utilisées pour définir les systèmes de transitions temporisés.

3.1.2 Opérateurs de choix

Nous considérerons ici que la garde et l'échéance d'une action définissent le comportement maximal de cette action, à savoir que l'action ne peut être exécutée que si sa garde est vraie et elle ne peut devenir urgente que si l'échéance est vraie. Par contre, une telle action mise en concurrence avec d'autres actions pourra voir son comportement limité et devenir équivalente à une action de garde et échéance plus petites que la garde et l'échéance initiales. Ce sera fait à travers des opérateurs de choix, qui sont des extensions temporelles du choix non déterministe. Nous nous baserons donc sur l'opérateur de choix non déterministe, noté $\sum$, des algèbres de processus.

Notation 5 *Un état noté $\sum_{i \in I} a_i.s_i$ peut effectuer chacune des actions a_i, menant dans l'état s_i. Ainsi, si à partir de s on a les deux transitions $s \xrightarrow{a_1} s_1$ et $s \xrightarrow{a_2} s_2$, on pourra écrire $s = \sum_{i \in \{1,2\}} a_i.s_i$, ou encore dans ce cas précis $s = a_1.s_1 + a_2.s_2$.*

Nous verrons que dans le cas temporisé, il est possible de définir plusieurs équivalents de $\sum$, comme illustré figure 3.4. Nous verrons donc dans un premier temps quelles sont les règles générales que nous voulons respecter lors de la définition des opérateurs de choix, puis dans un second temps nous examinerons deux exemples intéressants : le choix avec priorités et le choix consensuel.

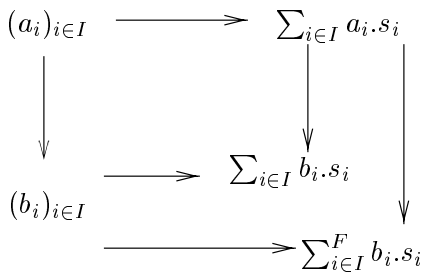


FIG. 3.4 – *Choix temporisés*

Généralités

Nous allons définir des opérateurs de choix sur le modèle suivant. Pour un état donné, nous considérerons toutes les transitions issues de cet état, et nous modifierons la temporisation de chaque transition en tenant compte de toutes les transitions issues de l'état.

Voyons alors la forme générale que nous adopterons pour les opérateurs de choix.

Définition 14 *Opérateurs de choix*
Un opérateur de choix $\sum^F$ est associé à la fonction de restriction $F : B^* \to B^*$, telle que pour tout vecteur fini d'actions temporisées $(a_i, g_i, c_i, f_i)_{i\in I}$, $F((a_i, g_i, c_i, f_i)_{i\in I})$ est de la forme $(a_i, g'_i, c'_i, f_i)_{i\in I}$ avec pour tout i de I, $g'_i \Rightarrow g_i$ et $c'_i \Rightarrow c_i$. L'opérateur $\sum^F$ est alors défini par la formule :

$$\sum_{i\in I}^{F} (a_i, g_i, c_i, f_i).s_i = \sum_{i\in I} (a_i, g'_i, c'_i, f_i).s_i$$

en respectant les notations précédentes
(i.e. $F((a_i, g_i, c_i, f_i)_{i\in I}) = (a_i, g'_i, c'_i, f_i)_{i\in I}$).

On définit le choix $\sum^F$ comme étant le choix non déterministe appliqué à des actions restreintes par F. F est appelée la fonction de restriction du choix $\sum^F$, car elle transforme un vecteur de n actions en un autre vecteur de n actions où seules sont modifiées les gardes et les échéances. Nous verrons plus loin que l'on ne peut pas prendre n'importe quelle fonction de restriction. Mais déjà la définition apporte les précisions suivantes :

- les gardes obtenues après restrictions (g'_i) ne dépendent que des gardes avant restrictions, mais chaque garde peut être modifiée par toutes les autres.

- pour les échéances, il en va de même, sauf que l'on veut préserver la réactivité temporelle ($c'_i \Rightarrow g'_i$), donc les échéances sont aussi affectées par les gardes.

Si nous nous contentons de cette définition, nous aurons la possibilité de définir des choix qui ne font rien ($g'_i = FAUX$). Ceci ne nous intéresse pas. Plus

précisément, nous imposerons de conserver les possibilités d'actions, éventuellement avec un certain retard. Cette exigence répond au besoin que si le système est spécifié comme actif (pouvant effectuer une action dans l'avenir), alors nous obtenons effectivement un système actif. Nous appellerons cette propriété la conservation de l'activité.

Définition 15 *Conservation de l'additivité*
Une fonction F utilisée pour définir un opérateur de choix conserve l'activité si la formule suivante est vérifiée :

$$\bigvee_{i\in I} \Diamond g'_i = \bigvee_{i\in I} \Diamond g_i$$

avec les notations de la définition précédente.

Nous ne nous intéresserons qu'aux opérateurs de choix définis grâce à des fonctions conservant l'activité. D'autres opérateurs peuvent être envisagés, mais au risque d'avoir des situations où la garde d'une transition est vraie, alors que le seul comportement possible pour le système est d'attendre indéfiniment. Ceci ne semble pas raisonnable.

Voyons maintenant deux exemples de ce que l'on peut définir comme choix. Le premier, le choix avec priorités, se révélera d'une importance capitale par la suite (voir chapitre 3.2.3). Le second, le choix consensuel, est un peu plus exotique, mais nous verrons que dans certains cas nous pouvons l'utiliser pour modéliser un partage de ressources.

Choix avec priorités

Nous voulons, par le moyen ce choix, avoir un moyen simple de spécifier plusieurs actions possibles à partir d'un état donné, en favorisant certaines de ces actions. Favoriser une action peut se faire de plusieurs manières. Nous utiliserons des priorités : une action a sera prioritaire sur une autre, b, si lorsque a peut être effectuée b est prohibée. Ainsi on limite b pour que a puisse se faire plus sûrement, mais cette limitation peut être étendue. En effet, dans le cas temporisé où nous nous plaçons, on peut empêcher b d'être effectuée si l'on sait que dans l'avenir a sera effectuée.

Pour formaliser cette notion, nous commencerons par définir un ordre de priorités, à partir duquel nous pourrons définir le choix.

Définition 16 *Ordre de priorités*
Un ordre de priorités sur l'alphabet A est une relation $<$ sur $A\times\mathbf{N}\cup\{\infty\}\times A$ satisfaisant les propriétés données ci-dessous. On notera $a <_k b$ le fait que le triplet (a,k,b) appartienne à $<$. $<_k$ peut alors être vue comme une relation sur A.

$<$ doit satisfaire :

- *pour tout $k \in \mathbf{N}\cup\{\infty\}$, $<_k$ est une relation d'ordre partiel sur A.*
- *si pour un k donné on a $a_1 <_k a_2$, alors pour tout k' inférieur à k, $a_1 <_{k'} a_2$.*
- *si $a_1 <_k a_2$ et $a_2 <_l a_3$ alors $a_1 <_{k+l} a_3$.*

On a alors trivialement la propriété :

Propriété 12
La relation $\ll$ définie par $a_1 \ll a_2 = \exists k \in \mathbf{N} \cup \{\infty\} \ . \ a_1 <_k a_2$ est une relation d'ordre.

Nous allons maintenant utiliser cet ordre de priorités pour définir des priorités entre actions. Les priorités auront une certaine portée dans le temps. Ainsi $a_1 <_k a_2$ signifie que lorsque la garde de a_2 est vraie ou sera vraie avant k unité de temps alors a_1 est inhibée. Si k vaut ∞, alors il suffit que la garde de a_2 soit vraie dans l'avenir pour que a_1 ne puisse pas être exécutée. Si k vaut 0, la priorité est immédiate: a_1 est inhibée uniquement si la garde de a_2 est vraie.

Pour exprimer ces propriétés, nous avons besoin d'étendre les opérateurs temporels définis en 2.5.2. C'est pourquoi nous posons les définitions suivantes.

Définition 17 *Opérateurs modaux*
Les opérateurs $\Diamond_k$ et $\Box_k$ sont définis comme suit pour tout prédicat p sur les valuations et tout $k \in \mathbf{N} \cup \{\infty\}$:

$$\begin{aligned}(\Diamond_k p)(v) &= \exists t \in [0,k] \ . \ p(v \triangleright t)\\ (\Box_k p)(v) &= \forall t \in [0,k] \ . \ p(v \triangleright t)\end{aligned}$$

On remarquera que $\Diamond_\infty = \Diamond$ et $\Box_\infty = \Box$ (voir la définition en 2.5.2). De plus $\Diamond_0 p = p = \Box_0 p$ et pour tout k $\Diamond_k p = \neg(\Box \neg p)$.

Ceci nous permet de définir les opérations de restriction pour l'opérateur de choix avec priorités.

Définition 18 *Choix avec priorités*
Étant donné un ordre de priorités $<$, on définit le choix avec priorités, $\sum^<$ de la façon suivante :

$$\sum_{i \in I}^{<} (a_i, g_i, c_i, f_i).s_i = \sum_{i \in I} (a_i, {g'}_i, {c'}_i, f_i)$$

avec

$$\begin{aligned}{g'}_i &= g_i \wedge \textstyle\bigwedge_{j \in I, a_i <_k a_j} \neg \Diamond_k g_j\\ {c'}_i &= c_i \wedge \textstyle\bigwedge_{j \in I, a_i <_k a_j} \neg \Diamond_k g_j\end{aligned}$$

Pour ne pas surcharger les notations, il n'est pas indiqué dans les définitions ci-dessus que k dépend de i et j : il s'agit de l'entier maximum tel que $a_i <_k a_j$ ou de ∞ si $a_i <_\infty a_j$. Si l'on ne peut trouver un tel k, alors il n'y a aucun terme en $\neg \Diamond_k g_j$. Ceci est normal puisque a_j n'étant pas prioritaire sur a_i, g_j n'intervient pas dans le calcul de ${g'}_i$.

Notation 6 *Pour les cas où il n'y aurait que deux transitions en jeu, on utilisera l'opérateur binaire $<+$, défini par :*

$$\sum_{i \in \{1,2\}}^{<} b_i.s_i = b_1.s_1 <+ b_2.s_2$$

Attention, malgré la notation qui peut induire en erreur, $<+$ est bien commutatif !

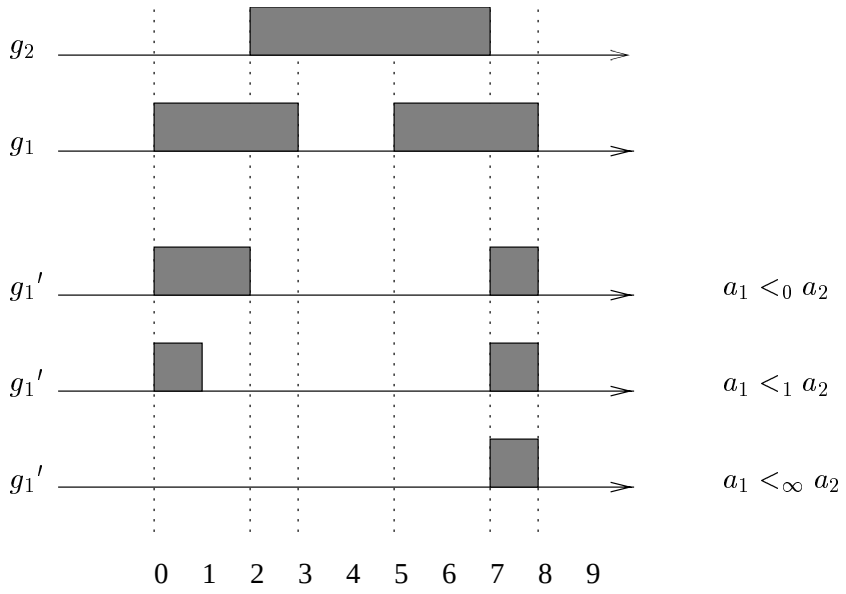


FIG. 3.5 – *Différentes priorités*

Un exemple de ce que peut donner une telle définition est donné en figure 3.5, pour deux actions a_1 et a_2 de gardes respectives g_1 et g_2.

> **Remarque 9** Cette définition peut induire des gardes et échéances ouvertes à gauches, à cause de la négation. Cependant, si l'on considère l'ensemble des actions, il est facile de vérifier que l'union des gardes est fermée à gauche (cf exemple suivant), donc la réactivité temporelle est maintenue.

Exemple 7
Ne considérons que deux transitions a_1 et a_2, a_2 ayant une priorité immédiate sur a_1. Supposons que a_1 ait pour garde $g_1 = 6 \leq x \leq 10$ et pour échéance $c_1 = 8 \leq x \leq 10$, et que a_2 ait pour garde $g_2 = 3 \leq x \leq 9$ et pour échéance $c_2 = 8 \leq x \leq 9$. a_2 étant prioritaire verra sa garde et son échéance inchangées. Par contre pour a_1, on obtient :

$$g'_1 = (6 \leq x \leq 10) \wedge \neg(3 \leq x \leq 9) = 9 < x \leq 10$$
$$c'_1 = (8 \leq x \leq 10) \wedge (9 < x \leq 10) = 9 < x \leq 10$$

Ainsi, si l'on considère uniquement a_1, en $x = 9$ le temps est bloqué (s'il progresse on entre dans c'_1), et a_1 n'est pas possible (g'_1 est fausse en ce point). Par contre si l'on tient compte de a_2, il n'y a plus de problème, car a_2 est possible en $x = 9$.

Si la priorité n'avait pas été immédiate, cela n'aurait rien changé. Intuitivement, la négation induit une garde ouverte à gauche pour g_1, uniquement sur le front descendant (frontière arrière) de $\diamond_k g_2$, c'est-à-dire lorsque g_2 est satisfaite. Donc l'union est fermée à gauche (cf figure 3.6).

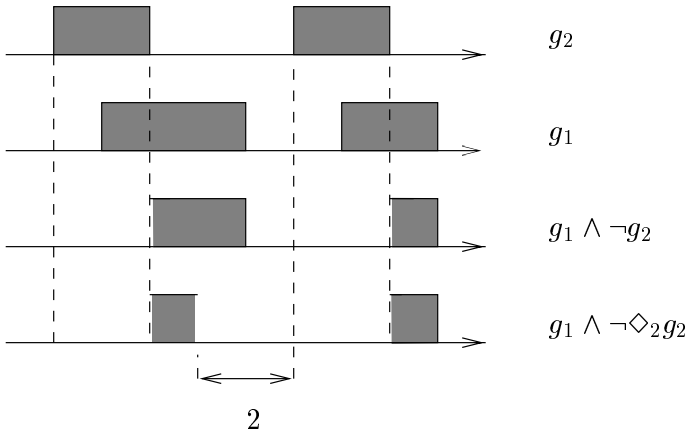


FIG. 3.6 – *Fermeture pour la priorité*

Tout cela vient du fait que pour une garde g fermée à gauche, $g \vee \neg \Diamond_k g$ est fermée à gauche.

Propriété 13
Pour toute garde g fermée à gauche, et tout $k \in \mathbf{N} \cup \{\infty\}$, le prédicat $g \vee \neg \Diamond_k g$ est fermé à gauche.

Preuve 13
Supposons que pour une valuation v on a $(g \vee \neg \Diamond_k g)(v) = faux$. Cela implique $g(v) = faux$ et $\Diamond_k g(v) = vrai$. g étant fermée à gauche, il existe $t > 0$, tel que pour tout $t' \in (0, t]$ on ait $g(v \triangleright t') = faux$. On a alors nécessairement $\Diamond_k g(v \triangleright t') = vrai$, et par conséquent $(g \vee \neg \Diamond_k g)(v \triangleright t') = faux$.
□

On remarquera de plus que les c'_i ainsi définis satisfont $c'_i = c_i \wedge g'_i$. Une autre façon de définir c'_i aurait été :

$$c'_i = c_i \wedge g'_i \wedge \bigwedge_{j \in I, a_i <_k a_j} \neg \Diamond_k c_j$$

Mais il est facile de voir que ces deux définitions donnent le même résultat.

Proposition 14

$$c_i \wedge g'_i \wedge \bigwedge_{j \in I, a_i <_k a_j} \neg \Diamond_k c_j = c_i \wedge \bigwedge_{j \in I, a_i <_k a_j} \neg \Diamond_k g_j$$

Preuve 14
En effet, on a :

$$\begin{aligned} c_i \wedge g'_i \quad & \wedge \bigwedge_{j \in I, a_i <_k a_j} \neg \Diamond_k c_j \\ & = c_i \wedge g_i \wedge \bigwedge_{j \in I, a_i <_k a_j} \neg \Diamond_k g_j \wedge \bigwedge_{j \in I, a_i <_k a_j} \neg \Diamond_k c_j \\ & = c_i \wedge g_i \wedge \bigwedge_{j \in I, a_i <_k a_j} (\neg \Diamond_k g_j \wedge \neg \Diamond_k c_j) \\ & = c_i \wedge g_i \wedge \bigwedge_{j \in I, a_i <_k a_j} \neg \Diamond_k (g_j \vee c_j) \\ & = c_i \wedge \bigwedge_{j \in I, a_i <_k a_j} \neg \Diamond_k g_j \end{aligned}$$

car $c \Rightarrow g$.
□

Pour des raisons pratiques, nous allons chercher à savoir comment ce choix interagit avec le choix non déterministe. En effet, dans certains cas il sera intéressant de grouper des transitions par le choix non déterministe avant de regarder les priorités; ce sera notamment le cas si l'on décrit le système par des algèbres de processus. On cherche donc à effectuer un choix plus complexe en combinant le choix non déterministe et le choix avec priorités.

La raison cachée de ce nouveau choix est que l'on veut que le choix avec priorités soit associatif. Cela n'aurait aucun sens si l'on se limitait à la définition précédente, on étendra donc cette définition pour capter la situation décrite figure 3.7, qui correspond à $(\sum_{i \in [0..n]} b_i.s_i) <+ (\sum_{i \in [n+1..m]} b_i.s_i)$. L'associativité permet alors de combiner des actions en plusieurs fois à l'aide du choix avec priorités, et le résultat sera indépendant de cet ordre. Il s'agit donc d'une facilité de spécification. Nous verrons aussi que cette associativité sera très utile lorsque nous modéliserons la composition parallèle.

Définition 19 *Choix avec priorités - cas général*
Nous conviendrons que la valeur de
$(\sum_{i \in I}(a_i, g_i, c_i, f_i).s_i) <+ (\sum_{j \in J}(a_j, g_j, c_j, f_j).s_J)$ *est*
$(\sum_{i \in I}(a_i, g_{i/J}, c_{i/J}, f_i).s_i) + (\sum_{j \in J}(a_j, g_{j/I}, c_{j/I}, f_j).s_j)$ *avec*

$$\begin{aligned} g_{i/J} & = g_i \wedge \bigwedge_{j \in J, a_i <_k a_j} \neg \Diamond_k g_j \\ c_{i/J} & = c_i \wedge \bigwedge_{j \in J, a_i <_k a_j} \neg \Diamond_k g_j \end{aligned}$$

$g_{j/I}$ *et* $c_{j/I}$ *sont définis de manière similaire.*

Nous avons donc choisi dans ce cas de ne restreindre chaque transition que par rapport aux actions qui ne sont pas du même côté de l'opérateur $<+$.

Attention, ceci implique que $<+$ n'est pas distributif par rapport à $+$:

$$(b_1.s_1 + b_2.s_2) <+ b_3.s_3 \neq (b_1.s_1 <+ b_3.s_3) + (b_2.s_2 <+ b_3.s_3)$$

En effet, si a_3 (étiquette de b_3) est l'action la moins prioritaire, alors dans $(b_1.s_1 + b_2.s_2) <+ b_3.s_3$, b_3 sera restreinte par les b_1 et b_2 simultanément, tandis que dans $(b_1.s_1 <+ b_3.s_3) + (b_2.s_2 <b_3.s_3)$, b_3 est restreinte par b_1 et b_2 séparément puis on fait l'union de ces deux comportements. Dans le premier cas le comportement de b_3 sera restreint plus encore que pour $b_1.s_1 <+ b_3.s_3)$, alors

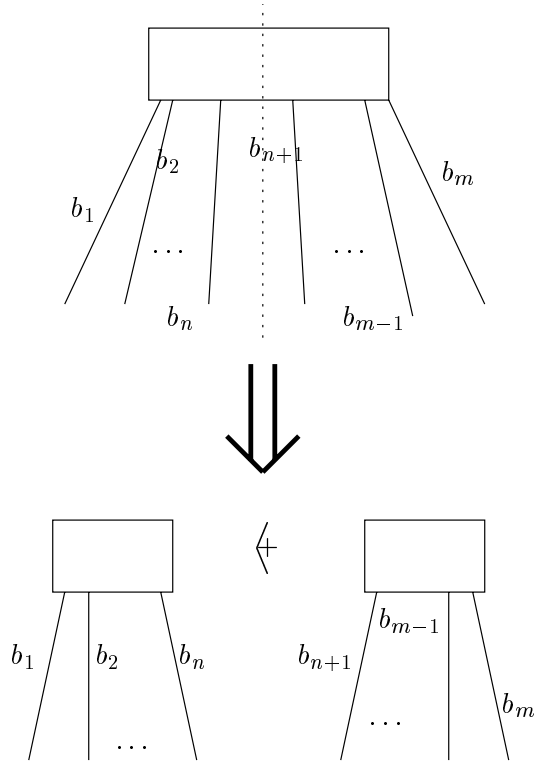


FIG. 3.7 – *Choix avec priorités étendu*

que c'est le contraire pour le second cas.

Néanmoins, cette définition a deux avantages. Tout d'abord, il est facile de généraliser l'opérateur pour obtenir une somme n-aire et non plus simplement binaire. De plus on peut vérifier une propriété d'associativité.

Propriété 15
Si l'on note b_i les actions temporisées (a_i, g_i, c_i, f_i), on a :

$$\begin{array}{ll}((\sum_{i\in I} b_i.s_i) <+(\sum_{j\in J} b_j.s_j)) & <+(\sum_{k\in K} b_k.s_k) = \\ & (\sum_{i\in I} b_i.s_i) <+((\sum_{j\in J} b_j.s_j) <+(\sum_{k\in K} b_k.s_k)\end{array}$$

Preuve 15
On va en fait montrer que $((\sum_{i\in I} b_i.s_i) <+(\sum_{j\in J} b_j.s_j)) <+(\sum_{k\in K} b_k.s_k)$ *est égal au résultat de l'opérateur ternaire appliqué aux termes* $\sum_{i\in I} b_i.s_i$, $\sum_{j\in J} b_j.s_j$ *et* $\sum_{k\in K} b_k.s_k$. *L'opérateur* $<+$ *étant commutatif, cela sera suffisant pour montrer son associativité.*

Calculons donc ce que cela vaut :

$$\begin{array}{ll}((\sum_{i\in I} b_i.s_i) & <+(\sum_{j\in J} b_j.s_j)) <+(\sum_{k\in K} b_k.s_k) \\ & = ((\sum_{i\in I} {b'}_i.s_i) + (\sum_{j\in J} {b'}_j.s_j)) <+(\sum_{k\in K} b_k.s_k) \\ & = ((\sum_{i\in I} {b''}_i.s_i) + (\sum_{j\in J} {b''}_j.s_j)) <+(\sum_{k\in K} {b''}_k.s_k)\end{array}$$

avec

$$\begin{array}{rl}{g'}_i & = g_i \wedge \bigwedge_{j\in J, a_i <_l a_j} \neg\Diamond_l(g_j) \\ {g'}_j & = g_j \wedge \bigwedge_{i\in I, a_j <_l a_i} \neg\Diamond_l(g_i) \\ {g''}_i & = {g'}_i \wedge \bigwedge_{k\in K, a_i <_l a_k} \neg\Diamond_l(g_k) \\ {g''}_j & = {g'}_j \wedge \bigwedge_{k\in K, a_j <_l a_k} \neg\Diamond_l(g_k) \\ {g''}_k & = g_k \wedge \bigwedge_{i\in I, a_k <_l a_i} \neg\Diamond_l({g'}_i) \wedge \bigwedge_{j\in J, a_k <_l a_j} \neg\Diamond_l({g'}_j)\end{array}$$

Ce que l'on veut montrer c'est que :

$$\begin{array}{rl}{g''}_i & = g_i \wedge \bigwedge_{j\in J, a_i <_l a_j} \neg\Diamond_l(g_j) \wedge \bigwedge_{k\in K, a_i <_l a_k} \neg\Diamond_l(g_k) \\ {g''}_j & = g_j \wedge \bigwedge_{i\in I, a_j <_l a_i} \neg\Diamond_l(g_i) \wedge \bigwedge_{k\in K, a_j <_l a_k} \neg\Diamond_l(g_k) \\ {g''}_k & = g_k \wedge \bigwedge_{i\in I, a_k <_l a_i} \neg\Diamond_l(g_i) \wedge \bigwedge_{j\in J, a_k <_l a_j} \neg\Diamond_l(g_j)\end{array}$$

Les deux premières égalités sont immédiates. La dernière est plus dure à vérifier. Tout d'abord constatons qu'il suffit de montrer :

$$\bigvee_{i\in I\cup J, a_k <_l a_i} \Diamond_l({g'}_i) = \bigvee_{i\in I\cup J, a_k <_l a_i} \Diamond_l(g_i)$$

L'égalité précédente en découle alors trivialement. Cette dernière propriété est donnée par le lemme suivant.
□

Lemme 16 *Pour tout ordre de priorité* $<$ *sur un ensemble d'actions* $\{a_i\}_{i\in I}$, *on a pour tout* $i \in I$ *:*

$$\bigvee_{j\in I, a_i <_l a_j} \Diamond_l(g_j \wedge \bigwedge_{k\in I, a_j <_m a_k} \neg\Diamond_m g_k) = \bigvee_{j\in I, a_i <_l a_j} \Diamond_l(g_j)$$

Preuve 16
Montrons le résultat par récurrence sur l'ordre inverse de $<$.

- *Si a_i est maximal pour $<$, il n'y a rien à montrer.*
- *Supposons que pour tout $a_{i'}$ supérieur à a_i (dans l'ordre $<$), on ait*

$$\bigvee_{j\in I, a_{i'}<_l a_j} \Diamond_l(g_j \wedge \bigwedge_{k\in I, a_j<_m a_k} \neg\Diamond_m g_k) = \bigvee_{j\in I, a_{i'}<_l a_j} \Diamond_l(g_j)$$

Montrons que cette égalité est vraie pour a_i. Par transitivité de $<$ et du fait que $\Diamond_l\Diamond_m g = \Diamond_{l+m} g$, on a :

$$\begin{aligned}
\bigvee_{j\in I, a_i<_{l_{ij}} a_j} \Diamond_{l_{ij}} \quad & (g_j \wedge \bigwedge_{k\in I, a_j<_{l_{jk}} a_k} \neg\Diamond_{l_{jk}} g_k) \\
&= \bigvee_{j\in I, a_i<_{l_{ij}} a_j} (\Diamond_{l_{ij}}(g_j \wedge \bigwedge_{k\in I, a_j<_{l_{jk}} a_k} \neg\Diamond_{l_{jk}} g_k) \vee \\
&\quad \bigvee_{j'\in I, a_j<_{l_{jj'}} a_{j'}} \Diamond_{l_{ij'}}(g_{j'} \wedge \bigwedge_{k\in I, a_{j'}<_{l_{j'k}} a_k} \neg\Diamond_{l_{j'k}} g_k)) \\
&= \bigvee_{j\in I, a_i<_{l_{ij}} a_j} \Diamond_{l_{ij}}((g_j \wedge \bigwedge_{k\in I, a_j<_{l_{jk}} a_k} \neg\Diamond_{l_{jk}} g_k) \vee \\
&\quad \bigvee_{j'\in I, a_j<_{l_{jj'}} a_{j'}} \Diamond_{l_{jj'}}(g_{j'} \wedge \bigwedge_{k\in I, a_{j'}<_{l_{j'k}} a_k} \neg\Diamond_{l_{j'k}} g_k)) \\
&= \bigvee_{j\in I, a_i<_{l_{ij}} a_j} \Diamond_{l_{ij}}((g_j \wedge \bigwedge_{k\in I, a_j<_{l_{jk}} a_k} \neg\Diamond_{l_{jk}} g_k) \vee \\
&\quad \bigvee_{j'\in I, a_j<_{l_{jj'}} a_{j'}} \Diamond_{l_{jj'}} g_{j'}) \\
&= \bigvee_{j\in I, a_i<_{l_{ij}} a_j} \Diamond_{l_{ij}}(g_j \vee \bigvee_{k\in I, a_j<_{l_{jk}} a_k} \Diamond_{l_{jk}} g_k) \\
&= \bigvee_{j\in I, a_i<_{l_{ij}} a_j} (\Diamond_{l_{ij}} g_j \vee \bigvee_{k\in I, a_j<_{l_{jk}} a_k} \Diamond_{l_{ij}}\Diamond_{l_{jk}} g_k) \\
&= \bigvee_{j\in I, a_i<_{l_{ij}} a_j} \Diamond_{l_{ij}} g_j
\end{aligned}$$

□

La propriété précédente permet donc d'affirmer que l'opérateur de choix avec priorités tel que nous l'avons défini est un opérateur associatif. De plus le lemme va permettre de montrer que cet opérateur conserve l'activité.

Propriété 17
Tout opérateur de choix conserve l'activité.

Preuve 17
Le lemme précédent nous donne :

$$\bigwedge_{a_i<_l a_j} \Diamond_l(g'_j) = \bigwedge_{a_i<_l a_j} \Diamond_l(g_j)$$

or, par définition,

$$g'_i = g_i \wedge \neg \bigvee_{a_i<_l a_j} \Diamond_l(g_j)$$

Nous avons donc :

$$\begin{aligned}
g'_i \vee \bigvee_{a_i<_l a_j} \Diamond_l(g'_j) \quad & = g_i \wedge \neg \bigvee_{a_i<_l a_j} \Diamond_l(g_j) \vee \bigvee_{a_i<_l a_j} \Diamond_l(g_j) \\
& = g_i \vee \bigvee_{a_i<_l a_j} \Diamond_l(g_j)
\end{aligned}$$

En appliquant $\Diamond$ qui est distributif par rapport à la disjonction :

$$\Diamond g'_i \vee \bigvee_{a_i <_l a_j} \Diamond g'_j = \Diamond g_i \vee \bigvee_{a_i <_l a_j} \Diamond g_j$$

En faisant l'union sur toutes les actions intervenant dans le processus :

$$\bigvee_{i \in I} \Diamond g'_i = \bigvee_{i \in I} \Diamond g_i$$

□

Un autre propriété importante de ce choix est qu'il conserve l'équivalence entre systèmes donnée par le remplacement d'une transition par deux transitions de même comportement. Plus précisément, nous avons la propriété suivante.

Proposition 18
La scission d'une transition (s, a, g, c, f, s') en deux transitions (s, a, g_1, c_1, f, s') et (s, a, g_2, c_2, f, s'), avec $g = g_1 \vee g_2$ et $c = c_1 \vee c_2$, a le même effet que le choix soit non déterministe ou avec priorités.

Preuve 18
Par définition du choix avec priorités, on a :

$$\sum_{i \in I}^{<} (a_i, g_i, c_i, f_i).s_i = \sum_{i \in I} (a_i, g'_i, c'_i, f_i)$$

avec

$$\begin{aligned} g'_i &= g_i \wedge \textstyle\bigwedge_{j \in I, a_i <_k a_j} \neg \Diamond_k g_j \\ c'_i &= c_i \wedge \textstyle\bigwedge_{j \in I, a_i <_k a_j} \neg \Diamond_k g_j \end{aligned}$$

Comme $\neg \Diamond_k g_{j,1} \wedge \neg \Diamond_k g_{j,2} = \neg \Diamond_k (g_{j,1} \vee g_{j,2})$, il est évident que scinder une transition n'affecte pas la façon dont les autres transitions sont modifiées.
De plus, si l'on scinde la transition i en $i,1$ et $i,2$, avec $g_i = g_{i,1} \vee g_{i,2}$ et $c_i = c_{i,1} \vee c_{i,2}$, comme $i,1$ et $i,2$ ont même étiquette, elles seront incomparables entre elles, et auront les même priorités que la transition i. Il est alors évident que les gardes et échéances transformées satisferont : $g'_i = g'_{i,1} \vee g'_{i,2}$ et $c'_i = c'_{i,1} \vee c'_{i,2}$.
□

Nous verrons par la suite que cet opérateur est capital pour la composition. Mais examinons maintenant un autre opérateur de choix.

3.1.3 Choix consensuel

Nous définirons ici un choix très spécifique correspondant à la situation suivante. Le système est composé d'un serveur pouvant servir deux clients avec des contraintes spécifiques à chaque client(figure 3.8). On veut assouplir ces contraintes pour ne favoriser aucun client, on réduira donc les échéances correspondantes de façon à ne bloquer le temps que dans les deux cas suivants :

– les deux clients sont d'accord pour bloquer le temps.

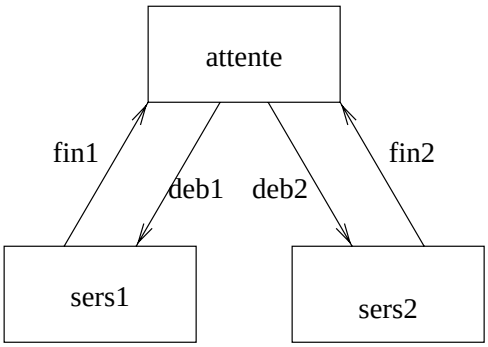


FIG. 3.8 – *Schéma du système*

– l'un des clients n'aura plus besoin de bloquer le temps, l'autre pourra alors le faire normalement.

Ainsi un client ne peut bloquer le temps pour obliger le serveur à le servir que si l'autre client est hors course. On réduira chaque échéance en tenant compte de l'échéance de l'autre action. On aurait pu affiner encore en tenant compte de la garde de l'autre action, mais le but de cet exercice est uniquement de faire un choix qui ne fonctionne que sur les échéances et ignore les gardes. L'inverse est impossible, car si l'on réduit les gardes on est obligé de réduire les échéances.

On cherche donc à obtenir des échéances vérifiant l'équation suivante:

$$\begin{aligned} c_1 &= c_1 \wedge (c_2 \vee \Box\neg c_2) \\ c_2 &= c_2 \wedge (c_1 \vee \Box\neg c_1) \end{aligned}$$

Ceci est bien sûr une équation de point fixe. À cause des négations, on ne peut appliquer les méthodes classiques de recherche du plus grand ou plus petit point fixe, qui nécessitent des fonctions monotones. Néanmoins, on cherchera à calculer les échéances réduites en remplaçant c_1 et c_2 dans les équations précédentes, au moins partiellement. C'est ainsi qu'a été trouvée la solution suivante.

Proposition 19
Si c_1 et c_2 sont des échéances, les échéances c'_1 et c'_2 obtenues par

$$\begin{aligned} c'_1 &= c_1 \wedge (c_2 \vee \Box(\neg c_2 \vee (\neg c_1 \wedge \Diamond c_1))) \\ c'_2 &= c_2 \wedge (c_1 \vee \Box(\neg c_1 \vee (\neg c_2 \wedge \Diamond c_2))) \end{aligned}$$

satisfont l'équation précédente. Ce couple forme la solution maximale (pour l'implication) satisfaisant $\Diamond(c'_1 \vee c'_2) = \Diamond(c_1 \vee c_2)$ et inférieur à (c_1, c_2).

Preuve 19
Nous allons vérifier que c'_1 et c'_2 satisfont chacune des propriétés de la proposition.

– *Il est évident que $c'_1 \Rightarrow c_1$ et $c'_2 \Rightarrow c_2$.*

– *Voyons maintenant que* $\Diamond(c'_1 \vee c'_2) = \Diamond(c_1 \vee c_2)$. *L'implication* $\Diamond(c'_1 \vee c'_2) \Rightarrow \Diamond(c_1 \vee c_2)$ *est évidente, montrons l'implication inverse. Supposons pour cela que pour la valuation* v, *on a* $\Diamond c_1(v) = vrai$, *et montrons que* $\Diamond(c'_1 \vee c'_2)(v) = vrai$. c_1 *et* c_2 *ayant un nombre fini de composantes connexes, il existe une dernière composante connexe dans le sens du temps. Soit* t_1, *tel que* $v \triangleright t_1$ *appartienne à la dernière composante connexe de* c_1, *on a donc* $\Box(c_1 \vee \Box\neg c_1)$. *Et par conséquent :* $c'_1(v \triangleright t_1) = (c_1 \wedge (c_2 \vee \Box\neg c_2))(v \triangleright t_1) = (c_2 \vee \Box\neg c_2)(v \triangleright t_1)$. *Le seul cas où* $c'_1(v \triangleright t_1)$ *vaut faux est si* $(\neg c_2 \wedge \Diamond c_2)(v \triangleright t_1)$. *Il existe alors une date* t_2 *supérieure à* t_1, *telle que* $v \triangleright t_2$ *soit dans la dernière composante connexe de* t_2. *Comme précédemment on a* $c'_2(v \triangleright t_2) = (c_1 \vee \Box\neg c_1)(v \triangleright t_2)$, *ce qui vaut vrai car* $t_2 > t_1$. *Si l'on résume, on a soit* $c'_1(v \triangleright t_1)$, *soit* $c'_2(v \triangleright t_2)$. *Dans tous les cas, on obtient :* $\Diamond(c'_1 \vee c'_2)(v)$.

– *Le couple* (c'_1, c'_2) *est solution de l'équation : vue la symétrie de l'expression, il suffit de vérifier que* $c'_1 = c'_1 \wedge (c'_2 \vee \Box\neg c'_2)$. *Nous allons en fait montrer que* $c'_1 \wedge \neg c'_2 \wedge \Diamond c'_2 = FALSE$. *On a en effet :*

$$\begin{aligned}
\Diamond c'_2 &= \Diamond((c_1 \wedge c_2) \vee (c_2 \wedge \Box(\neg c_1 \wedge (\neg c_2 \vee \Diamond c_2)))) \\
&= \Diamond(c_1 \wedge c_2) \vee \Diamond(c_2 \wedge \Box(\neg c_1 \wedge (\neg c_2 \vee \Diamond c_2))) \\
\neg c'_2 &= \neg(c_1 \wedge c_2) \wedge \neg(c_2 \wedge \Box(\neg c_1 \vee (\neg c_2 \wedge \Diamond c_2))) \\
c'_1 &= (c_1 \wedge c_2) \vee (c_1 \wedge \Box(\neg c_2 \vee (\neg c_1 \wedge \Diamond c_1)))
\end{aligned}$$

On en déduit :

$$\begin{aligned}
c'_1 \wedge \neg c'_2 &= c_1 \wedge \Box(\neg c_2 \vee (\neg c_1 \wedge \Diamond c_1)) \wedge \neg(c_1 \wedge c_2) \wedge \\
&\quad \neg(c_2 \wedge \Box(\neg c_1 \vee (\neg c_2 \wedge \Diamond c_2))) \\
&= c_1 \wedge \neg c_2 \wedge \Box(\neg c_2 \vee (\neg c_1 \wedge \Diamond c_1)) \wedge \\
&\quad \neg(c_2 \wedge \Box(\neg c_1 \vee (\neg c_2 \wedge \Diamond c_2))) \\
&= c_1 \wedge \neg c_2 \wedge \Box(\neg c_2 \vee (\neg c_1 \wedge \Diamond c_1))
\end{aligned}$$

On remarquera que $\Box(\neg c_2 \vee (\neg c_1 \wedge \Diamond c_1))$ *implique* $\Box\neg(c_1 \wedge c_2)$. *D'où :*

$$c'_1 \wedge \neg c'_2 \wedge \Diamond c'_2 = c_1 \wedge \neg c_2 \wedge \Box(\neg c_2 \vee (\neg c_1 \wedge \Diamond c_1)) \wedge \Diamond(c_2 \wedge \Box(\neg c_1 \vee (\neg c_2 \wedge \Diamond c_2)))$$

Nous allons montrer par l'absurde que cette expression vaut $FAUX$. *Supposons en effet qu'il existe une valuation* v *telle que* $(c'_1 \wedge \neg c'_2 \wedge \Diamond c'_2)(v) = vrai$. *Nécessairement, il existe* t_1 *positif tel que* $(c_2 \wedge \Box(\neg c_1 \vee (\neg c_2 \wedge \Diamond c_2)))(v \triangleright t_1) = vrai$. *De plus, on a* $\Box(\neg c_2 \vee (\neg c_1 \Diamond c_1))(v)$ *donc aussi* $\Box(\neg c_2 \vee (\neg c_1 \Diamond c_1))(v \triangleright t_1)$. $v \triangleright t_1$ *satisfait donc* $c_2 \vee \neg c_1$. *Il existe alors* $t_2 > t_1$ *tel que* $c_1(v \triangleright t_2)$, *et aussi* $(\Box(\neg c_2 \vee (\neg c_1 \wedge c_1)))(v \triangleright t_2)$, $(\Box(\neg c_1 \vee (\neg c_2 \wedge c_2)))(v \triangleright t_2)$ *et* $\neg c_2(v \triangleright t_2)$. *On peut recommencer et construire une suite infinie* (t_i) *telle que*

$$(c_1 \vee \neg c_2)(v \triangleright t_{2i}) = vrai = (c_2 \vee \neg c_1)(v \triangleright t_{2i+1})$$

Ceci étant incompatible avec le fait que c_1 *et* c_2 *ont un nombre fini de composantes connexes, l'hypothèse* $(c'_1 \wedge \neg c'_2 \wedge \Diamond c'_2)(v) = vrai$ *est fausse.*

– (c'_1, c'_2) *est solution maximale : supposons que ce ne soit pas le cas, il existerait un couple* (c''_1, c''_2) *qui soit solution et tel que* $c'_1 \Rightarrow c''_1 \Rightarrow c_1$ *et* $c'_2 \Rightarrow c''_2 \Rightarrow c_2$. *De plus, on aura* $c''_1 \neq c'_1$ *ou* $c''_2 \neq c'_2$ *(au moins pour certains couples* (c_1, c_2)*). Supposons alors qu'il existe une valuation* v *telle que* $(c''_1 \wedge \neg c'_1)(v) = vrai$. *On a alors aussi* $c_1(v) = vrai$. *Par définition de* c'_1, v *satisfait*

$$\neg c_2 \wedge \Diamond(c_2 \wedge (c_1 \vee \Box\neg c_1))$$

Il existe alors $t_0 > 0$ *tel que* $(c_2 \wedge (c_1 \vee \Box\neg c_1))(v \triangleright t_0) = vrai$. *On a de plus* $c_2(v) = faux$, *donc* $c''_2(v) = faux$, *et comme* (c''_1, c''_2) *vérifie* $c''_1 = c''_1 \wedge (c''_2 \vee \Box\neg c''_2)$, *on a* $\Diamond\neg c''_2(v) = vrai$. *Ceci implique* $\neg c''_2(v \triangleright t_0) = vrai$ *et donc* $\neg c'_2(v \triangleright t_0)$. $v \triangleright t_0$ *satisfait donc* $c_2 \wedge \neg c'_2$. *On peut alors recommencer l'opération précédente et obtenir une suite* (t_i) *telle que :*

$$(c_1 \wedge \neg c'_1)(v \triangleright t_{2i+1}) \qquad\qquad (c_2 \wedge \neg c'_2)(v \triangleright t_{2i})$$

Comme $(c_1 \wedge \neg c'_1) \Rightarrow c_1 \wedge \neg c_2$, *cette suite contredit le fait que* c_2 *ait un nombre fini de composantes connexes. Donc* $c''_1 \Rightarrow c'_1$. *D'où* $c''_1 = c'_1$ *et de même* $c''_2 = c'_2$. (c'_1, c'_2) *est donc maximal.*

□

Une autre propriété intéressante de cet opérateur est que la fermeture à gauche est conservée :

Propriété 20
Si c_1 et c_2 sont fermés à gauche alors $c_1 \wedge (c_2 \vee \Diamond(\neg c_2 \wedge (\neg c_1 \vee \Diamond c_1)))$ est fermé à gauche.

Preuve 20
Nous allons vérifier que $c_2 \vee \Diamond(\neg c_2 \wedge (\neg c_1 \vee \Diamond c_1))$ *est bien fermé à gauche. Supposons donc que pour une valuation* v *on ait*

$$\neg(c_2 \vee \Diamond(\neg c_2 \wedge (\neg c_1 \vee \Diamond c_1)))(v) = vrai$$

v *satisfait alors :*

$$\neg c_2 \wedge \Box(c_2 \vee (c_1 \wedge \Box\neg c_1))$$

c_2 *étant fermé, il existe* $t > 0$ *tel que :*

$$\forall t' \in [0, t] \;.\; \neg c(v \triangleright t')$$

Comme de plus on a $\Box(c_2 \vee (c_1 \wedge \Box\neg c_1))(v)$, *on obtient :*

$$\forall t' \in [0, t] \;.\; \Box(c_2 \vee (c_1 \wedge \Box\neg c_1))(v \triangleright t')$$

Et par conséquent :

$$\forall t' \in [0, t] \;.\; \neg(c_2 \vee \Diamond(\neg c_2 \wedge (\neg c_1 \vee \Diamond c_1)))(v \triangleright t')$$

$c_1 \wedge (c_2 \vee \Diamond(\neg c_2 \wedge (\neg c_1 \vee \Diamond c_1)))$ *est donc fermé à gauche, ce qui permet de conclure quant à la propriété.*
□

Ce choix est défini comme un opérateur binaire, mais on peut généraliser à un nombre quelconque d'opérandes, il suffit alors de prendre la définition suivante :

$$c'_i = c_i \wedge \bigwedge_{j \neq i} (c_j \vee \Box(\neg c_j \vee (\neg c_i \wedge \Diamond c_i)))$$

Les propriétés ci-dessus sont conservées (il est relativement simple d'adapter les preuves). On a donc obtenu un opérateur de choix consensuel n-aire.

Cet opérateur, s'il n'est applicable que dans certains cas n'en pas moins intéressant, car il montre que l'on peut faire facilement spécifier des comportements relativement complexes avec un unique opérateur. Comme ce choix se traduit en un choix non déterministe, on aurait très bien pu représenter le comportement souhaité avec ce choix non déterministe, néanmoins cela aurait nécessité des calculs préalables, et ce pour chaque état où l'on veut utiliser ce type de choix. On peut ainsi définir des opérateurs complexes permettant de spécifier plus simplement certains comportements.

3.2 Compositions parallèles

3.2.1 Introduction

Nous venons de voir que l'on peut spécifier des comportements relativement complexes avec des opérateurs de choix. Cependant d'autres comportements peuvent être envisagés, qui ne correspondent pas à des opérateurs de choix. De plus, on peut décomposer des systèmes complexes en systèmes plus simples que l'on voudra relier par la suite. C'est là le rôle des opérateurs de composition parallèle.

Les opérateurs classiques, comme nous l'avons constate sur l'exemple 1.4, posent quelques problèmes dûs à la façon de spécifier le temps. Nous verrons ici comment construire des opérateurs de composition parallèle qui conservent la réactivité temporelle, tout en garantissant le progrès maximal.

Nous nous baserons pour cela sur des systèmes ne comportant que des opérateurs de choix avec priorités. Cela peut sembler restrictif, néanmoins si l'on remarque que le choix non déterministe est un choix avec priorité où toutes les actions sont incomparables entre elles, on voit que l'on capture la plupart des systèmes. D'autre part, tous les opérateurs de choix que nous avons définis précédemment se ramènent à des opérateurs de choix non déterministe. On pourra donc spécifier les composants d'un système avec des opérateurs de choix quelconques, les traduire en systèmes avec des opérateurs de choix non déterministes et enfin appliquer les compositions.

D'autre part, l'idée importante que nous retrouverons ici est que l'on doit relier les comportement temporels aux actions, même dans les compositions parallèles. Cela nous donnera des compositions plus flexibles que la composition habituelle, en ce sens que l'on admettra d'infléchir les exigences que l'on a imposées sur le passage du temps, et ceci afin d'éviter les blocages. Comme les conditions de progression du temps sont liées à la définition de l'urgence des actions, cela signifie que l'on admet de diminuer cette urgence.

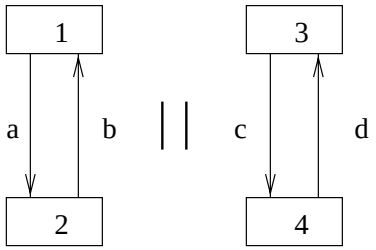


FIG. 3.9 – *Automates à composer*

Cette approche est en opposition avec une approche de vérification, puisque la vérification chercherait à détecter ces blocages. Par contre pour spécifier des systèmes, elle apporte la facilité que l'absence de blocage est assurée. Cette propriété pourra être utilisée si l'on cherche à faire de la synthèse de systèmes.

3.2.2 Composition de systèmes discrets

Regardons d'abord comment faire pour des systèmes non temporisés. Nous étendrons ensuite la méthode aux systèmes hybrides.

Pour composer deux systèmes non temporisés, nous utiliserons la méthode suivante. Nous commencerons par spécifier pour chaque couple d'actions si ces actions synchronisent ou non, et si oui, quel est le résultat de cette synchronisation. Cela se fera avec l'opérateur $|$: si a et b ne synchronisent pas on aura $a|b = \perp$ sinon, $a|b$ sera l'étiquette de la synchronisation de a et b. On appellera $A|$ l'ensemble des étiquettes obtenues en combinant des étiquettes de A par un nombre quelconque de synchronisations.

Le système de transitions résultant est alors obtenu à partir des systèmes de transitions des composants par les règles suivantes :

$$\frac{p_1 \xrightarrow{a} p'_1}{p_1 \| p_2 \xrightarrow{a} p'_1 \| p_2} \qquad \frac{p_2 \xrightarrow{b} p'_2}{p_1 \| p_2 \xrightarrow{b} p_1 \| p'_2}$$

$$\frac{p_1 \xrightarrow{a} p'_1 \ , \ p_2 \xrightarrow{b} p'_2 \ , \ a|b \neq \perp}{p_1 \| p_2 \xrightarrow{a|b} p'_1 \| p'_2}$$

Regardons ce que cela donne pour l'exemple suivant décrit par des automates.

Exemple 8
Sur les automates de la figure 3.9, nous prendrons les synchronisations suivantes : $a|d = \perp$, $a|c = ac$, $b|c = bc$, $b|d = bd$. Les systèmes de transitions sont les suivants :

$$1 \xrightarrow{a} 2 \quad 3 \xrightarrow{c} 4$$
$$2 \xrightarrow{b} 1 \quad 4 \xrightarrow{d} 3$$

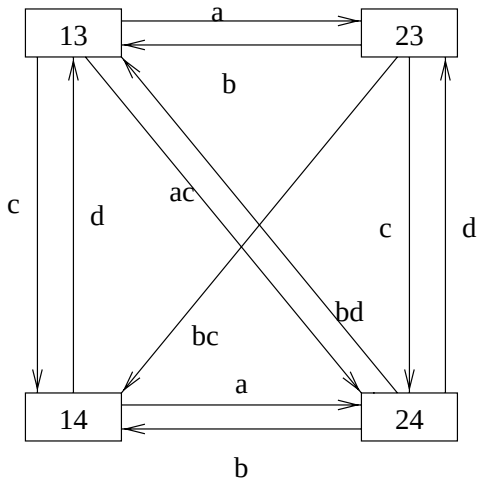


FIG. 3.10 – *Automate composé*

Si l'on note 13 l'état $1 \parallel 3$, le résultat est alors :

$$\begin{array}{lll} 13 \xrightarrow{a} 23 & 13 \xrightarrow{c} 14 & 13 \xrightarrow{ac} 24 \\ 23 \xrightarrow{b} 13 & 23 \xrightarrow{c} 24 & 23 \xrightarrow{bc} 14 \\ 14 \xrightarrow{a} 24 & 14 \xrightarrow{d} 13 & \\ 24 \xrightarrow{b} 14 & 24 \xrightarrow{d} 23 & 24 \xrightarrow{bd} 13 \end{array}$$

Ce qui donne l'automate de la figure 3.10.

> **Remarque 10** On remarquera que ce modèle permet la synchronisation de deux actions et l'avancement indépendant des deux composants. On peut interdire la synchronisation, mais en aucun cas on ne peut interdire à un composant de progresser isolément. Ceci peut être gênant, mais nous verrons que dans le cas temporisé le problème peut être réglé.

En général, les systèmes un peu complexes nécessitent plusieurs composants. On cherchera alors à ce que le résultat ne dépende pas de l'ordre de composition. Si l'on exige que $\mid$ soit un opérateur commutatif, associatif et ayant $\perp$ comme élément absorbant, alors la composition parallèle est commutative et associative.

3.2.3 Composition des SHEC

Schéma général

Pour les systèmes hybrides, nous garderons les mêmes transitions, mais il faut maintenant définir les gardes, échéances, etc... de ces transitions. Rappelons nous les propriétés que nous exigeons de la composition. Nous voulons que, si dans un état l'un des systèmes, pris isolément, peut effectuer une action, alors le système composé puisse aussi faire une action (synchronisation ou échappement); nous voulons de plus favoriser la synchronisation (propriété de progrès maximal).

Pour cela nous allons bien sûr utiliser les priorités. Ainsi on utilisera le schéma suivant, où $b_1 = (a_1, g_1, c_1, f_1)$ et $b_2 = (a_2, g_2, c_2, f_2)$:

$$b_1.p_1 \parallel b_2.p_2 = b_1.(p_1 \parallel b_2.p_2) <+ b_2.(b_1.p_1 \parallel p_2) <+ (b_1 | b_2)(p_1 \parallel p_2)$$

Si nous voulons utiliser ce schéma, il nous faut étendre la définition de $|$ aux actions temporisées. Nous allons voir comment procéder.

Composition des actions temporisées

Le principe de synchronisation des actions temporisées est une simple extension de la synchronisation des actions non temporisées. Cela signifie que l'on définira $(a_1, g_1, c_1, f_1) | (a_2, g_2, c_2, f_2)$ comme $(a_1 | a_2, g_1 | g_2, c_1 | c_2, f_1 | f_2)$. $a_1 | a_2$ est défini précédemment; si $a_1 | a_2$ vaut $\perp$, alors la synchronisation est impossible.

Les gardes et échéances composées posent un certains nombre de problèmes, aussi les étudierons-nous plus loin en détail. Si les composants ne partagent aucune variable, alors on notera (v_1, v_2) la juxtaposition de deux valuations v_1 et v_2, et nous utiliserons la définition :

$$f_1 | f_2(v_1, v_2) = (f_1(v_1), f_2(v_2))$$

Nous nous limiterons à ce cas. Techniquement il est possible (et facile) de spécifier des variables partagées (il suffit de vérifier la cohérence des évolutions des variables partagées dans les deux composants). Mais les variables partagées pouvant poser de nombreux problèmes, en particulier lors de la vérification, nous n'utiliserons pas cette possibilité.

Passons donc à la composition des gardes et des échéances. Nous ne donnerons ici que des principes généraux. Des exemples seront donnés au chapitre suivant. Tout d'abord intéressons nous aux gardes. Il nous paraît nécessaire de respecter un certain principe de causalité : si une synchronisation peut se produire, c'est que l'une au moins des actions composantes peut se produire. Pour une synchronisation, nous pourrions exiger que les deux actions composantes s'effectuent pour que la synchronisation ait lieu, mais nous étendrons la notion de synchronisation pour pouvoir capter d'autres formes de coordination, comme les interruptions par exemple. Nous exigerons aussi que si les deux composantes d'une synchronisation sont prêtes à coopérer, alors la composition peut se faire. En d'autres termes nous demandons que :

$$g_1 \wedge g_2 \Rightarrow g_1 | g_2 \Rightarrow g_1 \vee g_2$$

Nous exigerons de même :

$$c_1 \wedge c_2 \Rightarrow c_1 | c_2 \Rightarrow (c_1 \vee c_2) \wedge (g_1 | g_2)$$

Remarque 11 Il faut bien noter que $g_1 \mid g_2$ (resp. $c_1 \mid c_2$) ne dépend pas uniquement de g_1 et g_2 (resp. c_1 et c_2). En effet, suivant les étiquettes a_1 et a_2 des actions à composer, on pourra vouloir effectuer un traitement

différent sur les gardes et les échéances. De plus pour assurer la réactivité temporelle, il faut relier $c_1 \mid c_2$ à $g_1 \mid g_2$. Ceci n'est pas uniquement un détail technique, mais représente une modification fondamentale par rapport à l'approche traditionnelle de la composition. En effet, traditionnellement le comportement temporel du système composé était calculé sans tenir compte des actions de ce système. C'est là la cause des blocages que l'on pouvait obtenir. En faisant intervenir les gardes dans le calcul des échéances, nous tenons compte des actions dans le calcul du comportement temporel. Revenons aux sources de la composition pour comprendre un peu mieux les mécanismes mis en jeu. Sur les systèmes discrets, on a défini une composition parallèle que l'on a étendu aux systèmes temporisés. Or cette composition des systèmes discrets était valable pour des processus communicants, c'est-à-dire sans variables partagées, les communications s'effectuant lors de la synchronisation d'actions. Le problème de la temporisation est que le temps est une variable partagée. Cette variable n'apparaît peut-être pas directement dans la modélisation des systèmes, mais elle régit l'évolution des variables continues de chacun des systèmes et est utilisée par ce biais dans tous les composants. Nous apportons ici une méthode pour composer les systèmes en tenant compte de ce fait.

En adoptant les contraintes précédentes, on peut vérifier que $b_1.p_1 \parallel b_2.p_2$ a la même activité que $b_1.p_1 + b_2.p_2$. En effet

$$b_1.p_1 \parallel b_2.p_2 = b_1.(p_1 \parallel b_2.p_2) <+ b_2.(b_1.p_1 \parallel p_2) <+ (b_1 | b_2).(p_1 \parallel p_2)$$

Donc son activité est :

$$\Diamond g_1 \vee \Diamond g_2 \vee \Diamond(g_1 | g_2)$$

qui vaut trivialement $\Diamond(g_1 \vee g_2)$. Cela nous permet de garantir que localement la composition ne "perd pas d'action", au sens où si une action est possible avant composition, inévitablement une action sera possible dans la composition.

3.2.4 Garde de la synchronisation

Les contraintes que nous avons données pour la synchronisation sont très faibles. Si nous nous en tenions à cette situation, nous aurions certes un très grand choix de synchronisations possibles, mais au prix de la perte d'un certain nombre de propriétés. Ces propriétés, que nous allons voir ici, nous semblent suffisamment importantes pour que l'on élimine les synchronisations qui ne les respectent pas. Regardons d'abord ce que nous exigerons sur les gardes.

L'une des exigences que l'on attendra de la composition parallèle est qu'elle préserve une certaine forme d'équivalences entre systèmes. Si S_1 est équivalent à S_2 (en un sens que l'on verra ci-dessous), on voudra que $S_1 \parallel S$ soit équivalent à $S_2 \parallel S$, pour tout SHEC S.

L'une des équivalences que l'on peut souhaiter garder, est que scinder une transition en deux transitions spécifiant le même comportement temporel donne un système équivalent. Plus précisément, si S' est le système S où l'on a rem-

placé la transition $s \overset{(a,g,c,f)}{\rightarrow} s'$ par les transitions $s \overset{(a,g_1,c_1,f)}{\rightarrow} s'$ et $s \overset{(a,g_2,c_2,f)}{\rightarrow} s'$ telles que $g = g_1 \vee g_2$ et $c = c_1 \vee c_2$ alors S et S' sont équivalents.

Il peut paraître étrange de demander cette équivalence, alors que l'un des choix que nous avons présentés (le choix consensuel), ne la respecte pas. Néanmoins, observons que les deux choix qui servent de base à notre modèle, le choix non déterministe et le choix avec priorités, la respecte. Pour le choix non déterministe, c'est trivial d'après la sémantique des SHEC. Pour le choix avec priorités, nous l'avons vu avec la propriété 18. Il paraît alors raisonnable de demander que la composition parallèle préserve cette équivalence. Pour cette raison, nous limiterons les gardes composées à celles de la forme suivante.

Définition 20 *Synchronisation des gardes*
La garde de la composition de deux transitions de gardes g_1 et g_2 seront de la forme :

$$g_1 | g_2 = (g_1 \wedge m_2(g_2)) \vee (m_1(g_1) \wedge g_2)$$

avec des fonctions m_1 et m_2 satisfaisant :

- $\forall x \ . \ x \Rightarrow m_i(x)$
- $\forall x, y \ . \ m_i(x \vee y) = m_i(x) \vee m_i(y)$

Ceci nous garantit que la synchronisation satisfait :

$$g_1 \wedge g_2 \Rightarrow g_1 | g_2 \Rightarrow g_1 \vee g_2$$

On a alors la propriété voulue :

Propriété 21
Les synchronisations de la forme précédente satisfont :

$$(g_1 \vee g_2) | g_3 = (g_1 | g_3) \vee (g_2 | g_3)$$

Preuve 21
Par distributivité de m_1, on a :

$$\begin{array}{rcl} (g_1 \vee g_2) | g_3 & = & ((g_1 \vee g_2) \wedge m_2(g_3)) \vee (m_1(g_1 \vee g_2) \wedge g_3) \\ & = & (g_1 \wedge m_2(g_3)) \vee (m_1(g_1) \wedge g_3) \vee \\ & & (g_2 \wedge m_2(g_3)) \vee (m_1(g_2) \wedge g_3) \\ & = & (g_1 | g_3) \vee (g_2 | g_3) \end{array}$$

□

3.2.5 Synchronisation des échéances

Nous imposerons une valeur pour l'échéance composée. En effet, seule l'échéance $c_1 | c_2 = (g_1 | g_2) \wedge (c_1 \vee c_2)$ semble raisonnable. Toutes les autres échéances donneraient des synchronisations qui perdraient de l'urgence. Or, par définition de la composition parallèle, l'urgence est déjà réduite à cause du choix avec priorités que nous utilisons. En outre, cela permet de simplifier la définition

d'une synchronisation. En effet, après avoir défini l'étiquette de la synchronisation, il ne reste qu'à calculer sa garde pour la définir complètement. Nous utiliserons donc la notion suivante.

Définition 21 *Modes de synchronisation*
Nous appellerons mode de synchronisation l'opérateur qui à (g_1, c_1) *et* (g_2, c_2) *associe le couple* $(g_1 | g_2, c_1 | c_2)$ *et on le notera* $mode((g_1, c_1), (g_2, c_2))$.

Ceci assure la propriété suivante.

Propriété 22
L'échéance $c_1 | c_2$ peut être calculée directement par :

$$c_1 | c_2 = (c_1 \wedge m_2(g_2)) \vee (m_1(g_1) \wedge c_2)$$

Preuve 22
Comme on a $c_i \Rightarrow g_i \Rightarrow m_i(g_i)$, *on a:*

$$\begin{aligned} c_1 | c_2 &= (g_1 | g_2) \wedge (c_1 \vee c_2) \\ &= ((g_1 \wedge m_2(g_2)) \vee (m_1(g_1) \wedge g_2)) \wedge (c_1 \vee c_2) \\ &= (c_1 \wedge m_2(g_2)) \vee (c_1 \wedge g_2) \vee (g_1 \wedge c_2) \vee (m_1(g_1) \vee c_2) \\ &= (c_1 \wedge m_2(g_2)) \vee (m_1(g_1) \wedge c_2) \end{aligned}$$

□

On peut aussi vérifier que l'échéance calculée est préservée par scission d'une transition.

Propriété 23

$$\begin{aligned} mode((g_1, c_1) \vee (g_2, c_2), \quad & (g_3, c_3)) = \\ & mode((g_1, c_1), g_3, c_3) \vee mode((g_2, c_2), (g_3, c_3)) \end{aligned}$$

Preuve 23
Il suffit de montrer que $(g_1 \vee g_2) | g_3 = (g_1 | g_3) \vee (g_2 | g_3)$ *et* $(c_1 \vee c_2) | c_3 = (c_1 | c_3) \vee (c_2 | c_3)$. *La première égalité a été montrée dans la propriété 21. La seconde se montre de même.*

$$\begin{aligned} (c_1 \vee c_2) | c_3 &= ((c_1 \vee c_2) \wedge m_2(g_3)) \vee (m_1(g_1 \vee g_2) \wedge c_3) \\ &= (c_1 \wedge m_2(g_3)) \vee (m_1(g_1) \wedge c_3) \vee \\ &\quad (c_2 \wedge m_2(g_3)) \vee (m_1(g_2) \wedge c_3) \\ &= (c_1 | c_3) \wedge (c_2 | c_3) \end{aligned}$$

□

Enfin, nous pouvons généraliser le schéma de composition. En effet, pour de nombreux modes les opérateurs obtenus ne sont pas associatifs. Il faut donc définir des opérateurs n-aires, correspondant à la composition de n systèmes en parallèle. La définition précédente se généralise alors pour un ensemble fini de transitions.

Définition 22 *Mode - cas général*
Si l'on note G la garde de la transition composée pour les actions $(a_i)_{i \in I}$,

il suffit de prendre :

$$G = \bigvee_{i\in I}(g_i \wedge \bigwedge_{j\in I, j\neq i} m_j(g_j))$$
$$mode((g_i)_{i\in I}, (c_i)_{i\in I}) = (G, G \wedge \bigvee_{i\in I} c_i)$$

avec les mêmes contraintes sur les m_j.

On peut vérifier que les propriétés concernant le calcul des échéances et la scission de transitions sont conservées.

3.2.6 Composition de plus de deux actions

Nous venons de voir comment composer deux processus qui ne débutent que par une seule action, et non par un choix. L'idée intuitive pour réaliser la composition parallèle dans le cas de processus plus complexes serait alors :

$$\begin{aligned}\textstyle\sum_{i\in I} b_i.p_i \parallel \sum_{j\in J} b_j.p_j = \quad & \textstyle\sum_{i\in I} b_i.(p_i \parallel \sum_{j\in J} b_j.p_j) \\ & <+ \textstyle\sum_{j\in J} b_j.(\sum_{i\in I} b_i.p_i \parallel p_j) \\ & <+ \textstyle\sum_{i\in I, j\in J, a_i \mid a_j \neq \perp} (b_i \mid b_j).(p_i \parallel p_j)\end{aligned}$$

Cependant, cette définition pose un gros problème : même dans le cas très simple où la synchronisation est définie par $g_i \mid g_j = g_i \wedge g_j$, on n'obtient pas un opérateur associatif. En effet, à cause des priorités, les gardes des actions qui s'effectuent isolément sont modifiées, et lors d'une composition ultérieure ce ne sont plus les mêmes actions qui sont composées !

Le problème vient du fait que l'on mélange deux types de choix : le choix non déterministe et le choix avec priorités. Comme on veut garder le choix avec priorités pour avoir la propriété de progrès maximal, on va changer la définition pour ne garder que des choix de ce type.

Définition 23 *Composition parallèle*
Nous définirons l'opérateur de composition parallèle associé à la fonction de synchronisation $\mid$ par la formule suivante :

$$\begin{aligned}\textstyle\sum^{<}_{i\in I} b_i.p_i \parallel \sum^{<}_{j\in J} b_j.p_j = \quad & \textstyle\sum^{<}_{i\in I} b_i.(p_i \parallel \sum^{<}_{j\in J} b_j.p_j) \\ & <+ \textstyle\sum^{<}_{j\in J} b_j.(\sum^{<}_{i\in I} b_i.p_i \parallel p_j) \\ & <+ \textstyle\sum^{<}_{i\in I, j\in J, a_i \mid a_j \neq \perp} (b_i \mid b_j).(p_i \parallel p_j)\end{aligned}$$

On peut remarquer que cette définition est quelque peu incomplète, car on va regarder des priorités entre des actions composées et des actions simples. Il faut donc étendre l'ordre de priorité de A à $A_{\mid}$. Cela se fera de la façon suivante.

Définition 24 *Extension de l'ordre de priorités*
L'ordre de priorités sur $A_{\mid}$ est obtenu par une extension naturelle de l'ordre sur $A,<$. On prendra pour cela le plus petit ordre sur $A_{\mid}$ au-dessus de $<$ tel que :

$$\forall a_1, a_2 \in A \;.\; a_1 <_\infty a_1 \mid a_2 \quad et \quad a_2 <_\infty a_1 \mid a_2$$
$$\forall a_1, a_2, a_3 \in A \;.\; a_1 <_k a_2 \Rightarrow a_1 <_k a_2 \mid a_3$$

Dès lors, du fait des propriétés d'associativités du choix avec priorités, l'opérateur de composition parallèle est associatif si la fonction de synchronisation est associative sur les actions temporisées (c'est-à-dire $b_1|(b_2|b_3) = (b_1|b_2)|b_3$). Cela ne dépend que de la synchronisation des gardes et échéances, puisque $|$ est associatif par définition sur A et que les fonctions de transfert se composent de façon associative.

De nombreuses façon d'effectuer la synchronisation existent. Parmi celles-ci, peu se révèlent intéressantes. Nous verrons au chapitre suivant trois synchronisations qui se distinguent des autres : ET, MIN et MAX.

3.3 Critique du modèle

Le modèle des SHEC, que nous venons de présenter, correspond à ce que nous attendions, à savoir :

- il est réactif temporellement, et cette propriété est préservée par composition parallèle.
- La composition parallèle garantit par construction les propriétés de progrès maximal et de conservation de l'activité.

Nous avons de plus une grande gamme d'opérateurs aussi bien pour le choix que la composition parallèle. Nous avons ainsi mis en évidence les possibilités qu'offre la temporisation.

Cependant, du fait de cette grande liberté qui nous est offerte, on peut être amené à spécifier des systèmes dont le comportement est tout à fait aléatoire en associant différentes compositions à des choix exotiques. Le fait que la composition parallèle soit définie à partir du choix avec priorités complique énormément les calculs dès lors que l'on veut utiliser un autre choix, et il devient difficile de savoir ce que l'on a spécifié. Dans de nombreux cas, on peut néanmoins se contenter de quelques compositions et du choix avec priorités. De plus, il est souvent possible de déterminer les échéances à partir des gardes d'une façon ou d'une autre. C'est pour cette raison que nous définirons un modèle simplifié, moins puissant, mais probablement plus facile à utiliser.

Chapitre 4

Un modèle simplifié : les ATTT

4.1 Pourquoi un autre modèle ?

Le modèle que nous venons d'étudier, s'il est satisfaisant du point de vue théorique, a quelques gros défauts du point de vue pratique. Comme nous l'avons vu, sa richesse en fait un outil difficile à exploiter. Un humain peut déjà se perdre dans la diversité des opérateurs de choix et de composition parallèle, mais si l'on a une idée de ce que l'on fait, on peut tirer profit de cette diversité. Par contre, lorsque l'on veut faire de la synthèse automatique, ce modèle est complètement inutilisable. C'est pourquoi nous nous contenterons d'un modèle simplifié, moins expressif, mais utilisable d'un point de vue pratique. Nous appellerons ce modèle simplifié les ATTT (Automates Temporisés à Transitions Typées). Ce modèle, s'il est moins riche, est suffisant dans bien des cas pratiques. De plus, il est facile d'y ajouter des extensions si besoin.

Par rapport au modèle des SHEC, nous nous baserons sur les automates et non plus simplement sur les systèmes de transitions. Ceci permettra une écriture plus compacte du système. Nous limiterons de plus plusieurs aspects du modèle général :

- On se placera dans le cas temporisé et non plus hybride, à savoir que les variables seront des réels servant à mesurer le temps. Pour cette raison, la fonction d'évolution $\triangleright$ sera $+$. On se limitera de plus aux fonctions de transfert qui soit ne changent pas les variables, soit les remettent a zéro.

- Les échéances ne seront plus définies comme des conditions indépendantes des gardes, mais comme le résultat de fonctions appliquées aux gardes. Nous obtiendrons ainsi des transitions temporisées typées, le type correspondant à l'urgence de la transition.

- Nous ne garderons que l'opérateur de choix avec priorités, indispensable pour la composition parallèle, et qui couvre le choix non déterministe. Les autres choix, qui peuvent être intéressants dans des cas précis, sont rarement nécessaires, et on a bien souvent avantage à calculer directement

les gardes que l'on désire plutôt que de calculer un opérateur parfois très complexe.

- Nous ne garderons que trois opérateurs de composition parallèle: ET, MIN et MAX. Nous verrons ces opérateurs en 4.4.1.

Regardons plus en détail chacun de ces aspects.

4.2 Transitions typées

Jusqu'à présent, nous avons spécifié les échéances indépendamment des gardes, or bien souvent nous désirons simplement spécifier le "degré d'urgence" d'une transition. C'est-à-dire qu'une fonction permettant de calculer l'échéance en fonction de la garde suffit.

Nous utiliserons trois fonctions (ou *types*) particulièrement intéressantes :

- les transitions paresseuses (en anglais *lazy*), sont celles qui ne bloquent jamais le temps. Ces transitions sont optionnelles : même s'il n'y a pas la concurrence d'autres actions, elle peuvent ne jamais avoir lieu. Cela peut, par exemple, représenter des perturbations du système, dont l'occurrence est incertaine. Nous utiliserons la fonction λ pour les représenter :

  $$\forall g \ . \ \lambda(g) = FAUX$$

  Ainsi les transitions paresseuses auront une échéance toujours fausse, ce qui correspond à la définition que nous avons donnée.

- les transitions retardables (en anglais *delayable*), sont les plus courantes. Ce sont des transitions qui peuvent ne pas être exécutées immédiatement, mais qui doivent être exécutées avant que leur garde ne soit dépassée (à part si une transition concurrente est exécutée avant). Cela représente toutes les transitions dont la date d'exécution est soumise à une certaine incertitude, comme nous le verrons plus loin. Nous utiliserons la fonction δ pour les représenter :

  $$\forall g \ . \ \delta(g) = g\downarrow$$

  L'échéance est alors le front descendant de la garde. La transition peut être exécutée dès que la garde est vraie mais doit être exécutée au plus tard lorsque l'échéance devient vraie, c'est-à-dire au dernier moment où la garde est vraie.

  Ce type n'est possible que si la garde est fermée à droite dans le sens du temps, sinon l'échéance ne serait pas incluse dans la garde.

- les transitions immédiates (en anglais *eager*), sont les transitions à exécuter aussi tôt que possible. Nous les représenterons par la fonction ϵ :

  $$\forall g \ . \ \epsilon(g) = g$$

  L'échéance étant égale à la garde, le temps est bloqué dès que la transition est possible. Ceci impose une exécution immédiate. On notera cependant

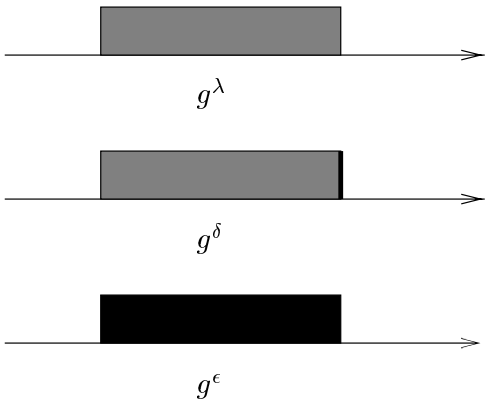


FIG. 4.1 – *Échéances engendrées par les types*

que, du fait de la définition que nous avons prise pour les choix, le temps est bloqué, mais une autre action peut être exécutée.

Nous représenterons les gardes typées avec la notation suivante : g^{type}. Ainsi l'action maximale aura pour garde typée $VRAI^{\epsilon}$.

Voyons sur des exemples ce que donnent ces trois types.

Exemple 9
Tout d'abord, la figure 4.1 nous montre les différentes échéances engendrées par les différents types. L'une des manières d'utiliser les types, mais il y en bien d'autres, est d'attribuer un type aux actions en fonction de la nature de l'action :

- une action du système sera immédiate.
- une action non contrainte de l'environnement sera paresseuse.
- une action de l'environnement nécessitant une réaction du système sera retardable.

La distinction est arbitraire, et la frontière entre les deux derniers cas est relativement floue. Néanmoins, pour essayer de fixer les idées on regardera ce modèle très simplifié de téléphone (figure 4.2). Les actions du téléphone lui-même seront immédiates, celles de l'utilisateur seront paresseuses sauf celles soumises à une contrainte de temps. Ainsi, on ne laissera qu'un certain délai pour numéroter. Ce modèle est extrêmement simplifié et omet certains aspects, aussi les gardes ne sont-elles pas données.

Pour conclure, nous remarquerons que nous ne perdons ici aucune généralité, puisque toute transition avec garde et échéance peut se décomposer en une transition immédiate et une transition paresseuse.

Propriété 24
Une transition de garde g et d'échéance c quelconques peut être représentée par l'union de la transition immédiate de garde c et de la transition paresseuse de garde $g \wedge \neg c$.

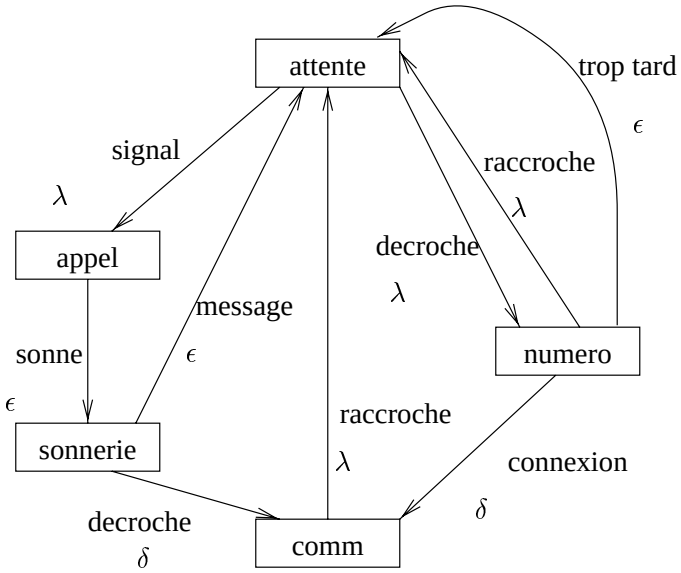


FIG. 4.2 – *Un téléphone simplifié*

Nous avons ajouté le type retardable, car il est souvent utile. Mais on pourra au besoin le considérer comme une macronotation puisqu'il est couvert par les deux autres types.

En conclusion, nous avons gardé trois types, qui sont les plus utilisés, et qui couvrent la totalité des cas. Cependant, si un système utilise abondamment un autre type, celui-ci pourra être défini comme une macronotation à partir des trois types de bases, ce qui permettra de simplifier la spécification. Dans certains cas, comme pour les transitions retardables, on pourra effectuer des compositions directement avec les types spécifiés (cf 4.4.2), mais dans la plupart des cas, on se contentera de les traduire en transitions paresseuses et immédiates.

4.3 Choix

Pour simplifier la spécification des composants élémentaires du système, nous n'utiliserons que l'opérateur de choix avec priorités. Plus exactement, nous n'utiliserons qu'un opérateur de choix. Nous définirons un ordre de priorités sur l'alphabet du système, qui sera utilisé pour tous les choix de tous les composants. Cette restriction, qui peut sembler sévère, est nécessaire pour pouvoir utiliser la composition parallèle simplement.

Dans le cas où l'on voudrait avoir des ordres différents suivant les composants, il faudra soit ajouter de nouvelles étiquettes (ce qui se réalise simplement, mais ne correspond pas forcément à ce que l'on veut), soit calculer à l'avance les gardes correspondant au résultat des priorités. Cette dernière solution a l'inconvénient que la composition parallèle peut alors donner des résultats très

différents de ce que l'on attendrait. Ceci est donc à éviter autant que possible, ce qui devrait généralement être le cas. En effet, on aura rarement plus de quelques étiquettes qui seront comparables, et le cas que nous venons d'évoquer ne devrait se présenter qu'avec une extrême rareté.

Une caractéristique importante du choix avec priorités est aussi qu'il est stable par rapport aux trois types utilisés.

Proposition 25
Si l'on réduit le choix avec priorités à un choix non déterministe, les transitions immédiates et paresseuses restent respectivement immédiates et paresseuses. De plus les transitions retardables sont transformées en transitions qui peuvent être décomposées en une transition paresseuse et une transition retardable.

Preuve 25
Par définition du choix avec priorités, il est évident que les transitions paresseuses restent paresseuses et que les transitions immédiates restent immédiates. Quant aux transitions retardables, on peut les voir comme des transitions paresseuses jusqu'à leur front descendant qui est immédiate. L'effet du choix avec priorités peut alors supprimer le front descendant, ce qui donne une transition paresseuse, ou non, ce qui donne une transition retardable. Une transition retardable est donc transformée en l'union de deux transitions, l'une paresseuse, l'autre retardable, définies suivants les endroits où le front descendant de la transition initiale est supprimé.
□

Cette preuve est insatisfaisante, dans le sens où elle ne permet pas de construire effectivement la décomposition d'une transition retardable. Néanmoins, elle assure que le calcul des gardes et échéances effectives d'un système (par réduction du choix avec priorités au choix non déterministe), est stable par rapport aux types utilisés. De plus on pourra utiliser le fait que le choix avec priorités préserve l'équivalence induite par scission d'une transition en deux autres. C'est-à-dire que nous appliquerons la décomposition de toute transition en une transition paresseuse et une transition immédiate en étant assurés que le choix avec priorités ne modifie pas le comportement du système.

4.4 Compositions parallèles

Nous allons voir trois modes de synchronisation. Ces trois modes ont été choisis pour deux raisons. La première est qu'ils couvrent une grande partie des interactions entre deux systèmes. La seconde est une propriété liée aux types : on peut typer les gardes obtenues (au besoin en scindant la synchronisation). De plus, nous apporterons une simplification supplémentaire qui consistera à définir un seul opérateur de composition parallèle, mais pouvant utiliser les trois modes de synchronisation. Ce mode sera choisi en fonction des transitions à synchroniser.

4.4.1 Modes de synchronisation

Les trois modes de synchronisation que nous avons gardés sont les suivants : ET, MAX et MIN.

Le mode ET correspond à la synchronisation traditionnelle, c'est-à-dire que la transition composée peut avoir lieu lorsque les deux transitions composantes peuvent avoir lieu. On obtient alors $g_1 | g_2 = g_1 \wedge g_2$. Ceci correspond à ce que l'on comprend habituellement par synchronisation : les composantes coopèrent pour donner une action composée. S'il y a désynchronisation (les composantes ne pouvant être effectuées simultanément), la synchronisation ne peut avoir lieu.

Le mode MAX correspond à une synchronisation avec attente : la première action pouvant s'effectuer attend la seconde pour synchroniser, et ce même si la deuxième action devient possible à un moment où la première ne l'est plus. Ceci est réalisé par : $g_1 | g_2 = (g_1 \wedge \diamondsuit\, g_2) \vee (\diamondsuit\, g_1 \wedge g_2)$. La garde composée est vraie lorsque l'une des gardes est vraie et l'autre l'a été dans le passé. Le nom de cette synchronisation vient du fait que si les deux gardes g_1 et g_2 sont des intervalles, alors la garde synchronisée est aussi un intervalle, dont la borne supérieure est le maximum des bornes supérieures des deux intervalles et la borne inférieure est le maximum de leur borne inférieure. Cela peut être simulé avec des compositions ET, tant que l'on ne cherche pas à composer à nouveau le système obtenu; dans le cas contraire cela ne peut être réalisé car la composition donnée par MAX est atomique tandis que la version faite avec ET comporte des états intermédiaires.

On remarquera de plus que MAX ne correspond effectivement à une synchronisation avec attente que si l'on peut contrôler la valeur des variables lorsque le système entre dans l'état de contrôle concerné. Supposons en effet que l'on veuille composer par MAX deux transitions dont les gardes respectives sont $g_1 = (x > 5)$ et $g_2 = (y = 3)$ dans le cas où x et y sont des horloges. Si lorsque le système entre dans l'état de contrôle on a $x = y = 6$, alors g_2 n'a jamais été vraie dans l'évolution du système, par contre $\diamondsuit\, g_2$ est vraie, donc la synchronisation est possible. Il faut bien voir que nous avons ici des propriétés locales à un état de contrôle et que les opérateurs modaux ne tiennent pas compte de l'évolution réelle du système, mais d'une évolution virtuelle dans le cas où toute l'évolution se ferait dans l'état de contrôle donné sans aucune transition discrète.

Le mode MIN correspond à une interruption : la première action pouvant s'effectuer interrompt la seconde et force la synchronisation. On réalise cela par $g_1 | g_2 = (g_1 \wedge \diamondsuit g_2) \vee (\diamondsuit g_1 \wedge g_2)$. La garde composée est vraie lorsque l'une des deux gardes est vraie et l'autre sera vraie dans l'avenir. Le nom MIN vient d'une propriété similaire à celle observée pour MAX.

La figure 4.3 nous montre les gardes composées pour chacun des trois modes dans le cas où $g_1 = (1 \leq x \leq 3)$ et $g_2 = (2 \leq y \leq 4)$.

Une propriété intéressante de ces trois modes de synchronisation est qu'ils

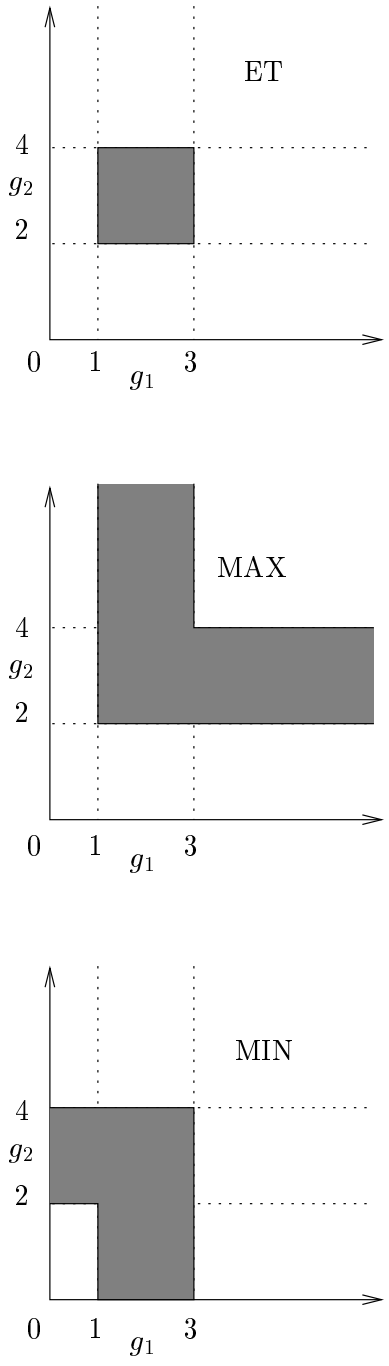


FIG. 4.3 – *Les trois modes de compositions*

sont associatifs.

Propriété 26
Les trois mode ET, MAX et MIN sont associatifs.

Preuve 26
Commençons par ET. Comme on a vu que l'on peut scinder toute transition en une transition immédiate et une transition paresseuse, on ne considérera que les types ϵ et λ. On utilisera de plus l'ordre sur les types : $\epsilon > \lambda$.

On a alors, pour les transitions typées $g_1^{\alpha_1}$ et $g_2^{\alpha_2}$:

$$ET(g_1^{\alpha_1}, g_2^{\alpha_2}) = (g_1 \wedge g_2)^{max(\alpha_1,\alpha_2)}$$

Pour trois gardes typées on a alors :

$$ET(\ ET(g_1^{\alpha_1}, g_2^{\alpha_2})\ , g_3^{\alpha_3}) = (g_1 \wedge g_2 \wedge g_3)^{max(\alpha_1,\alpha_2,\alpha_3)}$$

L'associativité en découle trivialement.

Passons maintenant aux types MAX et MIN. Les preuves étant identiques, on ne verra ici que la preuve pour MIN. De même que pour ET, on ne considérera que les types ϵ et λ. On a alors :

$$MIN(g_1^{\alpha_1}, g_2^{\alpha_2}) = (g_1 \wedge \Diamond g_2)^{\alpha_1} \vee (\Diamond g_1 \wedge g_2)^{\alpha_2}$$

Et en conséquence :

$$\begin{aligned} MIN(\quad & MIN(g_1^{\alpha_1}, g_2^{\alpha_2})\ , g_3^{\alpha_3}) = \\ & = MIN((g_1 \wedge \Diamond g_2)^{\alpha_1}, g_3^{\alpha_3}) \vee MIN((\Diamond g_1 \wedge g_2)^{\alpha_2}, g3^{\alpha_3}) \\ & = (g_1 \wedge \Diamond g_2 \wedge \Diamond g_3)^{\alpha_1} \vee (\Diamond(g_1 \wedge \Diamond g_2) \wedge g_3)^{\alpha_3} \vee \\ & \quad (\Diamond g_1 \wedge g_2 \wedge \Diamond g_3)^{\alpha_2} \vee (\Diamond(\Diamond g_1 \wedge g_2) \wedge g_3)^{\alpha_3} \\ & = (g_1 \wedge \Diamond g_2 \wedge \Diamond g_3)^{\alpha_1} \vee (g_2 \wedge \Diamond g_1 \wedge \Diamond g_3)^{\alpha_2} \vee \\ & \quad ((\Diamond(g_1 \wedge \Diamond g_2) \vee \Diamond(\Diamond g_1 \wedge g_2)) \wedge g_3)^{\alpha_3} \end{aligned}$$

Or on a $\Diamond(g_1 \wedge \Diamond g_2) \vee \Diamond(\Diamond g_1 \wedge g_2) = \Diamond(g_1) \wedge \Diamond(g_2)$. On obtient ainsi une expression symétrique en g_1, g_2 et g_3. L'associativité en découle.
□

4.4.2 À propos des échéances

De même que nous avons typé les transitions, nous allons chercher à typer les transitions composées. Mais un problème se pose alors. Nous exigeons $c_1 | c_2 = (g_1 | g_2) \wedge (c_1 \vee c_2)$. Comme le montre la figure 4.4, les transitions obtenues par synchronisation ne sont pas directement typables. Il faut donc les décomposer en plusieurs transitions qui elles seront typées. Nous allons voir ici comment effectuer cette décomposition dans le cas où les transitions sont soit immédiates soit paresseuses (ce qui recouvre donc tous les cas), et une simplification dans le cas où les deux transitions sont retardables.

Nous allons utiliser les propriétés suivantes.

Propriété 27
Pour α appartenant à $\{\epsilon, \lambda\}$, on a :

1. $mode(g_1^{\alpha}, g_2^{\alpha}) = (g_1 | g_2)^{\alpha}$

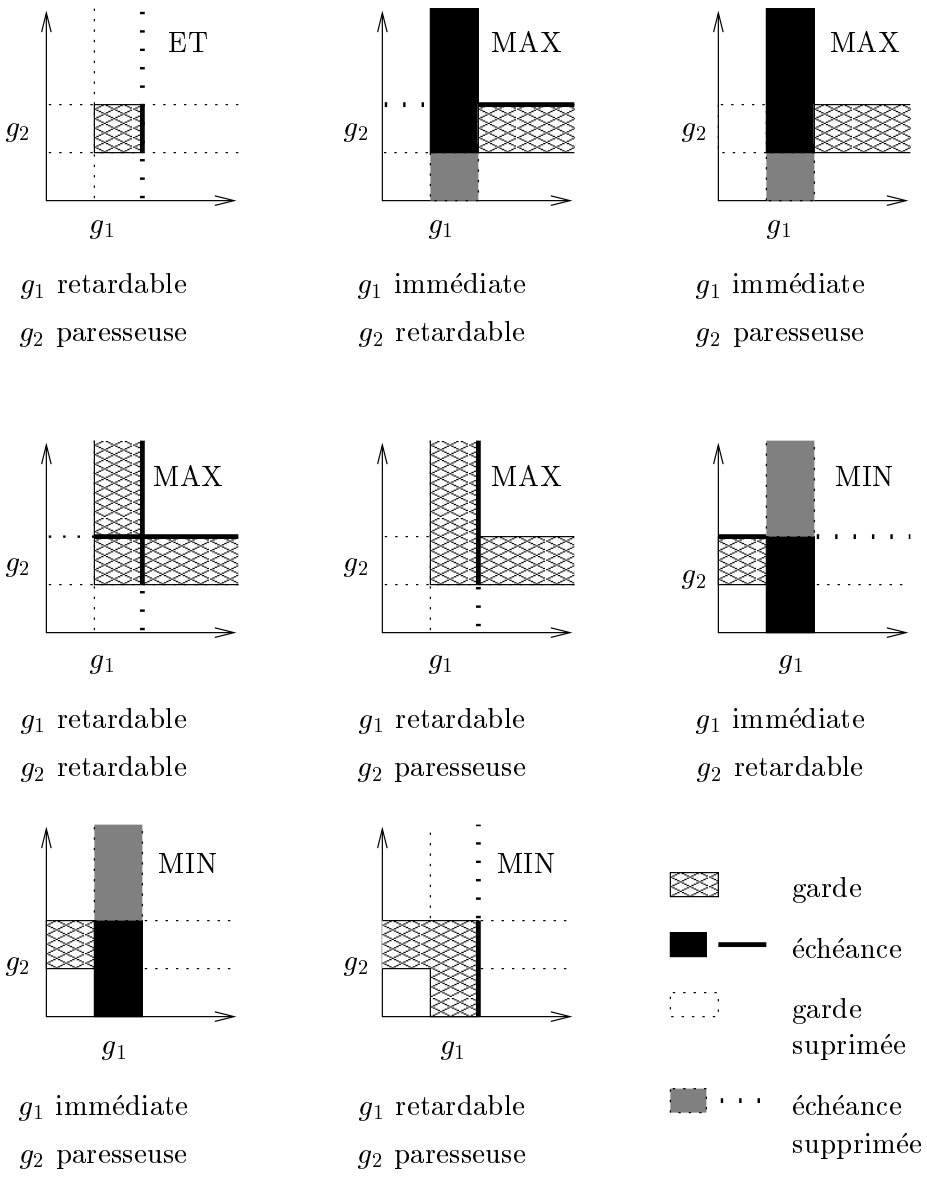


FIG. 4.4 – *Échéances après composition*

2. $mode(g_1^\epsilon, g_2^\lambda) = (g_1 \wedge m_2(g_2))^\epsilon \vee (m_1(g_1) \wedge g_2)^\lambda$

3. $mode(g_1^\lambda, g_2^\epsilon) = (g_1 \wedge m_2(g_2))^\lambda \vee (m_1(g_1) \wedge g_2)^\epsilon$

Preuve 27

1. *Comme* $g_1 | g_2 \Rightarrow g_1 \vee g_2$, *on a :*

$$\begin{array}{rl} mode(g_1^\epsilon, g_2^\epsilon) & = (g_1|g_2, (g_1|g_2) \wedge (g_1 \vee g_2)) \\ & = (g_1|g_2, g_1|g_2) \\ & = (g_1|g_2)^\epsilon \\ mode(g_1^\lambda, g_2^\lambda) & = (g_1|g_2, (g_1|g_2) \wedge (FAUX \vee FAUX)) \\ & = (g_1|g_2, FAUX) \\ & = (g_1|g_2)^\lambda \end{array}$$

2. *Comme* $g_i \Rightarrow m_i(g_i)$, *on a :*

$$\begin{array}{rl} mode(g_1^\epsilon, g_2^\lambda) & = (g_1|g_2, (g_1|g_2) \wedge (g_1 \vee FAUX)) \\ & = ((g_1 \wedge m_2(g_2)) \vee (m_1(g_1) \wedge g_2), g_1 \wedge m_2(g_2)) \\ & = (g_1 \wedge m_2(g_2))^\epsilon \vee (m_1(g_1) \wedge g_2)^\lambda \end{array}$$

3. *idem*

□

Ces trois propriétés nous donnent la décomposition des transitions lorsque les seuls types utilisés sont ϵ et λ. Ceci est suffisant puisque nous avons vu que l'on peut décomposer toute transition en une transition immédiate et une paresseuse. Dans le cas où les deux transitions sont retardables, nous pouvons nous passer de cette décomposition, comme le montre la propriété suivante.

Propriété 28
On a les égalités suivantes :

$$\begin{array}{rl} ET(g_1^\delta, g_2^\delta) & = (g_1 \wedge g_2)^\delta \\ MAX(g_1^\delta, g_2^\delta) & = (g_1 \wedge \diamondsuit\, g_2)^\delta \vee (\diamondsuit\, g_1 \wedge g_2)^\delta \\ MIN(g_1^\delta, g_2^\delta) & = (g_1 \wedge \Diamond g_2)^\delta \vee (\Diamond g_1 \wedge g_2)^\delta \end{array}$$

Preuve 28

Pour les gardes fermées à gauche, on a $(g_1 \wedge g_2)\downarrow = (g_1\downarrow \wedge g_2) \vee (g_1 \wedge g_2\downarrow)$. *On a alors immédiatement :*

$$\begin{array}{rl} ET(g_1^\delta, g_2^\delta) & = (g_1 \wedge g_2, (g_1\downarrow \wedge g_2) \vee (g_1 \wedge g_2\downarrow)) \\ & = (g_1 \wedge g_2)^\delta \end{array}$$

De plus, si g est fermée à gauche, il en est de même pour $\Diamond g$ et $\diamondsuit g$. Ce qui nous donne:

$$\begin{array}{ll}
(g_1 \wedge \Diamond g_2)^{\downarrow} & \vee(\Diamond g_1 \wedge g_1)^{\downarrow} \\
& = (g_1 \downarrow \wedge \Diamond g_2) \vee (g_1 \wedge (\Diamond g_2)\downarrow) \vee ((\Diamond g_1)\downarrow \wedge g_2) \vee (\Diamond g_1 \wedge g_2 \downarrow) \\
& = (g_1 \downarrow \wedge \Diamond g_2) \vee (\Diamond g_1 \wedge g_2 \downarrow) \\
(g_1 \wedge \diamondsuit g_2)^{\downarrow} & \vee(\diamondsuit g_1 \wedge g_1)^{\downarrow} \\
& = (g_1 \downarrow \wedge \diamondsuit g_2) \vee (g_1 \wedge (\diamondsuit g_2)\downarrow) \vee ((\diamondsuit g_1)\downarrow \wedge g_2) \vee (\diamondsuit g_1 \wedge g_2 \downarrow) \\
& = (g_1 \downarrow \wedge \diamondsuit g_2) \vee (\diamondsuit g_1 \wedge g_2 \downarrow)
\end{array}$$

On en déduit:

$$\begin{array}{ll}
MAX(g_1^{\delta}, g_2^{\delta}) & = ((g_1 \wedge \diamondsuit g_2) \vee (\diamondsuit g_1 \wedge g_2), (g_1 \downarrow \wedge \diamondsuit g_2) \vee (\diamondsuit g_1 \wedge g_2 \downarrow)) \\
& = (g_1 \wedge \diamondsuit g_2)^{\delta} \vee (\diamondsuit g_1 \wedge g_2)^{\delta} \\
MIN(g_1^{\delta}, g_2^{\delta}) & = ((g_1 \wedge \Diamond g_2) \vee (\Diamond g_1 \wedge g_2),, (g_1 \downarrow \wedge \Diamond g_2) \vee (\Diamond g_1 \wedge g_2 \downarrow)) \\
& = (g_1 \wedge \Diamond g_2)^{\delta} \vee (\Diamond g_1 \wedge g_2)^{\delta}
\end{array}$$

□

4.4.3 Réalisation de la composition

Il reste à réaliser la composition parallèle. Nous le ferons en spécifiant le mode de composition sur les étiquettes. Nous avons vu que chacun des modes est associatif. Pour assurer l'associativité de la composition parallèle on définira un mode pour les étiquettes, et seules les étiquettes de même mode pourront synchroniser. La composition parallèle est alors donnée par:

- l'opérateur $|$ qui donne l'étiquette $a_1|a_2$ de la synchronisation de deux transitions d'étiquettes a_1 et a_2 quand cette synchronisation existe, et qui indique que la synchronisation est impossible (par $\perp$) dans le cas contraire.

- le mode $mode(a)$ de toute action de A. Les actions de modes différents ne pourront synchroniser. On étendra de façon naturelle le mode aux actions composées: si a_1 et a_2 ont le même mode, alors $a_1|a_2$ aura aussi ce mode. On choisira le mode parmi les trois disponibles: ET, MAX ou MIN.

Les trois modes utilisés donnent les gardes et les échéances de la synchronisation, et nous avons vu que cette synchronisation peut être décomposée aisément en transitions typées. On obtient ainsi des transitions dont les étiquettes, les gardes et les types sont connus. Les fonctions de transfert se composant de façon unique, la synchronisation est entièrement spécifiée. Et nous avons vu que cela suffit pour spécifier la composition parallèle grâce à la formule d'expansion:

$$\begin{array}{ll}
\sum^{<}_{i\in I} b_i.p_i \parallel \sum^{<}_{j\in J} b_j.p_j = & \sum^{<}_{i\in I} b_i.(p_i \parallel \sum^{<}_{j\in J} b_j.p_j) \\
& <+ \sum^{<}_{j\in J} b_j.(\sum^{<}_{i\in I} b_i.p_i \parallel p_j) \\
& <+ \sum^{<}_{i\in I, j\in J, a_i|a_j\neq\perp} (b_i|b_j).(p_i \parallel p_j)
\end{array}$$

4.5 Synthèse

Maintenant que nous avons donné tous ces aspects techniques, voyons comment l'on peut définir un ATTT, et récapitulons les propriétés qui en découlent.

Définition 25 *ATTT*

Un ATTT est un n-uplet $(S, A, X, \triangleright, s_0, <)$ *où :*

- S *est un ensemble fini de places, appelées* états de contrôle.
- A *est un alphabet fini.*
- X *est un ensemble de variables réelles. On notera respectivement* V, G *et* F *l'ensemble des valuations de* X, *l'ensemble des prédicats sur* V *et l'ensemble des fonctions de* V *dans* V.
- $\rightarrow$ *est un sous-ensemble fini de* $S \times (A \times G \times \{\epsilon, \delta, \lambda\} \times F) \times S$ *appelé ensemble des transitions. Pour une transition* $(s, (a, g, \alpha, f), s')$ *appartenant à* $\rightarrow$, *on dit que*
  - s *est l'état source de la transition.*
  - a *est l'étiquette de la transition.*
  - g *est la garde de la transition.*
  - α *est le type de la transition.*
  - f *est la fonction de transfert de la transition.*
  - s' *est l'état d'arrivée de la transition.*

  On utilisera aussi la notation $s \overset{a,g^{\alpha},f}{\rightarrow} s'$ *pour représenter une transition. On exigera par ailleurs que* g *soit fermée à gauche dans le sens du temps et ait un nombre fini de composantes connexes. Si le type est* δ, *on exigera aussi que* g *soit fermée à droite. De plus la fonction de transfert* f *ne pourra affecter les variables que de deux manières : où bien elles resteront inchangées, ou bien elles seront remises à zéro.*
- s_0 *appartient à* S *et est appelé l'état de départ du système.*
- $<$ *est un ordre de priorités sur* A.

Le modèle sémantique de ce système est un système de transitions calculé en deux étapes :

- on commence par calculer le système temporisé avec choix non déterministe équivalent au ATTT considéré (voir 3.1.2).
- le système de transitions est ensuite calculé par les règles d'induction suivantes :

$$\frac{s \overset{a,g,\alpha,f}{\rightarrow} s' \quad \text{et} \quad g(v)}{(s, v) \overset{a}{\rightarrow} (s', f(v))} \qquad \frac{\forall t' \in [0, t) \;.\; \neg \bigvee_{s \overset{a,g,\alpha,f}{\rightarrow} s'} \alpha(g)(v \triangleright t')}{(s, v) \overset{t}{\rightarrow} (s, v \triangleright t)}$$

Les transitions $(s, v) \overset{a}{\rightarrow} (s', f(v))$ sont appelées les actions et les transitions $(s, v) \overset{t}{\rightarrow} (s, v \triangleright t)$ sont appelées des progressions du temps, la notation $v \triangleright t$ représentant la valuation où toutes les horloges ont été augmentées de la durée t.

Les ATTT vérifient la propriété de réactivité temporelle : ou bien une transition est possible, ou bien on peut laisser passer le temps.

Cette propriété n'implique pas la propriété dite "non-Zénon", qui assure que le système ne peut effectuer une séquence infinie d'actions en un temps fini.

Pour construire des systèmes complexes, nous disposons de l'opérateur de composition parallèle qui à partir de deux ATTT en construit un troisième. Cet opérateur n'est défini que pour deux systèmes $T = (S, A, X, \triangleright, \rightarrow, s_0, >)$ et $T' = (S', A', X', \triangleright', \rightarrow', s'_0, >')$ qui satisfont :

- $A = A'$
- $X \cap X' = \emptyset$
- $> = >'$

On définit alors sur A deux fonctions :

- le mode $mode : A \rightarrow \{ET, MAX, MIN\}$.
- la fonction de composition $| : A \times A \rightarrow B \cup \{\perp\}$. B est un alphabet dont l'intersection avec A est vide. $\perp$ est une étiquette spéciale n'appartenant ni à A, ni à B et qui représente l'impossibilité de la synchronisation. Deux étiquettes de mode différent ne synchroniseront pas.

Ces deux fonctions déterminent la composition parallèle. $T \parallel T'$ est alors le système $(S'', A'', X'', \triangleright'', \rightarrow'', s''_0, >'')$ défini par :

- $S'' = S \times S'$ ($\times$ est le produit cartésien). On notera $s \parallel s'$ le couple (s, s').
- $A'' = A \cup \{a_1 | a_2 \ / \ a_1 \in A, a_2 \in A\}$
- $X'' = X \cup X'$. On notera (v, v') une valuation sur X'', où v et v' sont respectivement une valuation sur X et X' (définies de façon non ambiguë puisque X et X' sont séparés).
- $\triangleright'' = \triangleright' = \triangleright$ par définition des ATTT.
- $\rightarrow''=\rightarrow \cup \rightarrow' \cup \rightarrow_c$. $\rightarrow_c$ est l'ensemble des synchronisations et est défini par la règle :

$$\frac{s \overset{a, g^\alpha, f}{\Leftrightarrow} s_1 \ , \ s' \overset{a', g'^{\alpha'}, f'}{\Leftrightarrow'} s'_1 \ , \ a|a' \neq \perp}{s \parallel s' \overset{a|a', mode(a)(g^\alpha, g'^{\alpha'}), f|f'}{\Leftrightarrow_c} s_1 \parallel s'_1}$$

  la composition des fonctions de transfert étant définie simplement par : $f|f'(v, v') = (f(v), f'(v'))$.
- $s''_0 = s_0 \parallel s'_0$
- $>''$ est l'ordre minimal induit sur A'' par les règles :
  - $a_1 \in A$ et $a_2 \in A$ impliquent $a_1 <_k a_2 \Leftrightarrow a_1 <''_k a_2$
  - $a_1 \in A$ implique $\forall a_2 \in A \ . \ a_1 <''_\infty a_1|a_2$
  - $a_1 \in A$, $a_2 \in A$ et $a_1 <_k a_2$ impliquent $\forall a_3 \in A \ . \ a_1|a_3 <''_k a_2|a_3$

4.6 Présentation sous forme d'algèbre de processus

Nous allons présenter ici une esquisse de formalisation des ATTT sous forme d'algèbre temporisée.

Nous ne travaillerons pour cela que sur des processus ayant un alphabet A, un ensemble de variables X et un ordre de priorités $<$ communs. Nous représenterons les ATTT par des termes de l'algèbre :

$$p ::= Nil \mid b.p \mid p <+ p \mid p \parallel p \mid Z \mid RecZ.p$$

avec b appartenant à l'ensemble B des actions de A temporisées sur X et Z appartenant à l'ensemble des variables de processus χ.

On interprétera les différents opérateurs de la façon suivante :

- Nil est le processus oisif: il ne peut que laisser passer le temps. Il correspond donc à un état puits.
- $b.p$ est le préfixage du processus p par l'action b.
- l'opérateur $<+$ est l'opérateur de choix avec priorités des ATTT.
- $RecZ.p$ est l'opérateur de récursion, permettant de représenter les cycles dans les ATTT. On ne considérera que les termes bien formés, c'est-à-dire sans variable libre, et où toute variable liée apparaît après un opérateur de préfixage. On interdira aussi les compositions parallèles à l'intérieur des récursions.

Ces opérateurs vérifient alors les propriétés suivantes, par définition :

$$\begin{array}{ll} Nil <+ p = p & Nil \parallel p = p \\ b_1.p_1 \parallel b_2.p_2 = & (b_1.(p_1 \parallel b_2.p_2) <+ b_2.(b_1.p_1 \parallel p_2)) \\ & <+ (b_1 | b_2).(p_1 \parallel p_2) \\ RecZ.p = & p[p/Z] \end{array}$$

De plus $<+$ étant associatif, on pourra utiliser la notation $\sum^{<}$. Les formules précédentes permettent alors décrire tout terme sous la forme

$$p = \sum_{i \in I}^{<} b_i.p_i$$

Si I est vide, on obtient Nil.

Nous ne donnerons pas ici la construction des termes de l'algèbre à partir des ATTT. Cette construction est tout à fait similaire à celle faite sur les automates classiques. Nous nous intéresserons, en revanche, à l'équivalence induite sur les processus par une équivalence sur les actions.

4.6.1 Équivalence sur les actions temporisées

L'un des problèmes posés par la temporisation est la présence d'actions dont la garde est fausse. Comme ces actions ne peuvent être effectuées, nous les considérerons toutes comme équivalentes. Nous utiliserons alors la notation CUT qui représentera une action de garde fausse. Nous allons voir que cette notation est justifiée, car l'étiquette est sans importance.

Remarquons que pour les trois modes de composition ET, MAX et MIN, si la garde de l'une des actions est fausse alors la garde de la synchronisation est fausse :

$$\forall b \in B \ . \ b|CUT = CUT$$

Attention, cette notation fait disparaître le fait que les étiquettes sont différentes.

4.6.2 Équivalence sur les processus

D'après ce qui précède, il est intuitif de considérer l'équivalence $\equiv$ sur les processus définie par :

$$\begin{array}{lcl} CUT.p & \equiv & Nil \\ p_1 \equiv p_2 & \Rightarrow & b.p_1 \equiv b.p_2 \\ p_1 \equiv p_2 & \Rightarrow & p_1 <+ p' \equiv p_2 <+ p' \\ p_1 \equiv p_2 & \Rightarrow & p_1 \parallel p' \equiv p_2 \parallel p' \\ p_1 \equiv p_2 & \Rightarrow & RecZ.p_1 \equiv RecZ.p_2 \end{array}$$

On remarquera que si p_1 et p_2 sont équivalents par $\equiv$, alors p_1 et p_2 ne diffèrent que par la présence éventuelle d'actions temporisées de gardes fausses. Intuitivement, p_1 et p_2 se comportent donc de la même manière : c'est le sens du résultat suivant.

Théorème 29
Si deux processus p_1 et p_2 sont équivalents par $\equiv$ alors pour toute valuation v, (p_1, v) et (p_2, v) sont fortement bisimilaires.

Preuve 29
On va procéder par induction sur la preuve que p_1 et p_2 sont équivalents par $\equiv$:

- *Si $p_1 = Cut.p$ et $p_2 = Nil$ (ou l'inverse), ni (p_1, v) ni (p_2, v) ne peuvent effectuer d'actions, et tous deux peuvent laisser passer le temps indéfiniment. Ils sont donc bisimilaires.*

- *Si $p_1 = (a, g, c, f).p'_1$ et $p_2 = (a, g, c, f).p'_2$: si v satisfait g alors on a les transitions $(p_1, v) \xrightarrow{a} (p'_1, f(v))$ et $(p_2, v) \xrightarrow{a} (p'_2, f(v))$ avec $p'_1 \equiv p'_2$, sinon ni (p_1, v), ni (p_2, v) ne peuvent effectuer d'action discrète. Les comportements temporels sont liés de même.*

- *Si $p_1 = p'_1 <+ p$ et $p_2 = p'_2 <+ p$: p_1 et p_2 ne diffèrent que par des éventuelles transitions de garde vide, or ces transitions ne modifient pas les autres transitions et de plus elles ne peuvent jamais être prises. Ceci*

implique que si (p_1, v) peut laisser passer le temps par t, il en est de même pour (p_2, v), et l'inverse est vrai également. Si (p_1, v) peut effectuer une action discrète, il s'agit soit d'une action que peut effectuer (p, v), et alors (p_2, v) peut l'effectuer aussi (les restrictions sur les actions de p sont les mêmes dans $p'_1 <+p$ et $p'_2 <+p$), soit d'une action que (p'_1, v) peut effectuer, alors (p'_2, v) peut l'effectuer et (p_2, v) peut aussi l'effectuer, car p n'empêchant pas l'action dans (p_1, v), il ne peut l'empêcher dans (p_2, v). Dans tous les cas, les états atteints après transition sont équivalents.

Ceci se voit bien sur le processus $CUT.p <+b.p$: CUT ne modifiant pas le comportement de b et ne pouvant que laisser passer le temps, $CUT.p <b.p$ se comporte comme $b.p$, c'est-à-dire $Nil <b.p$.

- *Si $p_1 = p'_1 \parallel p$ et $p_2 = p'_2 \parallel p$: en fait, vue la règle d'expansion définissant $\parallel$, il suffit de vérifier que $CUT.p_1 \parallel p_2 \equiv Nil \parallel p_2 = p_2$. On a*

$$\begin{aligned} CUT.p_1 \parallel \textstyle\sum_{i\in I} {<} b_i.q_i = \quad & CUT.(p_1 \parallel \textstyle\sum_{i\in I} {<} b_i.q_i) \\ & {<}\textstyle\sum_{i\in I} {<} b_i(CUT.p_1 \parallel q_i) \\ & {<}\textstyle\sum_{i\in I} {<} (CUT \upharpoonright b_i).(p_1 \parallel q_i) \end{aligned}$$

Or $CUT \upharpoonright b_i = CUT$, et $CUT.p <+p' = p'$ donc

$$CUT.p_1 \parallel \sum_{i\in I}{<} b_i.q_i = \sum_{i\in I}{<} b_i(CUT.p_1 \parallel q_i)$$

Ceci prouve que $CUT.p_1 \parallel \sum_{i\in I} {<} b_i.q_i$ et $\sum_{i\in I} {<} b_i.q_i$ sont bisimilaires.

- *Si $p_1 = RecZ.p'_1$ et $p_2 = RecZ.p'_2$: en utilisant le fait que $RecZ.p = p[RecZ.p/Z]$, le résultat est évident.*

Ceci est plus un squelette de preuve qu'une preuve complète, néanmoins on pourra compléter les points qui restent flous en regardant la sémantique des opérateurs $<+$ et $\parallel$.
□

On aurait pu aller un peu plus loin, et regarder l'équivalence qui semble naturelle :

$$(a, g_1 \vee g_2, c_1 \vee c_2, f).p \equiv (a, g_1, c_1, f).p <+(a, g_2, c_2, f).p$$

Cependant montrer un résultat de bisimilitude sur cette équivalence nécessiterait de donner les règles d'inférence permettant de construire le système de transition correspondant à l'algèbre de processus que nous étudions. Ces règles étant quelque peu complexes (à cause de la sémantique du choix avec priorités), il devient alors difficile de les utiliser dans une preuve. Si tous les espoirs sont permis quant à l'obtention de cette preuve, cela n'a pu être concrétisé, faute de temps.

Chapitre 5

Conclusions et perspectives

5.1 Objectifs

Nous avons cherché au cours de cette thèse à voir comment obtenir par construction des systèmes corrects. Cette exigence provient du fait que la vérification des systèmes complexes est extrêmement difficile, voire impossible. On cherchera donc à éliminer certaines sources d'erreur à la conception même du modèle. Le problème que nous avons abordé en priorité est celui de la composition parallèle des systèmes temporisés, qui peut engendrer des blocages si l'on ne fait pas attention.

La composition usuelle sur les systèmes temporisés est une extension de la composition parallèle sur les processus communicants. Or les systèmes temporisés ont une variable commune, le temps, même si cette variable n'apparaît pas directement dans la description des systèmes. Ces systèmes ayant des espaces d'états partagés ne sont donc pas des systèmes communicants au sens propre du terme et il est souhaitable d'adapter les opérateurs de compositions pour tenir compte de ce fait.

5.2 Résultats

Nous avons commencé par une étude préliminaire sur la façon de modéliser le temps et l'urgence des actions dans les systèmes temporisés. Cette étude nous a apporté un cadre dans lequel les liaisons entre spécification des contraintes temporelles (par les échéances) et utilisation de ces contraintes (par les fonctions de délais possibles) sont facilement formalisables. Nous avons vu que si nous nous restreignons à des échéances de forme modale, alors le calcul des FDP correspondantes peut être fait de façon relativement simple.

Tenant compte de ce résultat, nous avons développé un modèle de systèmes hybrides, les SHEC, qui par construction satisfont deux propriétés intéressantes :

- la réactivité temporelle : il s'agit en fait de l'absence de blocage : si le temps est arrêté, alors une action est possible. Comme cette propriété est une propriété locale, elle ne garantit pas que le temps pourra toujours

progresser (propriété dite de Zénon). Par contre, elle assure que si le temps est bloqué c'est qu'on a spécifié une infinité d'actions en un temps fini.

- le progrès maximal : cette propriété concerne les systèmes obtenus par composition parallèle, et dans lesquels on a concurrence entre des synchronisations et des transitions d'entrelacement permettant au système de transitions de progresser si la synchronisation est impossible. Le progrès maximal est la propriété que les synchronisations sont alors prioritaires sur les autres transitions. Contrairement aux autres solutions proposées jusqu'ici, nous n'avons pas ajouté d'opérateur de restriction, mais nous avons défini un opérateur de composition parallèle qui respecte cette propriété.

Pour réaliser ce modèle, nous avons apporté deux innovations :

- nous avons relié le comportement temporel des systèmes à leur comportement discret en imposant que les échéances devaient être incluses dans les gardes. Au-delà du détail technique il faut voir ici une volonté de satisfaire par construction des propriétés temporelles sur les systèmes. Ceci ne peut être réalisé si l'on spécifie indépendamment comportement temporel et comportement discret.

- nous avons montré qu'il existe de nombreuses façons de temporiser les opérateurs de composition discrets (opérateur de choix non déterministe et de composition parallèle). Évidemment parmi la grande variété d'opérateurs que nous pouvons construire, un grand nombre n'ont aucun intérêt. Néanmoins, on remarquera tout particulièrement les opérateurs de choix avec priorités et les compositions parallèles utilisant les synchronisations ET, MAX et MIN.

Enfin, nous avons montré un modèle simplifié, les ATTT. Ce modèle ne conserve que les opérateurs les plus intéressants. En restreignant les choix possibles, nous gagnons en facilité d'utilisation. Par ailleurs, on a poussé plus loin la formalisation des ATTT, et si ceci n'est qu'esquissé dans le présent ouvrage, une formalisation plus complète sous forme d'algèbre de processus et d'algèbre d'actions est en cours de réalisation.

5.3 Perspectives

Par la structure même de ce document, nous voyons que de nombreuses spécialisations du modèle des SHEC sont possibles. Nous n'en avons présenté qu'une seule, mais on pourra au cas par cas étudier des modèles plus adaptés à certaines situations et faisant intervenir d'autres opérateurs. Mais avant de chercher ces variantes, on s'attachera à valider le modèle originel par la pratique. Si plusieurs applications ont été réalisées (cf [BST97, Goe98, Alt98]), elles sont encore insuffisantes.

Même si l'on a un champ très vaste concernant les possibilités de formalismes utilisant les notions que nous avons présentées, nous sommes limités par

certaines contraintes qui pourraient éventuellement être assouplies. Ainsi une étude plus complète dans le cas où l'on autorise les échéances ouvertes à gauche permettrait d'augmenter sensiblement la puissance de notre modèle. Ceci nécessitera cependant de régler une foultitude de détails techniques.

Mais l'une des voies les plus prometteuses est d'appliquer les techniques que nous avons décrites à d'autres systèmes continus que les systèmes temporisés. En effet, partant de problèmes de temporisation, nous avons obtenu un opérateur de choix avec priorités utilisant des modalités temporelles. Il est tout à fait envisageable d'adapter les résultats obtenus au cas où l'on voudrait travailler sur des problèmes de coût (quel que soit le sens exact de ce coût), pour avoir des priorités liées à ce coût et donc adapter le modèle au calcul d'optimisations.

Bibliographie

[AD90] R. Alur and D. Dill. Automata for modeling real-time systems. In *17th ICALP*, LNCS 443, 1990.

[Alt98] K. Altisen. Génération automatique d'ordonnancements pour systèmes temporisés. Technical report, Mémoire de DEA, Ensimag, Grenoble, 1998.

[BST97] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In *International Symposium: Compositionality - The Significant Difference*, Malente (Holstein, Germany), September 1997. Lecture Notes in Computer Science 1536, Springer Verlag.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: an unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 14th Symp. on Principles of Programming Languages*, January 1977.

[CC92] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. Rapport technique, École Polytechnique, June 1992.

[Goe98] G. Goessler. Modélisation et contrôle des systèmes multimedia. Technical report, Mémoire de DEA, Ensimag, Grenoble, 1998.

[LGS+95] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1):11–44, January 1995.

[LL95] G. Leduc L. Léonard. An extended lotos for the design of real-time systems. In *workshop DARTS'95*, Bruxelles, Belgium, November 1995.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[MMP92] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Proc. REX Workshop "Real-Time: Theory in Practice"*, pages 447–484. LNCS 600, Springer-Verlag, 1992.

[MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1991.

[NRSV90] X. Nicollin, J.-L. Richier, J. Sifakis, and J. Voiron. ATP: an Algebra for Timed Processes. In *Proc. IFIP TC 2 Working Conference on Programming Concepts and Methods*, Israel, April 1990.

[NSY92] X. Nicollin, J. Sifakis, and S. Yovine. Compiling real-time specifications into extended automata. *IEEE TSE Special Issue on Real-Time Systems*, 18(9):794–804, September 1992.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th Symp. on Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977.

[PV94] A. Puri and P. Varaiya. Decidability of hybrid systems with rectangular differential inclusions. In *CAV'94*, LNCS 818, 1994.

[Ram74] C. Ramchandani. Analysis of asynchronous concurrent systems by petri nets. Technical Report MAC TR-120, MIT, 1974.

[SY96] J. Sifakis and S. Yovine. Compositional specification of timed systems. In *13th Annual Symposium on Theoretical Aspects of Computer Science, STACS'96*, pages 347–359, Grenoble, France, February 1996. Lecture Notes in Computer Science 1046, Spinger-Verlag.

[VW86] M. Vardi and P. Wolper. An automata-theoretic appoach to automatic program verification. In *proceedings of the IEEE symposium on logic in computer science*, pages 332 – 344, 1986.