Modèles composables et concurrents pour le temps-réel

Franck Pommereau

Université Paris 12 – Val-de-Marne U.F.R. de sciences

Numéro attribué par la bibliothèque : .....

Thèse

pour obtenir le grade de
Docteur de l'université Paris 12
discipline : informatique
présentée et soutenue publiquement par

Franck Pommereau

le 10 juin 2002

Modèles composables et concurrents pour le temps-réel

Directeur de thèse : Elisabeth Pelz
Co-directeur : Hanna Klaudel

Jury :	Gilles Bernot	Président
	Raymond Devillers	
	Paul Gastin	Rapporteur
	Hanna Klaudel	
	Elisabeth Pelz	
	Alex Yakovlev	Rapporteur

« *Les instants hors des choses ne sont rien* [...] *ils ne consistent que dans leur ordre successif.* »
— G. W. Leibniz (*Troisième écrit*)

Remerciements

En premier lieu, je veux remercier de tout mon cœur ma directrice, Elisabeth Pelz, qui a toujours été très attentionnée et qui m'a accueilli dans son équipe, m'y a installé comme dans un nid douillet et s'est toujours dépensée sans compter pour me donner les meilleures conditions de travail.

J'ai eu la chance d'avoir une deuxième directrice, Hanna Klaudel ; je suis en train de réaliser combien une thèse est quelque chose d'unique et elle y a grandement contribué. Ces trois années de recherches acharnées resteront toujours indissociables de sa présence attentive et de son intérêt actif pour mon travail.

Je remercie aussi vivement Paul Gastin et Alex Yakovlev qui me font l'honneur de rapporter ce mémoire. J'espère qu'ils prendront autant de plaisir à le lire que j'en ai eu à l'écrire. Je veux également exprimer toute ma gratitude à Gilles Bernot pour avoir accepté de présider le jury malgré son emploi du temps surchargé. Raymond Devillers m'a beaucoup appris durant toute cette thèse qu'il a suivie de près malgré la distance. Il me fait en plus le plaisir de participer au jury et je tiens à lui transmettre un grand merci pour tout cela.

Merci à Maciej Koutny qui a toujours répondu présent face à mes nombreuses sollicitations et m'a beaucoup transmis lors de ces échanges.

Je remercie chaleureusement Jacques Guignot qui m'a fait l'amitié et le plaisir d'accepter de relire ce mémoire. Merci aussi à Cécile Bui Thanh pour sa relecture en un temps record ; j'espère qu'elle me demandera de lui rendre le même service pour sa propre thèse.

Je ne voudrais pas oublier de remercier Gernot Richter, dont l'aide m'a été très précieuse au tout début de ce travail, et Guy Vidal-Naquet, dont l'intérêt précoce pour mon travail a beaucoup contribué à ma motivation.

Je tiens à saluer mes parents qui, d'une manière ou d'une autre, ont permis que j'en sois à écrire ces lignes aujourd'hui.

Merci à Claire pour avoir accepté les « clapotements » de mon clavier et pour sa patience durant la rédaction de ce mémoire.

Cette thèse ne serait pas la même sans la multitude de contributions discrètes, parfois anonymes, de tous ceux que je ne saurais citer ici. Je les remercie collectivement.

Sommaire

1 Introduction

L'informatique traite de plus en plus de questions complexes. Les systèmes dits *temps-réel*, qui vont nous intéresser dans cette thèse, possèdent des caractéristiques particulières qui amènent souvent à des problèmes difficiles à résoudre.

Pour illustrer notre propos, nous considérons l'exemple d'un avion de ligne moderne dans lequel la plupart des éléments sont contrôlés par informatique (l'avion est truffé de capteurs contrôlés par autant de logiciels, et il y a bien longtemps que le « manche à balai » n'est plus qu'un capteur parmi d'autres, très semblable aux *joysticks* utilisés par les amateurs de jeux vidéos). C'est aussi le cas de la majorité des manœuvres dans lesquelles l'humain n'a plus que le rôle d'assistant du logiciel (on pense bien sûr au pilote automatique, mais c'est aussi vrai pour la procédure d'atterrissage). Notons d'ailleurs que, si l'on parle volontiers de « systèmes électroniques », on oublie facilement que l'électronique n'est souvent que le support de l'exécution de logiciels.

Les systèmes temps-réel se rencontrent plus fréquemment à mesure que l'informatique étend ses domaines d'application. Ce que nous entendons par « système » est assez vaste. Un avion de ligne, par exemple, est un système complexe qui comprend des phénomènes physiques (comme le déplacement d'un avion ou le vent qu'il subit), des dispositifs concrets (comme les capteurs ou les moteurs de cet avion) et des ordinateurs permettant la prise en compte des informations et la commande des dispositifs. Nous pouvons décomposer ce système complexe en un *système physique* (le monde dans lequel évolue l'avion), un *système mécanique et électronique* (l'avion lui même) et un *système informatique* (ce qui s'exécute sur les ordinateurs). Les deux premiers systèmes constituent *l'environnement* du système informatique. Les ordinateurs exécutent des *logiciels* (des *programmes* informatiques) dont la conception est délicate. En effet, elle doit déjà résoudre les difficultés spécifiques au domaine ; mais il faut de plus que les défauts de conception soient détectables. Il vaut mieux, par exemple, être capable de repérer une défaillance du logiciel de guidage d'un avion pendant la phase de conception plutôt que lors de son utilisation en plein ciel.

La première des difficultés liées aux systèmes temps-réel est qu'il faut en général faire face à des contraintes temporelles fortes : ces systèmes sont le plus souvent utilisés pour des tâches critiques et ils doivent répondre dans des délais très stricts dont le non respect

peut entraîner des catastrophes. Les systèmes temps-réel sont confrontés au temps de l'environnement avec lequel ils interagissent, c'est le temps *réel* d'où provient le nom de ces systèmes. Dans le cas du guidage d'un avion, il est impensable que ce dernier s'immobilise en plein vol pour attendre que le logiciel qui le guide effectue ses calculs : c'est donc au logiciel de calculer assez vite.

Ensuite, ces systèmes peuvent être composés de différentes parties actives, différentes *tâches*, opérant au même moment : ce sont des *systèmes parallèles*. Cette multiplication des activités amène des complications qui n'existent pas dans les systèmes où une seule tâche s'exécute. En particulier, on doit résoudre le problème de l'accès concurrent à des ressources communes. Il est par exemple inadmissible que le logiciel qui surveille le moteur d'un avion ne puisse pas signaler une panne parce que la voie de communication vers la cabine de pilotage est occupée par des messages de moindre importance.

Bien souvent, les tâches qui composent un système ne s'exécutent pas toutes au même endroit, mais plutôt sur différents ordinateurs : on a alors des *systèmes répartis*. Dans notre avion de ligne, les programmes permettant de gérer les capteurs liés aux moteurs s'exécutent sur des ordinateurs incorporés aux moteurs eux-mêmes. Dans ces conditions, les communications entre les tâches deviennent cruciales et les organiser de manière efficace est souvent un problème difficile. Il est en particulier impératif que les échanges d'informations respectent des contraintes de temps permettant aux tâches qui communiquent de respecter leurs propres contraintes.

Finalement, la complexité naît aussi de la taille même des systèmes. Ils sont généralement composés d'un très grand nombre de tâches, s'exécutant dans des conditions, sous des contraintes et en des lieux très divers. Structurer de tels systèmes peut se révéler un travail délicat et les interactions entre tous les paramètres en jeu sont souvent la source d'importantes difficultés. On imagine facilement que les gros avions modernes sont composés d'une multitude d'éléments gérés par informatique, cette multitude doit être organisée rigoureusement pour former un tout cohérent.

Toutes ces sources de complexité sont l'objet de très nombreux travaux théoriques et d'expérimentations en tous genres. Le but en est souvent de permettre le traitement automatique des problèmes en autorisant leur représentation dans un formalisme satisfaisant, permettant notamment la preuve de propriétés (de façon automatique ou semi-automatique). Ces formalismes sont appelés *modèles* et sont au cœur de nombreuses études dont l'objet est fréquemment (c'est le cas dans cette thèse) de fournir un modèle permettant de résoudre au mieux les difficultés énoncées ci-dessus.

1.1 Modèles et vérification

De plus en plus de problèmes sont confiés à l'informatique qui se doit d'y apporter des solutions satisfaisantes. Concrètement, il s'agit le plus souvent de fournir de « bons » programmes, ou, au moins, le moyen d'y parvenir. Ce qu'est un « bon » programme est principalement affaire de goût lorsqu'il s'agit d'un jeu vidéo, mais c'est affaire de vies humaines lorsqu'il s'agit de piloter un avion. Dans le contexte du temps-réel, les programmes ont souvent à gérer des systèmes critiques, leur première qualité est alors

une *fiabilité* à toute épreuve.

Pour assurer la fiabilité des programmes, on peut utiliser les ressources de l'informatique théorique et on se sert de *modèles formels* qui sont des moyens de représenter rigoureusement les systèmes afin d'établir des preuves de leurs propriétés. Dans notre exemple aéronautique, on peut vouloir prouver que jamais le programme de pilote automatique ne se bloque subitement (comme cela arrive si souvent à certains logiciels de bureautique). Les modèles sont très variés mais ils proposent tous une façon de représenter les systèmes et leurs propriétés. Cette étape de représentation symbolique est la *modélisation* du système. Elle aboutit à une version complètement formelle du système sur laquelle la théorie permet de travailler et de *prouver* (mathématiquement) ou de *vérifier* (avec certitude) des propriétés telles que la fiabilité ou l'absence de blocage.

La démarche de modélisation peut s'appliquer au système concret avec lequel on doit interagir par un programme ; cela permet notamment de mieux comprendre les problèmes en jeu et de pouvoir y apporter une réponse appropriée. On peut aussi modéliser le programme lui même, pour l'analyser et assurer qu'il possède les qualités attendues. Enfin, on peut se placer à un niveau plus abstrait et modéliser une *spécification* du programme, c'est-à-dire une représentation mettant en avant les aspects essentiels et éliminant ce qui relève de la technique pure. Cette abstraction sert souvent lors de la phase de conception d'un programme, il est important de pouvoir l'analyser et de détecter ses défauts avant d'avoir réalisé un programme complet (tâche souvent plus longue, plus coûteuse et plus complexe). Une spécification rigoureuse sera ensuite le point de départ de la conception du programme. On s'applique alors à réaliser un programme le plus fidèle à la spécification ; pour cela, on utilise autant que faire se peut des méthodes automatiques ou semi-automatiques assurant la conservation des propriétés prouvées sur la spécification.

Le modèle formel pouvant intervenir à tous ces niveaux, on peut mettre en relation les différentes modélisations obtenues et les comparer sur le plan des propriétés formelles. Il est donc important que les liens entre les différents niveaux soient les plus forts possibles. Dans l'idéal, on réalisera une modélisation formelle de l'environnement, une spécification au moyen du même modèle et on dérivera le programme de manière automatique à partir de cette spécification. Ainsi, les propriétés vérifiées sur la spécification formelle (en regard de la modélisation de l'environnement) seront reportées dans le programme. Cet idéal est rarement atteint car l'obtention d'un programme passe souvent par des étapes informelles (comme la traduction de la spécification en langage de programmation) ou ne garantissant pas la conservation des propriétés. Même dans l'idéal, il faut encore que la modélisation de l'environnement concret soit correcte, ou en tout cas suffisamment fidèle à la réalité. Ce dernier critère est en général le plus difficile à assurer car les environnements auxquels on est confronté en pratique sont extrêmement complexes et leur modélisation n'est valable que sous un certain nombre d'hypothèses qu'il convient d'expliciter. Pour modéliser l'environnement d'un avion, on peut par exemple supposer que ce dernier ne rencontre jamais le sol à dix mille mètres d'altitude. C'est une hypothèse raisonnable mais qui peut avoir des conséquences importantes en cas de défaillance de l'altimètre (si l'avion se trouve à mille mètres et que l'ordinateur de bord croit être à dix

mille mètres, il pourrait ne pas prendre en compte ce qui se passe au sol). La démarche même de modélisation de l'environnement permet de mettre à jour et d'expliciter les hypothèses sur lesquelles le reste est fondé.

Ces considérations nous amènent à dégager plusieurs qualités que doit posséder un modèle pour servir de la manière que nous avons décrite.

1. *Représentation satisfaisante du parallélisme.* On doit pouvoir caractériser et différencier la simultanéité, la dépendance ou l'indépendance des activités d'un système. Ce point est crucial lorsqu'il s'agit de représenter le monde réel dans lequel des événements peuvent avoir n'importe laquelle de ces caractéristiques.

2. *Représentation des communications.* Il faut être capable de mettre en valeur les communications entre les différentes parties d'un système complexe ; c'est particulièrement important dans le cas de systèmes répartis pour lesquels la communication entre les composants joue un rôle primordial.

3. *Modularité.* Pour traiter des systèmes complexes, il faut avoir la possibilité de les *décomposer* en sous-systèmes plus simples à modéliser, puis pouvoir *recomposer* les « sous-modélisations » ainsi obtenues en un tout cohérent qui forme la modélisation du système de départ.

4. *Formalisme.* La représentation d'un système doit pouvoir être analysée et donc posséder les qualités formelles permettant la preuve. On ne saurait se contenter de modèles informels dans lesquels rien ne peut être montré avec certitude.

5. *Réalisme.* Pour permettre la réalisation de programmes concrets, un modèle doit s'approcher le plus possible d'une réalisation dans un langage de programmation. Il doit donc être capable de représenter des systèmes concrets de manière naturelle. En particulier, le modèle doit permettre de représenter les *données* manipulées par les programmes.

6. *Représentation du temps.* Afin de traiter les systèmes temps-réel, il faut pouvoir faire référence au temps (à des durées ou à des dates) pour représenter sa progression ou les traitements dont le comportement dépend de l'écoulement du temps.

La question de la représentation du temps est un point particulièrement important lorsqu'on s'intéresse aux systèmes temps-réel. En effet, de la façon dont les modèles représentent et font référence au temps dépendent beaucoup leurs qualités et limites.

1.2 Problématique et approche proposée

Il existe de nombreux modèles permettant de traiter les problèmes du temps-réel. Nous donnons au chapitre 2 un aperçu des principales approches et des techniques de vérification qu'elles permettent d'envisager. Si nous confrontons ces modèles aux critères énumérés plus haut (en particulier la représentation du parallélisme et celle des données), nous pouvons noter plusieurs motifs d'insatisfaction :

- dans la plupart des cas, le parallélisme se réduit à l'entrelacement des événements ;
- les modèles parallèles ne représentent pas les données de manière satisfaisante ;
- le temps est intégré comme un concept externe.

Le dernier point constate que les modèles temporisés sont basés (par extension ou par inspiration) sur des modèles non temporisés qu'on a augmentés d'une représentation du temps. Cette approche amène à adapter tout ce qui existe pour tenir compte du temps : les sémantiques, les propriétés, les preuves, les méthodes de travail, les techniques de vérification, les logiciels développés, etc. Tout le domaine doit être mis à jour puisqu'on change de modèle. En particulier, il arrive souvent que l'ajout du temps fasse perdre des propriétés (par exemple, certaines questions ne peuvent plus être vérifiées) ou rendent les problèmes beaucoup plus complexes à résoudre (ce qui peut empêcher en pratique leur traitement automatique).

En nous basant sur cette constatation, nous avons cherché un moyen d'obtenir un modèle permettant de représenter le temps en évitant ces inconvénients. *L'algèbre des M-nets* répond correctement à la plupart des critères énoncés plus haut mais ne proposait pas, lorsque cette thèse a débuté, de représentation du temps. Il s'agit donc d'un modèle entièrement basé sur la relation de *causalité* entre les événements (le lien de cause à effet). Cette thèse a donc pour point de départ l'interrogation suivante :

> *Dans quelle mesure un modèle purement causal peut-il permettre de traiter les problèmes liés au temps-réel ?*

Dans ce mémoire, nous nous attacherons à répondre à cette question en gardant à l'esprit les points évoqués plus haut. En particulier, nous allons travailler pour l'essentiel sur l'algèbre des M-nets qui permet une représentation de haut niveau, notamment des données. Comme premier élément de réponse, nous adaptons à cette algèbre l'approche du *temps causal* qui, au lieu d'étendre les modèles, choisit de représenter le temps comme une composante de ceux-ci (avec des horloges explicitement modélisées). Nous étudions ensuite les conséquences pratiques et théoriques de cette approche.

L'introduction du temps causal permet la *temporisation* des systèmes. Bien souvent, les systèmes temps-réel font appel à la notion de *préemption* permettant d'interrompre l'exécution de certaines tâches (par exemple pour réagir au non respect d'une échéance). Cela motive l'introduction d'une extension des M-nets avec des opérateurs pour modéliser la préemption. Cette extension constitue notre deuxième élément de réponse.

Le reste de cette introduction est consacré à la présentation du concept de temps causal dans le cadre qui nous intéresse.

1.2.1 Temps et horloges

La nature du temps est l'objet de débats contradictoires dans un grand nombre de disciplines (en particulier en physique et en philosophie). Sans entrer dans ces débats, nous pouvons faire une distinction importante entre *deux temps* bien distincts. Le *temps réel* est celui dont la nature nous échappe, c'est le temps du monde dans lequel les systèmes informatiques sont plongés, c'est un temps idéal qui reste toujours inaccessible. De l'autre côté, le *temps concret* est celui des machines, celui qui peut être manipulé ; il est donné

par des *horloges* auxquelles on se réfère et qui sont généralement des composants des systèmes informatiques. Le temps réel est donc toujours *externe* aux systèmes informatiques alors que le temps des horloges peut être *interne*. Cette distinction est discutée en détail dans [Nis97, chap. 2] où ces deux notions de temps sont mathématiquement mises en relation. Remarquons que nous appelons systèmes *temps-réel* (avec un trait d'union) les systèmes mis en relation avec le *temps réel* (sans trait d'union), c'est-à-dire plongés dans un environnement indépendant avec lequel ils interagissent.

La distinction temps réel/temps concret est importante quand on s'intéresse à la représentation du temps au niveau d'un modèle, soit pour définir un modèle temporisé soit pour choisir un modèle approprié à un problème qu'on se pose. Le choix du temps réel amène à un modèle idéal (du point de vue du temps) qui, pour être analysé automatiquement, doit être traité symboliquement ou transformé en un modèle à temps concret. Le choix d'un temps donné par une horloge permet d'obtenir un modèle plus réaliste mais impose de tenir compte des spécificités de cette représentation du temps. En effet, le temps ressenti intuitivement est généralement celui du monde réel ; l'utilisation d'un autre temps impose une gymnastique de l'esprit à la personne qui utilise le modèle. Remarquons enfin que des contraintes telles que l'efficacité (et parfois la possibilité) des traitements automatisés peuvent aussi orienter le choix.

1.2.2 Horloges causales

En informatique, en dernier recours, les systèmes qu'on étudie fonctionnent sur des ordinateurs pour lesquels la seule référence au temps est leur horloge interne. Une telle horloge est construite à partir d'un événement qui se répète en produisant un changement d'état observable : un oscillateur. Au dessus de ce mécanisme élémentaire, on ajoute généralement un compteur et des programmes qui convertissent un nombre d'oscillations en unités plus habituelles (secondes, années, etc.).

Depuis que les ordinateurs peuvent communiquer entre eux, on s'est posé la question de la *synchronisation* de leurs horloges. On a mis au point des systèmes et des protocoles permettant à chaque ordinateur de modifier sa propre horloge en cohérence avec d'autres. Même alors, ce n'est pas l'oscillateur qui change, mais la façon dont on prend en compte ses impulsions.

En ce qui concerne le temps, nous pouvons donc considérer qu'un ordinateur, et donc un système informatique, est un *monde fermé* (c'est *l'hypothèse du monde clos*). De plus, le temps auquel nous pouvons faire référence est donné par une horloge. Il s'agit d'un temps discret, dont la progression est indiquée par l'occurrence de *tics* ponctuels. Ce point de vue pragmatique permet de simplifier grandement la démarche de représentation du temps tout en permettant une approche plus réaliste, puisque dictée par les contraintes matérielles (sur lesquelles on finit toujours par buter).

Il est parfaitement possible de construire un oscillateur dans un contexte purement causal. Il faut pour cela disposer d'un événement de référence, le *tic d'horloge*, et assurer que son occurrence est toujours prise en compte. Pour cela, il est nécessaire que le tic amène un changement d'état du système. En effet, dans un cadre purement causal, un événement n'est observable par le système que s'il implique un changement d'état. Ce

changement doit être pris en compte (par exemple par un compteur) avant que le tic ne se reproduise. Il est logiquement équivalent de dire : *un tic ne peut se produire que si le précédent a été pris en compte*. Cela définit une double dépendance causale : d'une part, le système dépend du tic qu'il doit prendre en compte ; d'autre part, le tic dépend de la partie du système qui le prend en compte.

Cette dépendance mutuelle est parfaitement admissible sous l'hypothèse du mode clos et quand la causalité est le seul moyen de lier les événements. Cela ne signifie pas un couplage parfait entre le système et l'oscillateur. En fait, il suffit de coupler ce dernier avec un compteur qui devient alors la référence pour le reste du système. Le degré de couplage entre le compteur et les autres parties du système peut alors être quelconque puisque nous assurons déjà que le compteur ne manque aucun tic.

L'approche purement causale pour la représentation du temps est discutée en profondeur et d'une manière non technique dans [Dur91]. Dans [Ric85, Ric98] elle est appliquée à la construction d'horloges ayant pour but d'être « branchées » dans des systèmes. Cette approche est convaincante dans son intention mais souffre dans la réalisation d'un manque de simplicité ; en particulier, la construction des horloges est complexe et leur utilisation est difficile. Ces difficultés semblent avoir pour cause l'utilisation d'un modèle faiblement structuré.

Nous montrons dans la suite de ce chapitre comment des horloges plus génériques peuvent être construites en utilisant l'algèbre des M-nets. Pour cela, nous allons procéder par étapes en utilisant successivement des *réseaux de Petri* simples (qui constituent le modèle sur lequel les M-nets sont basés) puis des réseaux colorés (plus généraux) pour enfin aboutir à une version sous forme de M-nets.

1.2.3 Réseaux de Petri

Les réseaux de Petri constituent le cadre idéal pour traiter de la causalité. En effet, il s'agit d'une famille de modèles tous basés sur la même idée simple : des ressources locales peuvent être partagées par des activités ; ces activités consomment certaines ressources et en produisent d'autres. La relation de causalité est mise en avant puisque certaines activités dépendent directement des ressources produites par d'autres activités. De même, les conflits sur les ressources ou l'indépendance des activités sont aussi rendus explicites.

Parmi les plus simples des modèles de réseaux de Petri, on trouve les réseaux *places/transitions* (*réseaux P/T*) [Petri62, Pet81]. Un réseau places/transitions est constitué d'un *ensemble de places*, d'un *ensemble de transitions* (disjoint de celui des places), et d'arcs reliant places et transitions. Dans la suite, nous utiliserons la représentation graphique habituelle des réseaux de Petri comme dans la figure 1.1.

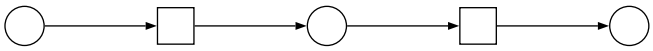


Figure 1.1 — La représentation graphique d'un réseau $N_{1.1}$ qui comporte trois places, représentées par des cercles, deux transitions, représentées par des carrés et quatre arcs, représentés par des flèches.

Afin de faire fonctionner un réseau P/T, il faut lui assigner un *marquage* ; il s'agit de donner pour chaque place le nombre de *jetons* qu'elle contient. Les jetons constituent les ressources d'un réseau, locales à chaque place, et permettent son activité. Le réseau $N_{1.1}$ de la figure 1.1 a le marquage vide ; nous le retrouvons à la figure 1.2 avec un autre marquage.

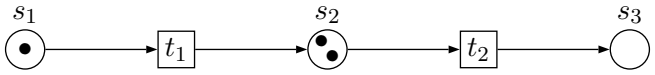


Figure 1.2 — La représentation graphique du réseau marqué $N_{1.2}$. Il s'agit du réseau $N_{1.1}$ de la figure 1.1 auquel nous avons ajouté des jetons représentés par des points noirs, ainsi que les noms des places et des transitions.

Un réseau marqué peut *s'exécuter* en *franchissant des transitions*. Chaque franchissement de transition est une activité élémentaire du réseau. Une transition t est franchissable si chaque place s qui envoie un arc sur t contient un jeton. Le franchissement de t consomme un jeton pour chacun des arcs entrant sur t et en produit un pour chacun des arcs sortant de t. (Il n'y a donc pas forcément conservation du nombre de jetons.) Par exemple, après le franchissement de t_1 dans le réseau $N_{1.2}$, nous aurions trois jetons dans s_2 et aucun jeton ailleurs.

La notion de franchissabilité et la règle de franchissement (aussi appelée règle de transition) donnée ici s'étendent à des ensembles (avec répétitions éventuelles) de transitions, appelés *steps*, permettant l'occurrence simultanée de plusieurs transitions (y compris plusieurs fois la même). Par exemple, le réseau $N_{1.2}$ peut franchir le step $\{t_1, t_2, t_2\}$ qui conduit au marquage dans lequel s_1 est vide, s_2 contient un jeton et s_3 en contient deux. Le réseau $N_{1.2}$ peut exécuter plusieurs steps à la suite : l'ensemble de ces séquences constitue sa *sémantique steps* [Gen+80, BD87, Bes96, BF88, GR83] ; elle contient notamment les séquences $\{t_1, t_2, t_2\}\{t_2\}$ et $\{t_1\}\{t_2, t_2, t_2\}$.

Pour introduire une représentation causale du temps dans un réseau P/T, nous lui ajoutons des *transitions de tic* de façon à exprimer les contraintes de temps souhaitées. Par exemple, dans le réseau $N_{1.2}$, nous pouvons forcer l'occurrence d'un tic entre le franchissement de t_1 et celui de t_2 en ajoutant une transition de tic comme illustré sur la figure 1.3. Cette fois-ci, la séquence de steps $\{t_1, t_2, t_2\}\{t_2\}$ n'est plus exécutable ; en fait, la seule séquence exécutable qui franchit t_1 et t_2 est $\{t_1\}\{tic\}\{t_2\}$, ce qui correspond bien à ce que nous voulions spécifier.

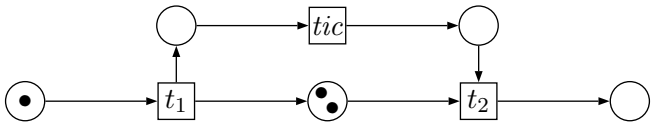


Figure 1.3 — Modification de $N_{1.2}$ pour forcer un tic entre t_1 et t_2. Après le franchissement de t_1, il manque un jeton dans la place au dessus de la transition t_2 pour quelle soit franchissable ; ce jeton sera produit par le franchissement de *tic*.

Nous pouvons bien entendu complexifier l'exemple à loisir en ajoutant des transitions de tic. Cependant, nous arrivons très rapidement à des situations impossibles à gérer, à cause justement de la complexité des réseaux obtenus. Par exemple, la contrainte « deux tics au plus peuvent survenir entre t_1 et t_2 » amène à construire le réseau représenté figure 1.4, où t_2 a dû être éclatée en trois (pour les trois cas possibles de franchissement : après 0, 1 ou 2 tics). D'autre part, la multiplication des transitions de tic rend plus difficile leur identification : comme nous l'avons fait, par abus, sur la figure 1.4, nous aimerions avoir un seul nom, *tic*, pour toutes les transitions de tic ; il faudrait donc n'en avoir qu'une. En ajoutant un jeton dans la place de gauche du réseau de la figure 1.4, la plus longue séquence de steps franchissable est $\{t_1\}\{tic\}\{tic\}\{t_2^2\}$.

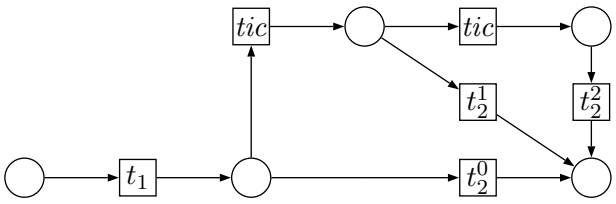


Figure 1.4 — Le réseau $N_{1.1}$ dans lequel nous imposons au plus deux tics entre t_1 et l'un des t_2^i $(0 \leq i \leq 2)$.

1.2.4 Réseaux de Petri colorés

Une première solution face à la complexité engendrée par l'ajout de transitions de tic est d'utiliser des *réseaux de Petri colorés* [Jen81, Jen92]. De tels réseaux permettent une représentation compacte des réseaux P/T. L'idée est que les places peuvent contenir plusieurs types de jetons. À l'origine, on utilisait des jetons de différentes couleurs, d'où le nom de la classe de réseaux. Par la suite, on a simplement utilisé différents symboles comme jetons. Par exemple, les jetons noirs habituels ($\bullet$), des entiers, des booléens ($\bot$ et $\top$), etc.

Pour permettre cela, les réseaux P/T sont augmentés d'un *étiquetage* qui décore leurs éléments (places, transitions et arcs). L'étiquette d'une place (on dit aussi son *inscription*) est constituée de l'énumération des valeurs de jetons qu'elle peut contenir : on l'appelle le *type* de la place. L'étiquette d'un arc est un ensemble (avec répétitions éventuelles) d'expressions (les expressions les plus simples étant des constantes ou des variables) indiquant ce que l'arc peut transporter lors d'un franchissement de transition (les expressions s'évaluent à ce moment en valeurs concrètes). Une transition est étiquetée par une expression booléenne sur les variables des arcs adjacents, appelée *garde*, qui doit s'évaluer à vrai pour que la transition soit franchissable. La figure 1.5 donne un réseau coloré équivalent à celui de la figure 1.4. Comme nous l'avons fait dans cette figure, nous convenons pour la suite que les places non étiquetées ont le type $\{\bullet\}$, qui est aussi l'inscription des arcs non étiquetés, et que les transitions sans étiquette portent une garde toujours vraie ($\top$).

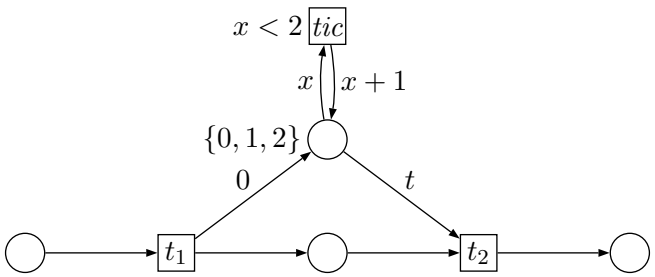


Figure 1.5 — Un réseau coloré abrégeant le réseau P/T de la figure 1.4.

Avec les réseaux colorés, le franchissement des transitions repose sur une *valuation* des variables de la transition et des arcs adjacents : on associe une valeur à chaque variable, et la transition peut être franchie sous cette valuation si elle respecte le type des places adjacentes et si sa garde s'évalue à $\top$. (Remarquons que dans la figure 1.5, la garde $x < 2$ n'est pas nécessaire compte tenu du type de la place centrale.) Un step est donc constitué de transitions associées à des valuations et non plus de transitions seules. Par exemple, si nous déposons un jeton $\bullet$ dans la place de gauche du réseau de la figure 1.5, la plus longue séquence de steps franchissable est $\{t_1\}\{tic[x = 0]\}\{tic[x = 1]\}\{t_2[t = 2]\}$, qui correspond à la séquence de la section précédente. (Nous avons indiqué entre crochets les valuations des variables impliquées dans chaque franchissement.)

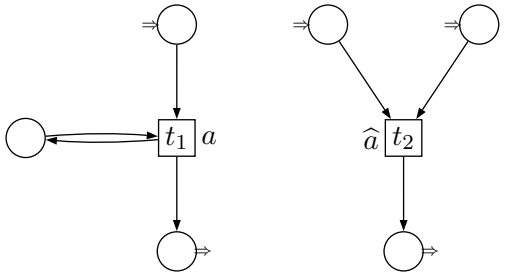
L'utilisation de réseaux colorés ne résout que partiellement les problèmes évoqués à la section précédente : en effet, s'il est clair qu'ils permettent de gagner en compacité et d'éviter l'éclatement des transitions (comme t_2), l'utilisation de contraintes en différents points d'un réseau complexe risque cependant de multiplier les transitions de tic, ce qui n'est pas souhaitable si nous voulons pouvoir utiliser une unique référence de temps pour tout un réseau.

1.2.5 Les M-nets : des réseaux composables

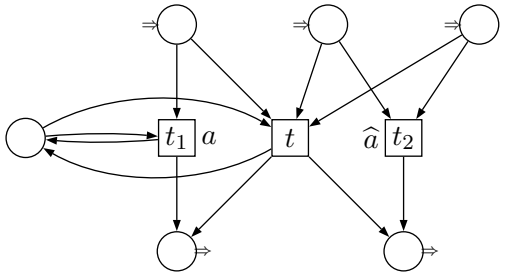
Le modèle des M-nets [Kla95, Bes+98] est une classe de réseaux colorés qu'on peut composer en utilisant divers opérateurs. Par exemple, si N_1 et N_2 sont deux M-nets, $N_1 \| N_2$ est un M-net représentant la composition parallèle des deux précédents (ils peuvent donc s'exécuter en totale concurrence). Ou encore, avec la composition séquentielle, $N_1; N_2$ est un M-net dont l'exécution correspond à celle de N_1 suivie par celle de N_2. Pour réaliser ce type de compositions, on utilise une annotation supplémentaire des places qui donne leurs *statuts* : `e` pour une place *d'entrée*, `i` pour une *place interne* et `x` pour une *place de sortie* (la lettre `x` venant du mot *exit*). Le marquage des places d'entrée correspond à l'état initial d'un M-net, celui-ci s'exécute ensuite en marquant ses places internes, puis il termine son exécution en marquant ses places de sortie. Ainsi, la composition séquentielle $N_1; N_2$ revient à agréger les places de sortie de N_1 et les places d'entrée de N_2, ce qui permet le comportement souhaité.

Pour alléger les figures, le statut des places est indiqué par une convention graphique :

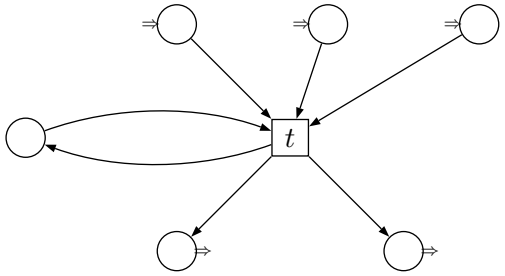
une place d'entrée est indiquée par une flèche (⇒) pointant dessus ; pour une place de sortie, la flèche est inversée ; une place interne ne porte aucune indication spéciale.



Le M-net $N_{1.6}$ dans lequel les transitions t_1 et t_2 peuvent être franchies indépendamment.



Le M-net $N_{1.6}\,\mathsf{sy}\,a$. Le franchissement de la nouvelle transition t est équivalent au franchissement simultané de t_1 et t_2 puisque tous leurs arcs ont été hérités par t. Les actions ayant servi à la synchronisation ne sont pas reportées sur la nouvelle transition.



Dans le M-net $(N_{1.6}\,\mathsf{sy}\,a)\,\mathsf{rs}\,a$, les transitions t_1 et t_2 ayant été supprimées, seul le franchissement simultané reste possible.

Figure 1.6 — Synchronisation et restriction dans les M-nets.

Outre ce type de composition permettant le *contrôle de flot*, on dispose d'opérations permettant d'établir des *communications synchrones* entre les transitions d'un M-net. Celles-ci portent, en plus de leurs gardes, un nombre arbitraire *d'actions*. Il s'agit d'expressions de la forme a ou $\widehat{a}$ qui permettent aux transitions de se *synchroniser* et donc de s'exécuter simultanément. Ces actions peuvent avoir des paramètres qui sont unifiés lors de la synchronisation ; par exemple, en synchronisant $a(x)$ et $\widehat{a}(y)$, x et y sont unifiés ce qui réalise une communication entre les deux transitions portant ces annotations. L'opérateur permettant de synchroniser les paires d'actions a et $\widehat{a}$ est noté $\mathsf{sy}\,a$. La synchronisation est généralement suivie d'une *restriction* qui rend impossible l'exécution non simultanée des transitions synchronisées et les empêche d'être synchronisées à nouveau. La restriction des actions a et $\widehat{a}$ est réalisée par l'opérateur $\mathsf{rs}\,a$. Une synchronisation suivie d'une restriction est appelée *scoping* et notée sc ; pour tout M-net N et toute action a on a : $N\,\mathsf{sc}\,a \stackrel{\text{df}}{=} (N\,\mathsf{sy}\,a)\,\mathsf{rs}\,a$. On utilise souvent une notation étendue avec

plusieurs actions, par exemple : $N \,\mathsf{sy}\, \{a_1, a_2\}$ au lieu de $(N \,\mathsf{sy}\, a_1) \,\mathsf{sy}\, a_2$ (ou de manière équivalente $(N \,\mathsf{sy}\, a_2) \,\mathsf{sy}\, a_1$). La figure 1.6 illustre ce mode de communication synchrone.

En utilisant les M-nets, nous pouvons construire une horloge générique utilisable depuis n'importe quel point d'un réseau. Cette horloge permet de modifier et de consulter un compteur de tics. Ces deux opérations sont réalisées au moyen d'une seule action $clock(c, c')$ qui permet de récupérer dans c le nombre de tics déjà comptés et de positionner le compteur à c'. Une telle horloge est présentée sur la figure 1.7.

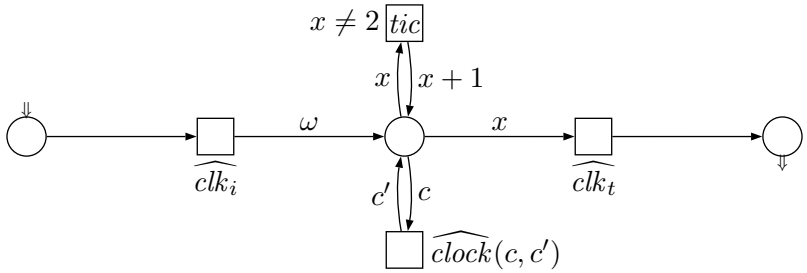


Figure 1.7 — Le M-net $N_{1.7}$ définit une horloge causale pouvant compter jusqu'à 2 tics. La place interne (au centre) a le type $\{0, 1, 2, \omega\}$, où ω est une constante telle que $\omega > 2$ et $\omega + 1 = \omega$.

Remarquons qu'ici encore le nombre maximal de tics à compter est mémorisé dans l'horloge (d'où la constante 2 dans la garde de la transition *tic*). Cela correspond au couplage du système et de l'horloge (*via* le compteur), et constitue la traduction en termes de réseaux de conditions telles que « au maximum 2 tics peuvent survenir ». Remarquons aussi que, puisque $\omega + 1 = \omega$, l'horloge ne compte rien lorsque le compteur est positionné à ω, mais elle continue de produire des tics. Cette caractéristique sera exploitée plus tard, lorsque nous aurons plusieurs compteurs dans une seule horloge.

Pour utiliser cette horloge et imposer sur deux transitions t_1 et t_2 le même genre de contraintes que dans la figure 1.5, nous ajoutons des annotations sur ces transitions pour qu'elles puissent communiquer avec l'horloge (voir figure 1.8) et nous plaçons ce réseau en parallèle avec l'horloge avant d'établir les communications : $(N_{1.8} \| N_{1.7}) \,\mathsf{sc}\, clock$.

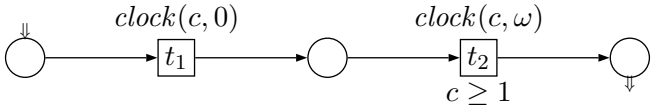


Figure 1.8 — Le réseau $N_{1.8}$ dans lequel nous forçons l'occurrence d'un ou deux tics entre t_1 et t_2.

Pour être complet, nous devons assurer l'initialisation et la terminaison de l'horloge (avec les actions clk_i et clk_t respectivement). D'une manière générale, étant donné un M-net N_{sys} représentant un système (par exemple celui de la figure 1.8), nous construirons

pour le temporiser le M-net :

$$N_{1.10} \stackrel{\text{df}}{=} \Big((N_i; N_{sys}; N_t) \Big\| N_{1.7} \Big) \,\mathsf{sc}\, \{clock, clk_i, clk_t\} \quad ,$$

qui est présenté figure 1.10 alors que N_i et N_t sont donnés figure 1.9.

Figure 1.9 — À gauche, le M-net N_i permettant d'initialiser l'horloge ; à droite, le M-net N_t permettant de la terminer.

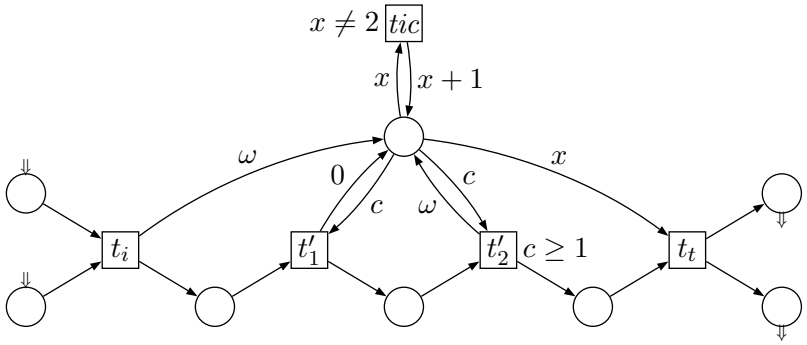


Figure 1.10 — Le M-net $N_{1.10}$. Les transitions t'_1 et t'_2 proviennent du scoping sur *clock*, alors que t_i et t_t sont issues des scopings sur clk_i et clk_t.

L'utilisation d'un M-net horloge est donc très simple. Pour ajouter encore de la souplesse, nous construisons dans la suite (au chapitre 4) des horloges permettant de réaliser plusieurs comptages simultanés, d'obtenir dynamiquement un identificateur pour chaque comptage et de définir, dynamiquement aussi, le maximum de tics à compter.

Liens asynchrones. Les identificateurs étant alloués dynamiquement, il nous faut les transmettre entre deux transitions. C'est assez facile dans des cas simples comme ceux présentés jusqu'ici, mais c'est plus difficile en général, surtout si les deux transitions viennent de réseaux différents qui ont été composés avec les opérateurs de l'algèbre (en séquence par exemple). Pour permettre cette transmission de manière simple et efficace, nous étendons les M-nets avec des *liens asynchrones* [1] permettant de modéliser, au niveau de l'algèbre, des communications asynchrones entre transitions. Pour réaliser ces communications, nous ajoutons une nouvelle classe de places, les *places de liens asynchrones* (ou simplement *places de liens*), identifiées par des statuts particuliers correspondant à des *symboles de liens*. Les places de liens ayant le même statut sont automatiquement fusionnées lors de la composition de plusieurs réseaux par un des opérateurs de l'algèbre, leurs marquages étant alors cumulés. Ainsi, la communication asynchrone est rendue possible. Cette fusion est illustrée sur la figure 1.11 ; remarquons

que les places de liens sont représentées en gras pour les différencier des places internes (sauf que les marquages des places de liens peuvent être non bornés).

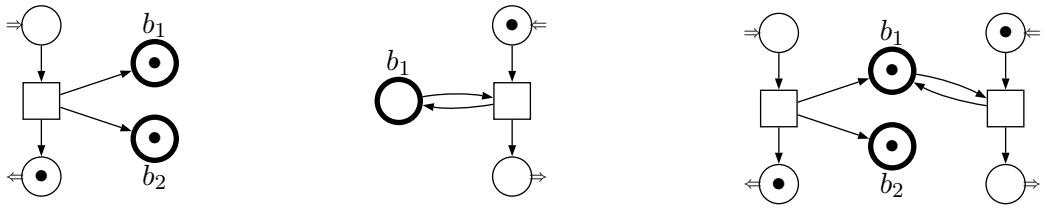


Figure 1.11 — De gauche à droite, les M-nets N_1, N_2 et $N_1 \| N_2$ dans lequel les places de liens issues des deux réseaux qui le composent ont fusionné.

En plus des places de liens, nous ajoutons un opérateur de *restriction de liens*, noté $\mathsf{tie}\, b$, qui change en b (pour *buffer*) le statut de la place de liens associée à b (empêchant ainsi sa fusion avec d'autres places par la suite) et ajoute une nouvelle place de liens isolée de statut b. Cela permet de rendre privées les communications déjà établies sur une place de liens. La figure 1.12 illustre le fonctionnement de cet opérateur ; nous y distinguons deux types de places de liens : les places *ouvertes* (représentées en gras) qui peuvent être fusionnées, et les places *fermées* (représentées avec un bord double qui symbolise graphiquement le statut b) qui ne peuvent plus être fusionnées et sont équivalentes à des places internes.

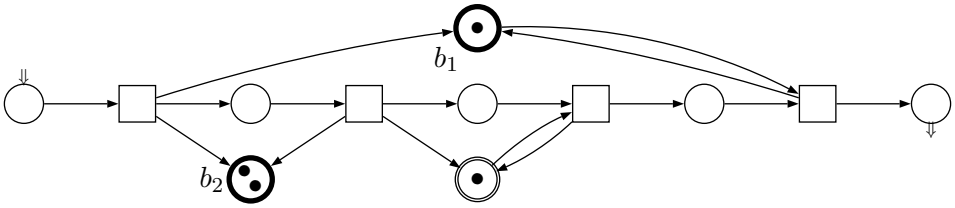


Figure 1.12 — Le M-net $N_1'; ((N_1'; N_2')\, \mathsf{tie}\, b_1); N_2'$, où N_1' et N_2' sont les M-nets N_1 et N_2 de la figure 1.11 dont les marquages sont restreints aux places de liens. Nous pouvons observer que les places de liens associées à b_2 ont fusionné alors que l'application de $\mathsf{tie}\, b_1$ a rendu privée la place de liens correspondant à b_1 dans la séquence $N_1'; N_2'$ (en changeant son statut en b).

1.3 Organisation de ce mémoire

Le chapitre 2 fait une synthèse des différentes approches formelles permettant de traiter les problèmes temps-réel. Nous présentons les principaux modèles et les techniques de vérification qu'ils proposent. Nous discutons aussi de la notion de préemption qui joue un rôle crucial dans beaucoup de problèmes temps-réel car elle permet de contrôler les activités d'un système en exécution en autorisant leur terminaison forcée ou leur suspension.

Le chapitre 3 introduit l'algèbre des M-nets ; en particulier, les définitions esquissées dans la section précédente sont précisément établies. Nous donnons aussi les propriétés essentielles des M-nets.

Le chapitre 4 détaille la présentation du temps causal faite dans cette introduction. Nous commençons par donner plusieurs modélisations d'horloges causales aux fonctionnalités différentes. Nous expliquons ensuite comment représenter des systèmes comportant plusieurs horloges (par exemple des systèmes répartis), indépendantes ou synchronisées. Nous terminons ce chapitre par la construction d'une machine d'exécution qui nous permet de montrer qu'une horloge causale modélise le temps de façon satisfaisante même sous des contraintes temps-réel.

Le chapitre 5 est consacré à l'étude d'un exemple concret. Nous modélisons un système de passage à niveau en utilisant des M-nets avec horloges causales et des automates temporisés. Les différentes modélisations sont vérifiées avec différents outils et les résultats expérimentaux sont présentés. Ils nous permettent d'illustrer la pertinence et l'efficacité de l'approche causale, ainsi que ses limites actuelles.

Au chapitre 6, nous étendons l'algèbre des M-nets pour y intégrer la notion de préemption. Ce concept n'est pas directement liée au temps mais est utilisé dans beaucoup d'applications temps-réel, ce qui motive le travail présenté dans ce chapitre.

Le chapitre 7 s'intéresse aux langages de programmation et à leur sémantique. Nous utilisons les modèles développés dans cette thèse pour étendre la sémantique d'un langage de programmation parallèle avec de nouvelles constructions (temporisation, exceptions et système de tâches).

Au chapitre 8 nous examinons en détail les conséquences de l'introduction des liens asynchrones dans un modèle de réseaux de Petri places/transitions composables sur lequel les M-nets sont basés. Cette étude nous donne les fondements théoriques pour envisager des travaux similaires sur les M-nets.

Nous concluons ce mémoire au chapitre 9 où nous présentons les travaux en cours et ceux que nous envisageons à l'avenir.

2 Modèles du temps-réel

Ce chapitre présente les principaux modèles permettant de traiter des problèmes temps-réel. Il existe de très nombreux modèles répondant à ce seul critère et il serait impossible de les présenter tous ici. Nous avons décidé de ne présenter ou citer que des modèles jouissant d'une certaine notoriété ou possédant une particularité remarquable. De plus, nous nous sommes restreints aux modèles formels, en éliminant donc tous ceux ne permettant pas de prouver des propriétés.

Nature du temps. Tous les modèles supposent ou sous-entendent l'existence d'un *axe du temps* (ou d'une *échelle du temps*) sur lequel on peut isoler des instants et mesurer des durées entre les instants. On peut donc classifier les représentations du temps en fonction des hypothèses qu'elles font sur la nature de celui-ci.

L'hypothèse « naturelle », c'est-à-dire celle qui rapproche le plus un modèle de ce que nous percevons intuitivement du temps, est de supposer que le temps est *dense*. On peut aussi le supposer continu, les propriétés mathématiques peuvent différer mais l'intuition reste la même : la progression du temps est un phénomène permanent et deux instants peuvent être arbitrairement proches. L'axe du temps est alors une droite, les instants sont des points de cette droite et les durées des nombres réels ou (généralement) rationnels (suivant qu'on considère un temps continu ou juste dense). On souhaite généralement fixer un instant particulier, appelé *origine du temps*, qui constitue une référence à partir de laquelle on peut *dater* les instants (une date est alors définie comme la durée entre un instant et l'origine du temps). Quand on s'intéresse à des systèmes informatiques, on considère volontiers l'axe du temps comme une demi-droite débutant à l'origine du temps. Cela revient à admettre que, du point de vue du système qu'on étudie, la notion de temps n'est pas pertinente avant son démarrage.

La donnée d'un temps continu (ou dense) suppose qu'on dispose d'un *pouvoir de séparation* des instants aussi fin que nécessaire. En effet, avec cette hypothèse, deux instants peuvent être différents mais arbitrairement proches. Il faut pour les distinguer une précision infinie, ce qui n'est réalisable avec aucun dispositif physique. Donc, si l'hypothèse du temps continu est séduisante, elle se révèle en fin de compte peu pragmatique. Beaucoup de modèles adoptent donc une vision *discrète* du temps. Le temps devient une succession

d'instants bien distincts ; ils sont les éléments d'un ensemble discret ordonné. Si on fixe une origine de temps, le temps peut être assimilé à l'ensemble des entiers, le nombre n représentant le n-ième instant. Si on admet que le temps n'existe pas avant l'origine, on peut utiliser les entiers naturels, sinon on prend plutôt les entiers relatifs.

Nature des systèmes. Indépendamment de la nature du temps qu'il considère, un modèle fait aussi des hypothèses sur la nature des systèmes qu'il peut représenter. On distingue généralement deux notions : les *états* d'un système sont les différentes configurations qu'il peut atteindre au cours de son évolution ; des *événements* peuvent se produire au cours de l'évolution d'un système. Lorsqu'ils amènent des changements d'état, on parle alors de *transitions* entre les états.

Lorsqu'on veut *observer* l'évolution d'un système (ou simplement la qualifier en fonction d'une observation potentielle), on induit implicitement des *relations temporelles* entre les états et les événements. Nous prendrons soin de noter la différence entre des relations *temporelles* qui sont qualitatives (avant, après, simultanément, pendant, etc.) et des relations *temporisées* qui sont quantitatives (quand, après combien de temps, pendant quelle durée, etc.). La temporalité permet de parler de l'ordonnancement des phénomènes dans le temps alors que la temporisation apporte une notion de mesure du temps.

L'observation amène aussi la question du *point de vue* de l'observateur : il peut être omniscient ou au contraire avoir une vue limitée du système ; on peut aussi supposer plusieurs observateurs partageant des vues locales afin de reconstituer une vue globale de l'évolution d'un système.

La *sémantique* d'un modèle définit généralement de quelle façon les exécutions des systèmes modélisés peuvent être observées. Un même modèle peut d'ailleurs posséder plusieurs sémantiques ne se basant pas nécessairement sur les mêmes hypothèses.

Les *sémantiques séquentielles* considèrent que tous les événements d'un système sont totalement ordonnés ou peuvent l'être. Pour ordonner ainsi tous les événements, on doit être capable d'observer globalement le système ; on représente alors les évolutions possibles d'un système comme un ensemble d'états avec des transitions entre eux. Les sémantiques séquentielles mettent donc en avant la dépendance entre les événements et l'enchaînement des états globaux.

Au contraire, les *sémantiques concurrentes* privilégient l'indépendance qu'elles représentent explicitement. On peut par exemple donner un ordre partiel sur les événements. Dans ce cas, les événements incomparables sont indépendants. En général, l'indépendance des événements provient d'une certaine localité de l'observation ou d'une observation partielle du comportement. Les évolutions possibles d'un système sont alors représentées par un ensemble d'états locaux, les transitions n'agissant pas forcément sur tous ces états à la fois.

On peut mettre en évidence la différence entre sémantiques concurrentes et séquentielles. Les premières n'ordonnent pas les événements qui ne sont pas causalement dépendants ou en conflit ; au contraire, les secondes *entrelacent* tous les événements (la simultanéité est alors simulée par identification d'événements). Considérons le réseau de

Petri de la figure 2.1, il a deux exécutions possibles, toutes deux séquentielles : t_1 suivie de t_2 ou bien t_2 suivie de t_1. Avec une sémantique séquentielle, on pourrait en déduire l'indépendance des deux événements. En réalité, ils sont en conflit sur l'utilisation du jeton de la place centrale et ne peuvent jamais être simultanés ; cela n'apparaît qu'en utilisant une sémantique concurrente. Par exemple, ce réseau n'admet pas le step $\{t_1, t_2\}$ ce qui permet de déduire le conflit. Par ailleurs, en ajoutant un jeton à la place centrale, on aurait la même sémantique séquentielle alors qu'une sémantique concurrente ferait apparaître la possibilité d'une exécution simultanée des deux transitions.

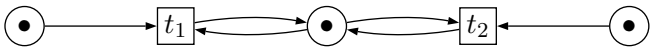


Figure 2.1 — Un réseau de Petri forçant l'entrelacement de ses transitions.

Relations entre états et événements. Dans ce qui précède, nous avons été amenés à des considérations *spatiales* à propos des états alors que les événements ont été envisagés selon des critères *temporels*. Cette même dichotomie se retrouve dans la plupart des modèles. Nous expliquons cela par la relation étroite qu'entretiennent états et événements dans l'intuition qu'on s'en fait.

Généralement des conditions sur la configuration d'un système servent à autoriser, interdire ou forcer l'occurrence d'un événement. Si l'état n'a pas d'influence sur l'événement, c'est que ce dernier est aléatoire (ce qui est rarement le cas de tous les événements d'un système intéressant). Les états, ou des vues locales des états, peuvent donc être pris comme conditions, inhibiteurs ou déclencheurs des événements. Leurs relations temporelles sont donc en grande partie déterminées par celles des événements auxquels ils sont liés. En particulier, un changement de configuration lors de l'évolution d'un système correspond à un changement d'état et peut être considéré comme un événement à part entière permettant de séparer temporellement les deux états.

De manière similaire, la localité spatiale des événements est en partie déterminée par celle des états. Si l'occurrence d'un événement est liée à des conditions locales sur une partie d'un système, cette même partie pourra être considéré comme le « lieu » de l'événement. D'autres questions de localité entrent en jeu pour les événements lorsque les modèles autorisent des représentations structurées. On peut alors avoir des événements locaux à des sous-systèmes, au même titre que l'on a des variables locales à un sous-programme. Cette localité est alors liée à la conception plutôt qu'à l'observation.

Plongement dans le temps. Lorsqu'on plonge l'évolution d'un système dans une représentation du temps, on doit fixer les caractéristiques temporelles des états et des événements. En particulier, on doit savoir s'ils correspondent à des instants ou à des durées.

La plupart des modèles faisant l'hypothèse du temps continu considèrent que les états ont une durée alors que les événements sont instantanés. Les modèles considérant des événements avec une durée peuvent en général être réduits à ceux où les événements

sont instantanés. En effet, on peut ajouter pour chaque événement un état modélisant son exécution et l'occurrence d'un événement correspond alors à cet état encadré des événements instantanés marquant son début et sa fin. Cependant, même avec cette réduction, on est toujours confronté à la difficulté de définir la simultanéité ou l'ordre des événements puisque leurs exécutions peuvent se recouvrir partiellement. Cette difficulté semble expliquer pourquoi ces modèles ont généralement moins de succès que ceux où les événements sont instantanés.

Les autres possibilités ne sont pas exploitées dans les modèles que nous présentons dans la suite. Cela peut s'expliquer par la difficulté de définir un état sans durée. En effet, si les événements sont aussi instantanés il devient difficile de les discerner des états ; si les événements ont une durée, en échangeant les notions d'états et d'événements, on se ramène à un modèle correspondant mieux à l'intuition.

Lorsqu'on prend l'hypothèse du temps discret, la plupart des modèles font aussi l'hypothèse que les états durent et que les événements sont instantanés, les raisons sont similaires. Avec un axe de temps discret il faut en plus choisir si les événements peuvent avoir lieu uniquement sur les instants de l'axe du temps ou entre ces instants. Le premier cas correspond aux systèmes *synchrones* dont tous les composants sont pilotés par une seule horloge globale. Pour représenter des systèmes *asynchrones*, il devient alors nécessaire de discrétiser le temps en quanta de durée fixée à priori, ce qui limite la précision avec laquelle les systèmes peuvent être modélisés. Dans ce cas, il est théoriquement possible que certains comportements soient « oubliés » dans la modélisation. Par exemple, dans [BS91], les auteurs montrent que certains problèmes ne peuvent pas être résolus correctement avec cette approche du temps.

Si on autorise les événements à survenir hors des points de l'axe du temps, on obtient un *modèle d'horloge* : un événement particulier, le tic d'horloge, est déclenché à chaque point de l'axe du temps. La durée entre deux événements correspond alors au nombre d'occurrences du tic. En considérant le tic comme un événement distingué, les événements suffisent à définir l'axe du temps puisqu'ils correspondent aux instants. On peut alors considérer que les états sont hors du temps et qu'ils n'ont pas d'autre durée que le nombre de tics pendant lesquels ils sont en place.

2.1 Modèles du temps

Nous présentons maintenant les principaux modèles formels intégrant une représentation du temps. Il s'agit le plus souvent d'évolutions ou d'extensions de modèles ayant fait la preuve de leur utilité pour l'étude de problèmes non temporisés.

2.1.1 Automates temporisés

Les automates temporisés [AD94] forment sans doute le modèle le plus réputé, ou en tout cas le plus utilisé. Il en existe de nombreuses variantes [Pett99] ; nous en présentons une que nous utiliserons au chapitre 5.

Un automate temporisé se compose d'un ensemble d'états, de transitions entre eux et

d'un ensemble de *chronomètres* permettant de mesurer le temps. (On parle plus traditionnellement *d'horloges* mais nous avons préféré marquer la différence avec les horloges causales.) L'exécution d'une transition est soumise à une condition portant sur les chronomètres ; lorsque la garde est vraie, la transition peut avoir lieu et assigner de nouvelles valeurs aux chronomètres tout en émettant des actions visibles. Dans certaines versions du modèle, les états portent des invariants qui doivent rester vrais tant que l'automate est dans l'état ; les invariants servent le plus souvent à forcer l'exécution d'une transition.

L'échelle du temps est supposée continue : les valeurs des chronomètres sont donc des nombres réels. De plus, les transitions sont instantanées. Le séjour dans les états peut aussi être instantané mais il peut durer, ce qui est le cas le plus souvent. Une exécution d'un automate temporisé est, à l'instar de celle d'un automate fini classique, un mot formé par la suite des actions émises par les transitions. La différence est que le passage de chaque transition est daté par rapport au démarrage de l'automate. Une autre différence notable est qu'on s'intéresse le plus souvent à des exécutions infinies (et donc à des mots infinis) puisque la plupart des systèmes temps-réels sont supposés fonctionner sans s'arrêter.

La figure 2.2 donne deux automates temporisés. Celui de gauche démarre dans son état initial Far (marqué en gras) ; la transition vers l'état Before se fait sans condition : elle émet une action *app*! et remet le chronomètre t à zéro. L'automate peut ensuite rester dans l'état Before tant que t ne dépasse pas la valeur 5. De plus, il ne peut exécuter la transition vers l'état Inside que si $t \geq 4$ et donc, cette transition doit avoir lieu dans un intervalle $[4, 5]$ après le franchissement de celle qui a remis t à zéro.

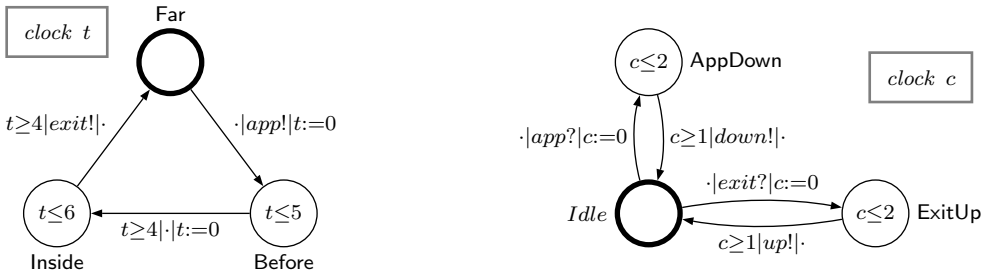


Figure 2.2 — À gauche, un automate temporisé à trois états utilisant un chronomètre t. À droite, un autre automate utilisant un chronomètre c.

L'automate à gauche de la figure 2.2 peut avoir une exécution correspondant au mot temporisé suivant :

$$(app!, z)(\cdot, z + 4{,}2)(exit!, z + 5{,}1)(app!, z + 42)(\cdot, z + 47)(exit!, z + 51) \cdots$$

où z est la date de la première transition (qui n'est pas forcément l'origine de l'axe du temps) et $\cdot$ représente une transition silencieuse. L'automate a donc attendu z unités de temps dans son état Far, puis il est passé dans l'état Before où il a séjourné 4,2 unités de temps. Il est ensuite passé à Inside et y est resté 0,9 unités de temps, etc. Par la suite,

l'automate a franchi la transition Before → Inside au plus tard et ensuite Inside → Far au plus tôt.

Afin de faciliter la décomposition de systèmes complexes, les actions sur les transitions peuvent être exploitées pour calculer le *produit synchronisé* de plusieurs automates. Cette opération crée un nouvel automate dont les exécutions correspondent aux entrelacements des exécutions de ses opérandes dans lesquelles les paires d'actions telles que *app*! et *app*? surviennent en même temps (et silencieusement). Les états de l'automate synchronisé sont formés par le produit cartésien des états des automates opérandes et les transitions sont calculées en fonction des états, en respectant la synchronisation des actions. La figure 2.3 montre le produit synchronisé des deux automates de la figure 2.2.

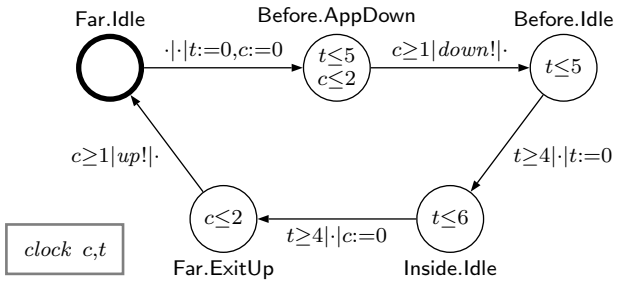


Figure 2.3 — Le produit synchronisé des automates de la figure 2.2.

Le produit synchronisé de plusieurs automates étant lui même un automate, séquentiel par nature, il devient clair que ce modèle propose une vision entrelacée du parallélisme. Lorsque plusieurs chronomètres sont présents dans un automate (comme dans celui de la figure 2.3), ils évoluent au même rythme.

Les automates temporisés sont analysés le plus souvent par vérification de formules de logiques temporelles [Alu+93] (voir aussi plus bas la section 2.1.4). Pour cela, on associe des prédicats aux états (par exemple, dans l'état Far.ExitUp de l'automate de la figure 2.3, les prédicats Far et ExitUp sont vrais) et on vérifie la formule sur toutes les exécutions possibles de l'automate. On ne peut pas le faire directement car l'échelle de temps étant continue, la plupart des transitions peuvent être datées d'une infinité de façons. Par exemple, la transition de Before à Inside de l'automate de gauche de la figure 2.2 peut (et doit) être franchie dans un intervalle de 4 à 5 unités de temps après que l'automate arrive dans l'état Before, or cet intervalle contient une infinité de points.

Afin de permettre la vérification d'une formule sur un automate, on calcule au préalable le *graphe des régions* de l'automate qui regroupe tous les états temporisés (c'est à dire les états de l'automate associés à des valeurs de chronomètres) qui permettent les mêmes évolutions. La figure 2.4 donne le graphe des régions de l'automate de gauche de la figure 2.2. La taille de ce graphe dépend exponentiellement du nombre de chronomètres utilisés par l'automate et du nombre de constantes distinctes auxquelles ils sont comparés. (C'est pour cette raison que nous n'avons pas présenté le graphe des régions de l'automate de la figure 2.3.)

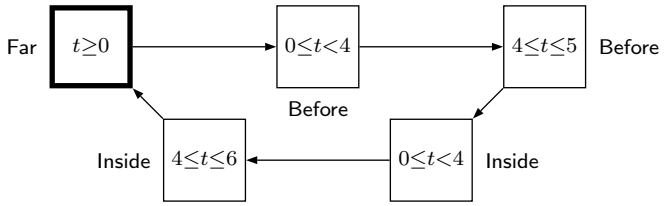


Figure 2.4 — Le graphe des régions correspondant à l'automate temporisé à gauche de la figure 2.2. Les nœuds sont étiquetés par les prédicats vrais et nous avons indiqué à l'intérieur les valeurs possibles du chronomètre t.

2.1.2 Algèbres de processus temporisées

Les algèbres de processus sont des modèles compositionnels par excellence : elles permettent de définir des expressions, représentant des processus, qui peuvent être composées arbitrairement (en parallèle, en séquence, etc.) et amenées à communiquer entre elles. Trois algèbres parmi les plus réputées sont CCS (*Calculus of Communicating Systems*) [Mil80, Mil83, Mil89], CSP (*Communicating Sequential Processes*) [Hoa89] et ACP (*Algebra of Communicating Processes*) [BK89]. (On peut aussi citer le π-calcul qui apporte le concept de mobilité [Mil$^+$92].)

Ces trois modèles ont été étendus de diverses façons afin d'incorporer le temps [AM96, BB91, Cor98, CN96, CZ91, FM95, Gor$^+$95, HR95, MT90, MT91, NS94, RR88, Yi90]. Alors que les modèles de départ sont reconnus comme des références, aucun consensus ne semble s'établir autour du choix d'un modèle temporisé. Nous nous limitons ici à présenter CCS et l'une de ses extensions temporisées appelée TCCS (*Timed CCS*) [MT90] ([Cor$^+$99] compare TCCS à trois autres extensions de CCS). Ce choix est guidé par les similitudes qui existent entre CCS et les modèles de réseaux de Petri que nous utilisons dans cette thèse (voir en particulier le chapitre 8).

L'élément de base d'un processus CCS est une action $a \in \mathbb{A} \uplus \{\tau\}$ où $\mathbb{A}$ contient pour chaque action a son action conjuguée $\overline{a}$ et τ représente une action silencieuse. On utilise aussi la constante 0 qui représente un processus ne pouvant exécuter aucune action. Le processus $\alpha.p$, où $\alpha \in \mathbb{A}$, se comporte comme p précédé de l'exécution de α. On note ce comportement par la transitions $\alpha.p \xrightarrow{\alpha} p$. Le processus $p + q$ se comporte comme p ou comme q, de façon non déterministe. La composition parallèle $p|q$ permet de faire évoluer les deux processus p et q en entrelaçant leurs actions ou en les faisant communiquer (silencieusement) par synchronisation des actions conjuguées ; ainsi, on a :

$$(p \xrightarrow{a} p') \wedge (q \xrightarrow{\overline{a}} q') \implies (p|q) \xrightarrow{\tau} (p'|q') \quad .$$

CCS comporte d'autres constructions qui permettent notamment la restriction d'actions (pour rendre les communications sur une action locales à un processus), le renommage d'actions ou la récursion.

TCCS ajoute des constructions directement liées au temps : $(t).p$ est le processus qui laisse passer $t \in \mathbb{N}$ unités de temps puis se comporte comme p ; $\delta.p$ peut attendre

arbitrairement longtemps avant de devenir p. Pour représenter le passage du temps, on ajoute des transitions spéciales ; par exemple on a : $(t).p \overset{t}{\rightsquigarrow} p$. On a ainsi un modèle de temps d'horloge et la sémantique est strictement séquentielle (le passage du temps et les actions sont entrelacées).

Comme son prédécesseur CCS, TCCS a l'avantage de la simplicité et de la compositionnalité. La modélisation d'un système est simple et intuitive et peut être faite de manière très structurée grâce aux différents opérateurs de composition. La structure algébrique des processus permet de leur appliquer des techniques de preuve dédiées, en particulier les méthodes dites « des cônes et foyers » [GS96, Zwa01] qui permettent de prouver la bisimilarité de deux processus sans avoir à calculer explicitement leurs évolutions.

2.1.3 Réseaux de Petri avec du temps

Les réseaux de Petri [Petri62, Pet81] ont connu un succès important par leur capacité à modéliser efficacement la concurrence. En effet, on peut y représenter explicitement l'indépendance des événements, la dépendance causale, le conflit sur une ressource, etc. De plus, les réseaux de Petri possèdent une notation graphique agréable qui supporte l'intuition lors de leur étude. Ils ont enfin une définition mathématique rigoureuse et de nombreuses techniques permettent de les analyser (voir notamment [Esp94, Mur89]).

Nous avons déjà présenté les réseaux de Petri à la section 1.2.3, aussi nous parlons directement de leurs extensions temporisées. Le temps peut être attaché à chacun des éléments d'un réseau de Petri : sur les places [CR83, Kha$^+$96, Kha$^+$96b, Kha97, Sif77, Sif79], sur les transitions [HV87, MSW90, Mer74, MF76, RH80, Ram74, Sta78], sur les arcs [BC89, Wal83] ou sur plusieurs de ces éléments [Cer99]. Une présentation complète des différentes approches est donnée dans [Boy01] où on trouve aussi quelques comparaisons intéressantes. Une étude formelle de l'expressivité de ces modèles est faite dans [Cer99] en se basant sur un cadre unificateur des différentes approches.

L'une des plus anciennes extensions (et l'une des plus expressives) définit les *time Petri nets* [Mer74, MF76] qui attachent à chaque transition un intervalle de temps durant lequel elle peut (et doit) être franchie. Le temps est compté par une horloge attachée à chaque transition à partir du moment où il y a assez de jetons pour permettre son franchissement. Les horloges d'un réseau sont toutes incrémentées simultanément, on a donc un modèle où le temps est global mais pris en compte localement.

La figure 2.5 donne deux exemples de réseaux de Petri temporisés qui modélisent les mêmes systèmes que les automates temporisés de la figure 2.2. Nous avons donné aux transitions des réseaux des étiquettes correspondant aux premières lettres des actions qui étiquetaient les transitions des automates (a pour *app*!, $\widehat{a}$ pour *app*?, et ainsi de suite pour *exit*, *down* et *up*).

Les réseaux de Petri temporisés souffrent des mêmes limitations que le modèle qu'ils étendent : ils restent de très bas niveau (ils ne peuvent pas représenter les données de manière directe) et n'autorisent pas la compositionnalité (ainsi, on ne peut pas faire communiquer les deux réseaux de la figure 2.5). Dans le modèle non temporisé, ces deux problèmes ont trouvé des solutions diverses. Le premier a été résolu par des versions

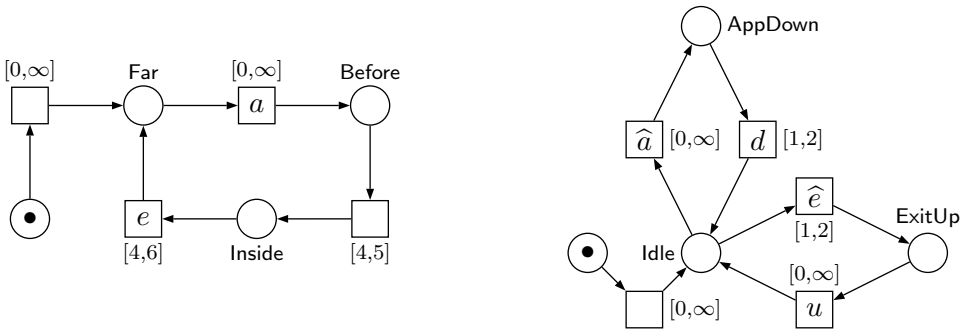


Figure 2.5 — Deux réseaux de Petri temporisés (N_1 à gauche et N_2 à droite). Nous avons ajouté des transitions permettant l'initialisation.

colorées autorisant différents types de jetons (voir section 1.2.4). Le second problème a été la motivation de départ pour la définition de nouveaux modèles de réseaux de Petri, en particulier *l'algèbre des Petri boxes* (voir chapitre 8) et celle des *M-nets* (chapitre 3). Dans ces modèles, on dispose de réseaux de Petri qui peuvent être composés, à la manières des termes des algèbres de processus, au moyen d'opérateurs mettant en avant la structure du contrôle et celle des communications. Ces modèles sont l'objet de recherches actives depuis maintenant une dizaine d'année et ont connu de nombreuses applications qui ont montré leur utilité.

Depuis 1999, plusieurs équipes travaillant sur ces modèles ont mis au point un programme de recherche visant à intégrer le temps dans l'algèbre des Petri boxes et celle des M-nets. Leurs travaux ont été fédérés au sein de projets bilatéraux (un projet franco-allemand, PORTA, et un projet anglo-allemand, BAT). D'autres groupes de recherche se sont intéressés au sujet, notamment en Belgique et en Espagne. Actuellement, tous ces groupes organisent des colloques réguliers pour présenter leurs travaux. Cela a permis des avancées substantielles dans le domaine : plusieurs approches ont été proposées, la plupart d'entre elles intégrant différentes extensions temporelles dans l'algèbre des Petri boxes. Cela résout le problème de la compositionnalité mais pas celui de la taille des réseaux modélisant des systèmes importants. C'est justement cette question qui nous intéresse dans cette thèse, où nous proposons une extension temporisée des M-nets.

Time Petri Box Calculus (tPBC). Une première approche [Kou00] étend PBC en utilisant les réseaux de Petri temporisés que nous avons présentés plus haut [Mer74, MF76] et en augmentant l'algèbre de processus correspondante avec des annotations exprimant les intervalles de temps durant lesquels les actions s'exécutent. Dans le modèle obtenu, baptisé *time Petri Box Calculus* (tPBC), les réseaux temporisés N_1 et N_2 présentés figure 2.5 peuvent être décrits respectivement par les expressions E_1 et E_2 ci-dessous :

$$E_1 \stackrel{\text{df}}{=} \Big[\tau[0,\infty] * (a[0,\infty]; \tau[4,5]; e[4,6]) * f\Big] \,\mathsf{sc}\, f$$

$$E_2 \stackrel{\text{df}}{=} \Big[\tau[0,\infty] * \Big((\widehat{a}[0,\infty]; d[1,2]) \,\square\, (\widehat{e}[0,\infty]; u[1,2])\Big) * f\Big] \,\mathsf{sc}\, f$$

On peut ensuite composer en parallèle ces expression et synchroniser les paires d'actions telles que a et $\widehat{a}$ pour forcer l'exécution simultanée des transitions leur correspondant :

$$E \stackrel{\text{df}}{=} (E_1 \| E_2)\,\mathsf{sc}\,\{a, e\} \quad .$$

La synchronisation prend soin de calculer les intervalles de franchissement adéquats. L'exécution suit alors la règle de franchissement habituelle des réseaux de Petri temporisés. Au niveau des expressions, l'exécution est définie par une *sémantique opérationnelle*, de manière similaire à ce que nous avons présenté pour TCCS (voir le chapitre 8 pour plus de détails). En ce qui concerne le passage du temps, une transition $\xrightarrow{\surd}$ correspond à la transition $\overset{1}{\rightsquigarrow}$ de TCCS. La sémantique opérationnelle est montrée cohérente avec la sémantique réseau.

Timed Petri Box Calculus (TPBC). Une approche similaire mais basée sur les *Timed Petri nets* [Ram74] est proposée dans [MF01]. Les auteurs étendent cette fois la syntaxe des actions sous la forme $a : d$ qui spécifie une action dont l'exécution *dure* au moins $d \in \mathbb{N}^+$ unités de temps. Ici, le temps est encore discret ; le début de l'exécution d'une action temporisée $a : d$ est marqué par une transition $\xrightarrow{\{a_t\}}$ qui indique que l'exécution durera $t \geq d$ unités de temps. L'évolution de cette action continue avec des transitions $\xrightarrow{\varnothing}$ dont chacune consomme une unité de temps.

Comme dans le modèle précédent, la sémantique opérationnelle est étendue pour tenir compte de l'ajout du temps et elle est prouvée cohérente avec la sémantique réseau.

Liens avec le temps causal. Suite à la première définition du temps causal dans les M-nets [1], une étude comparative des modèles de réseaux de Petri incluant le temps et de l'approche causale a débuté.

Dans [FP02], les auteurs définissent une version des M-nets dans laquelle le franchissement des transitions a une durée (comme dans l'extension de [Ram74]). Ce modèle est aussi doté d'une notion de *hiérarchie temporisée* permettant de raffiner des transitions (avec leurs durées) par des réseaux plus complexes (eux-mêmes temporisés). Cette hiérarchie est basée sur la substitution de transitions des M-nets (voir section 3.3.2) et l'étend pour prendre en compte la durée d'une transition hiérarchique. Les auteurs montrent ensuite que tout réseau de ce type peut être transformé en un M-net muni d'une horloge causale.

2.1.4 Logiques temporelles et logiques temporisées

Les logiques temporelles permettent d'exprimer des propriétés qualitatives sur le temps. On peut y parler notamment du futur, d'un instant suivant, précédent, etc. Bien qu'il ne s'agisse pas à proprement parler de modèles du temps, on peut utiliser les logiques temporelles pour représenter des systèmes par leurs propriétés. Les logiques CTL (*Computation Tree Logic*) [Cla$^+$96] et LTL (*Linear Time Logic*) [Pnu77] sont parmi les plus répandues. LTL permet de raisonner sur des suites linéaires d'états alors que CTL est interprété sur des structures d'arbres d'états représentant les branchements possibles

des exécutions. Nous limitons notre présentation à CTL, auquel nous ferons référence au chapitre 5.

L'aspect temporel de CTL est obtenu par les quantificateurs ; ainsi, $\forall\Box$ est interprété comme « pour tout état futur » et $\exists\Diamond$ comme « il existe un état accessible à partir de l'état courant ». Par exemple, la formule $\forall\Box(p \Rightarrow \exists\Diamond q)$ se lit : « Pour tout état futur, si p est vrai, alors il existe une exécution permettant d'atteindre un état où q est vrai. »

La plupart des logiques temporelles ont des extensions temporisées permettant d'exprimer quantitativement les liens avec le futur. Pour cela, on ajoute généralement des durées sur les quantificateurs. Par exemple, en TCTL [Alu+93], on peut écrire $\exists\Diamond_{<3}$ qui signifie « il existe un état accessible en moins de 3 unités de temps ».

Les logiques temporelles et leurs versions temporisées sont interprétées par des chemins sur des structures de Kripke [Kri71], c'est à dire des graphes dont les nœuds sont étiquetés par des propositions atomiques. On peut aussi se donner une structure de Kripke, et chercher à vérifier si une formule de logique temporelle est interprétable dessus : c'est la technique de *vérification de modèle* (*model checking*) [Cla+99]. Par exemple, la figure 2.6 donne une structure de Kripke pour laquelle on peut interpréter la formule $\forall\Box(p \Rightarrow \exists\Diamond q)$ (à gauche) et une autre pour laquelle on ne peut pas (à droite).

Figure 2.6 — Deux structures de Kripke. Les nœuds sont étiquetés par les propositions atomiques qui y sont vraies et les états courants correspondent aux nœuds en gras.

La vérification de modèle consiste donc à définir une structure de Kripke à partir d'une modélisation dans un formalisme, et à se donner une formule de logique temporelle définissant une propriété qu'on veut vérifier. On utilise ensuite un algorithme de vérification qui détermine si la formule est interprétable sur la structure. Par exemple, dans le cas des automates temporisés, on peut vérifier des formules CTL et TCTL sur les graphes des régions [Alu+93].

2.2 Modèles de la préemption

La préemption concerne le contrôle des tâches qui s'exécutent dans un système. Une tâche est dite *préemptible* si son exécution peut être interrompue sans que la tâche en soit prévenue (sinon, la tâche s'interrompt elle-même plus qu'elle n'est interrompue). Cette interruption peut être définitive, on parle alors d'*avortement*, ou temporaire et suivie d'un redémarrage au point d'interruption, on parle alors de *suspension/reprise*.

La notion de préemption est indépendante de celle de temps. Par exemple, une imprimante n'est généralement pas un système temporisé et pourtant, l'ouverture du bac à papier suspend l'impression. Par ailleurs, les imprimantes comportent le plus souvent un bouton d'annulation qui déclenche l'avortement du processus d'impression.

Malgré cette indépendance, les systèmes temps-réels font souvent appel à la préemption. Il peut en effet être nécessaire de suspendre une tâche afin d'assurer qu'une autre tâche respecte ses échéances. On peut aussi considérer que certaines tâches qui ne respectent pas leurs échéances sont en erreur et doivent être avortées.

Préemption impérative et facultative. On distingue habituellement deux formes de préemption. La préemption *impérative* est celle qu'on envisage intuitivement : elle prend effet aussitôt qu'elle est demandée (ce caractère immédiat dépend bien sûr du modèle d'exécution). La préemption *facultative* est moins stricte et autorise un délai entre la demande et la prise en compte de la préemption. Ce délai peut se révéler rédhibitoire, c'est pourquoi ce modèle de préemption n'est pas satisfaisant pour les systèmes temps-réel. Il se révèle pourtant utile en pratique ; par exemple, les systèmes d'exploitation de la famille Unix permettent l'avortement au moyen de la touche `Control+C` et la suspension avec `Control+Z`. Dans les deux cas, il s'agit de préemption facultative et le système y réagit « dès qu'il le peut ». Tout utilisateur d'Unix a fait la douloureuse expérience de l'aspect facultatif de cette préemption alors qu'il souhaitait interrompre une commande destructrice invoquée par erreur.

Pour prendre effet, la préemption impérative doit être prise en compte de manière prioritaire : le mécanisme de préemption doit toujours prendre le pas sur la partie du système qu'il doit interrompre. Pour cette raison, on ne peut imaginer la préemption impérative comme un service offert par les tâches. On pourrait sinon envisager des tâches peu coopératives qui refuseraient d'être avortées ou suspendues. La préemption impérative passe donc nécessairement par un mécanisme externe aux tâches qui peut les manipuler dans leur globalité.

Préemption et ordonnancement. Différents modèles de la préemption l'envisagent sous l'angle de l'ordonnancement [Corb94, GL94, Här+94] : étant donné un ensemble de tâches s'exécutant sur un ou plusieurs processeurs, on doit être capable d'exécuter une tâche prioritaire à tout moment, y compris si une autre tâche est déjà en exécution.

Ces modèles considèrent les tâches comme des processus séquentiels parmi lesquels il faut élire celui ou ceux qui peuvent s'exécuter à un instant donné. Dans l'algèbre de processus CCSR [GL94] par exemple, la préemption correspond à élire une branche de l'opérateur de choix en fonction des priorités des tâches, de leurs synchronisations et des ressources qu'elles utilisent.

Préemption dans les langages synchrones. Dans les langages synchrones [Mar89, Ber98, Benv+91, Cas+87, Hal+91], la préemption est un aspect central et elle est définie de façon primitive au même titre que les autres constructions comme le contrôle de flot ([Ber93] justifie ce choix en détail). Esterel [Ber98] est un langage de contrôle synchrone qui permet de programmer des systèmes réactifs. Il propose la préemption sous la forme d'une construction d'avortement et d'exceptions.

L'avortement permet de terminer une activité lorsqu'une condition devient vraie. Par exemple, l'extrait de programme suivant attend un signal *sig*, cette attente est interrom-

pue si le signal *tic* est reçu 15 fois avant que *sig* ne le soit :

abort
  await *sig*
when 15 *tic*

Les exceptions permettent de sortir d'un bloc de programme en déclenchant un traitement adéquat. Elles complètent en quelque sorte l'avortement : celui-ci permet en effet d'interrompre un bloc (compris entre **abort** et **when**) à partir d'un événement provenant de l'extérieur du bloc (15 occurrences de *tic* dans l'exemple) alors que les exceptions permettent l'interruption d'un bloc à partir d'un événement provenant du bloc lui-même (l'exception qui y est levée).

La sémantique des langages synchrones est purement séquentielle et déterministe ce qui permet un traitement simple et efficace de la préemption. Les travaux autour de ces langages ont permis de formaliser et de préciser la notion de préemption telle que nous l'avons présenté ci-dessus. En particulier, dans [Pin+98], les auteurs proposent une approche de la préemption qui leur permet de caractériser précisément cette notion à un niveau syntaxique.

Préemption dans un cadre asynchrone et concurrent. La représentation de la préemption devient plus difficile lorsqu'on sort du cadre synchrone et séquentiel des modèles que nous avons évoqué jusque là. En effet, la préemption requiert une vue globale des tâches à interrompre ainsi qu'une dépendance entre celles-ci et le mécanisme de préemption ; au contraire les sémantiques concurrentes reposent en général sur la localité des observations et l'indépendance de certaines parties des systèmes.

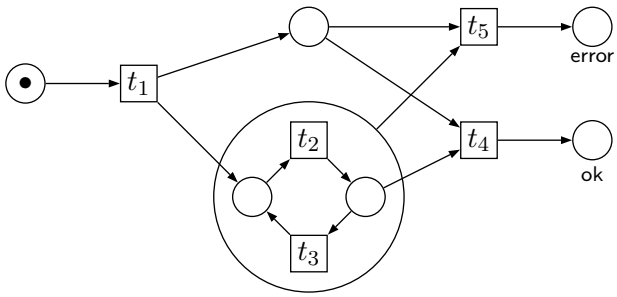


Figure 2.7 — Un place chart net comportant une place hiérarchique. La place ok est marquée lorsque le système termine normalement ; si un avortement a lieu, c'est la place error qui est marquée.

Le modèle des PCN (*Place Charts Nets*) [Kis+97] résout ce conflit en proposant une hiérarchie des systèmes reposant directement sur la préemption (qui est proposée ici sous forme d'avortement uniquement). Le modèle se base sur les réseaux de Petri et induit des dépendances entre les sous-réseaux avortables et les transitions pouvant les avorter. La figure 2.7 donne un exemple de tel système. Son exécution débute par une transition d'initialisation t_1. On peut ensuite franchir indéfiniment les transitions t_2

et t_3 en alternance. Après t_2, le franchissement de t_4 correspond à la terminaison du système. La transition t_5 possède un arc entrant venant d'une place hiérarchique qui englobe les transitions t_2, t_3 et les places qu'elles utilisent. Le sous-réseau ainsi englobé est avortable par la transition t_5 qui est franchissable lorsqu'un jeton est présent dans n'importe laquelle des places du sous-réseau (il faut aussi un jeton dans sa place d'entrée non hiérarchique). Le franchissement de t_5 consomme alors tous les jetons présents dans le sous-réseau avortable.

Le modèle des PCN est une généralisation non triviale des réseaux de Petri places/transitions :

- dans le cas fini (borné), on peut transformer un PCN en réseau de Petri places/transitions dont la taille est en général exponentiellement plus grande ;
- dans le cas infini (non borné), la classe de langages reconnus par les PCN (en étiquetant les transitions avec des lettres) est strictement plus grande que la classe de langages reconnus avec des réseaux P/T ;
- on peut décider si un PCN est borné.

Remarquons que le modèle des PCN ne propose pas de représentation de la suspension/reprise et que l'avortement y est facultatif (et non impératif). Remarquons encore que la hiérarchie dans les PCN est entièrement guidée par la structure de préemption, ce qui diffère considérablement de la composabilité des modèles que nous étudions dans cette thèse.

3 M-nets : une classe de réseaux colorés composables

Ce chapitre a pour objectif d'introduire le modèle des M-nets. Il s'agit d'une classe de réseaux de Petri colorés étiquetés qui sont composables au moyen d'opérateurs similaires à ceux des algèbres de processus. La définition originale des M-nets a été donnée dans [Kla95], comme une restriction à des types énumérés de réseaux utilisant des types abstraits algébriques ; cette définition a été finalisée dans [Bes+98]. Ce modèle étant basé sur des réseaux colorés, il autorise plus directement l'utilisation d'outils logiciels ; l'outil PEP [GB96, Gra97] intègre les M-nets et permet notamment de les éditer et de leur appliquer divers algorithmes de vérification (en particulier du *model-checking*). Les M-nets ont été le point de départ de nombreux travaux, ils ont en particulier servi à donner des sémantiques à des langages de programmation [Bes+95, Bes+98, Lil96, LP96, FG97, FG98], mais aussi dans d'autres domaines [Ben+98, Bes97]. Le modèle a par la suite connu plusieurs extensions [Dev+97, Dev+02] ; dont les plus récentes [1, 3, 5] sont présentées dans ce mémoire.

Pour définir les M-nets, nous commençons par rappeler les notions de base qui seront utilisées ensuite. On continue par la définition formelle des réseaux de Petri colorés qui nous serviront pour aboutir à celle des M-nets.

3.1 Définitions de base et notations

Multi-ensembles. Un multi-ensemble μ sur un ensemble E est une fonction de E dans l'ensemble des entiers naturels qui associe à chaque $e \in E$ son nombre d'occurrences dans le multi-ensemble. Un multi-ensemble est fini si l'ensemble $\{e \in E \mid \mu(e) > 0\}$ l'est aussi ; nous notons $\mathsf{mult}(E)$ l'ensemble des multi-ensembles finis sur E. Un sous-ensemble de E est souvent pris comme un multi-ensemble en l'identifiant à sa fonction caractéristique. Le multi-ensemble vide est noté $\varnothing$, il s'agit de la fonction constante nulle.

Les multi-ensembles pouvant être vus comme des ensembles avec répétition, nous notons $x \in \mu$ pour $(x \in E) \wedge (\mu(x) > 0)$. De même, pour les multi-ensembles finis, nous utilisons une notation d'ensembles étendus ; par exemple, $\{a, b, b\}$ représente le multi-ensemble μ sur $E \subseteq \{a, b\}$ tel que $\mu(a) = 1$, $\mu(b) = 2$ et $\mu(x) = 0$ pour $x \in E \setminus \{a, b\}$.

Étant donnés μ_1 et μ_2 deux multi-ensembles sur des ensembles E_1 et E_2 respectivement, nous notons $\mu_1 \leq \mu_2$ quand $E_1 \subseteq E_2$ et $\mu_1(x) \leq \mu_2(x)$ pour tout $x \in E_1$. De plus, nous notons par $+$ et $-$ la somme et la différence de multi-ensembles, qui sont des multi-ensembles sur $E_1 \cup E_2$ définis respectivement par :

$$(\mu_1 + \mu_2)(x) = \begin{cases} \mu_1(x) & \text{si } x \in E_1 \text{ et } x \notin E_2, \\ \mu_2(x) & \text{si } x \notin E_1 \text{ et } x \in E_2, \\ \mu_1(x) + \mu_2(x) & \text{sinon ;} \end{cases}$$

$$(\mu_1 - \mu_2)(x) = \begin{cases} \mu_1(x) & \text{si } x \in E_1 \text{ et } x \notin E_2, \\ 0 & \text{si } x \notin E_1 \text{ et } x \in E_2, \\ \max(0, \mu_1(x) - \mu_2(x)) & \text{sinon.} \end{cases}$$

Finalement, nous utilisons l'opérateur $*$ pour construire un multi-ensemble sur un singleton. Par exemple, $1 * a + 2 * b$ (aussi noté $a + 2 * b$) est le multi-ensemble $\{a, b, b\}$. On a donc en particulier, pour tout multi-ensemble μ sur un ensemble E :

$$\mu = \sum_{x \in E} \mu(x) * x \quad .$$

Cette construction est étendue sur les ensembles de plus d'un élément de la façon suivante : si E est un ensemble et k un entier alors $k * E$ est le multi-ensemble sur E qui vaut $\sum_{x \in E} k * x$.

Ensembles distingués. Nous utilisons dans la suite quelques ensembles deux à deux disjoints.

L'ensemble *Val* contient les *valeurs* manipulées par les M-nets ; il contient en particulier le jeton noir ($\bullet$), des entiers (0, 1, 2, ...), des booléens (vrai et faux, notés respectivement $\top$ et $\bot$) et d'autres constantes qui nous seront utiles. Suivant les cas, cet ensemble peut être fini ou non ; si rien n'est précisé, il est considéré fini. Dans les deux cas, nous utilisons la constante ω, plus grande que tout entier et pour laquelle nous posons $\omega + 1 = \omega$.

L'ensemble *Var* est un ensemble de *variables* qui sont utilisées dans les annotations des M-nets. Nous considérons que l'ensemble *Var* est infini afin de toujours pouvoir en extraire un élément nouveau (non encore utilisé). Les variables sont généralement notées par les lettres x, y et z.

Nous nous donnons aussi un ensemble $\mathbb{A}$ de *symboles d'actions*, utilisés pour les communications synchrones dans les M-nets. Cet ensemble est doté d'une fonction de conjugaison qui à tout $a \in \mathbb{A}$ associe $\widehat{a} \in \mathbb{A}$, telle que $\widehat{\widehat{a}} = a$ pour tout a (la conjugaison est donc une involution sur $\mathbb{A}$). De plus, à chaque symbole d'action a est associée son arité, l'entier $\mathsf{ar}(a) \geq 0$, indiquant le nombres de paramètres transmis lors des communication impliquant a et $\widehat{a}$. Les symboles d'actions sont généralement notés par la lettre a ou par des mots complets.

L'ensemble $\mathbb{B}$ contient les *symboles de liens* qui sont utilisés pour les communications asynchrones dans les M-nets. Les symboles de liens sont généralement notés par la lettre b.

Dans $\mathbb{X}$, se trouvent les *actions hiérarchiques* qui représentent des actions non élémentaires d'un réseau. Elles sont destinées à être remplacées par des actions plus complexes au moyen de substitution de transitions par des réseaux. Nous notons en général par $\mathcal{X}$ les éléments de $\mathbb{X}$.

Finalement, l'ensemble $\mathbb{S} = \{\mathsf{e}, \mathsf{i}, \mathsf{x}, \mathsf{b}\}$ contient les *statuts de places* qui sont utilisés notamment pour la composition des M-nets par les opérateurs de contrôle de flot (par exemple, la composition en séquence).

Dans les définitions de ce chapitre et des suivants, nous utilisons des éléments particuliers de ces ensembles, par exemple, l'action hiérarchique $\mathcal{X}_1^i$ (utilisé figure 3.8) ou le symbole d'action *clock* (utilisé section 4.1.1 et suivantes). Afin de garantir la validité des définitions, nous supposons que ces éléments sont réservés aux définitions y faisant appel.

3.2 Réseaux de Petri colorés et sémantique concurrente

Un réseau de Petri coloré [Jen81, Jen92] étiqueté est un triplet (S, T, ι) où S est l'ensemble de ses places, T celui de ses transitions (avec $S \cap T = \varnothing$) et ι associe une inscription aux places, aux transitions et aux arcs (les éléments de $(S \times T) \cup (T \times S)$).

L'étiquette d'une place $s \in S$ est son type $\iota(s) \subseteq \mathit{Val}$ qui correspond à l'ensemble des valeurs de jetons qu'elle peut contenir.

L'étiquette d'une transition $t \in T$ est sa *garde* $\iota(t)$; il s'agit d'un prédicat booléen construit sur les valeurs de *Val*, les variables de *Var* et les opérateurs de l'arithmétique et de la logique ($+$, $-$, $=$, $\geq$, $\wedge$, $\vee$, $\neg$, ...). La garde sert de condition de franchissement aux transitions.

L'étiquette d'un arc $(s, t) \in S \times T$ ou $(t, s) \in T \times S$, noté $\iota(s, t)$ ou $\iota(t, s)$ (en omettant une paire de parenthèses), indique ce qui est transporté sur cet arc lors du franchissement de la transition adjacente. Cette annotation est un multi-ensemble de valeurs ou d'expressions (en général de simples variables) qui doivent être évaluables en assignant des valeurs aux variables qu'elles contiennent.

Étant donné un réseau $N = (S, T, \iota)$, nous notons ${}^\bullet r$ l'ensemble des nœuds précédant immédiatement r dans le réseau : ${}^\bullet r = \{r' \in S \cup T \mid \iota(r', r) \neq \varnothing\}$. De manière similaire, $r^\bullet = \{r' \in S \cup T \mid \iota(r, r') \neq \varnothing\}$.

Représentation graphique. Nous utilisons pour représenter les réseaux de Petri la notation usuelle sous forme de graphe bipartis : les places sont représentés par des cercles, les transitions par des carrés (ou des rectangles) et les arcs par des flèches entre les éléments qu'ils relient. Afin d'alléger les figures, nous utilisons en outre les simplifications suivantes :

- le type des places n'est indiqué que s'il diffère de $\{\bullet\}$;
- une garde de transition toujours vraie est omise ;
- un arc d'annotation vide n'est pas dessiné ;
- une annotation $\{\bullet\}$ sur un arc n'est pas indiquée ;
- les accolades des ensembles et multi-ensembles sont souvent supprimées ;

- le nom des places et des transitions n'est donné que lorsqu'ils sont utilisés dans le texte ;
- un arc portant une flèche à chaque extrémité symbolise deux arcs de directions opposées ayant la même annotation ;
- les gardes sont données comme une énumération de prédicats (plutôt que comme une conjonction).

3.2.1 Marquages et franchissements

Considérons un réseau de Petri coloré $N = (S, T, \iota)$. Un marquage de N est une fonction qui associe à chaque place $s \in S$ un multi-ensemble des valeurs de son type (un élément de $\mathsf{mult}(\iota(s))$). Pour comparer des marquages, on se restreint à l'intersection de leurs domaines ; par exemple, $M_1 = M_2$ si pour toute place $s \in (\mathrm{dom}(M_1) \cap \mathrm{dom}(M_2))$, on a $M_1(s) = M_2(s)$. Un réseau marqué est donc un couple (N, M) où M est un marquage de N appelé son marquage *initial* ; par abus de notation, nous noterons aussi (S, T, ι, M) le réseau $N = (S, T, \iota)$ marqué avec M. Dans la suite, le marquage sera le plus souvent sous-entendu et nous pourrons parler d'un « réseau marqué N », le marquage sous-entendu sera implicitement noté M_N.

Pour une transition $t \in T$, nous notons $\mathsf{var}(t)$ l'ensemble des variables qui apparaissent dans l'annotation de t et de ses arcs adjacents. Une *valuation* d'une transition t est alors une fonction de $\mathsf{var}(t)$ dans *Val* qui associe une valeur à chaque variable liée à t. Une valuation permet de remplacer les variables par des valeurs de manière à permettre *l'évaluation* qui calcule la valeur effective d'un objet. Par exemple, si $\sigma = \{x \mapsto 1\}$ est une valuation, on a $\sigma(x + 2) = 1 + 2$ (x a été substitué par 1), cette expression peut ensuite être évaluée. Par extension, nous notons de la même façon la valuation d'un objet et son évaluation (cela revient à considérer que l'évaluation est automatique dès qu'elle est possible), par exemple, nous écrirons $\sigma(x + 2) = 3$ (valuation de x plus évaluation de l'expression) si $\sigma(x) = 1$ (valuation de x). Dans le texte, en cohérence avec cette notation, nous dirons qu'un objet est évalué par une valuation. Cette assimilation de valuation et d'évaluation ne prête pas à confusion.

Une transition t est franchissable pour une valuation σ et un marquage M, ce qui est noté $M[\sigma(t)\rangle$, si les conditions suivantes sont réunies :

1. Les places en entrée de t contiennent assez de jetons pour satisfaire le flot indiqué par les arcs : pour toute place $s \in S$, $\sigma(\iota(s, t)) \leq M(s)$.

2. La garde de t s'évalue à vrai par σ : $\sigma(\iota(t)) = \top$.

3. Les annotations des arcs sortants de t s'évaluent en respectant le type des places en sortie de t : pour toute place $s \in S$, $\sigma(\iota(t, s)) \leq \iota(s)$.

Lorsque $M[\sigma(t)\rangle$, le franchissement de t conduit au marquage M' défini pour chaque place $s \in S$ par $M'(s) \stackrel{\mathrm{df}}{=} M(s) - \sigma(\iota(s, t)) + \sigma(\iota(t, s))$. Nous notons alors $M[\sigma(t)\rangle M'$ ce franchissement.

Une valuation σ d'une transition t est *légale* s'il existe un marquage M tel que $M[\sigma(t)\rangle$. Cela correspond aux conditions 2 et 3 données ci-dessus.

Un marquage M_k est *accessible* à partir d'un marquage M_0 s'il existe une séquence de transitions valuées $\sigma_1(t_1), \ldots, \sigma_k(t_k)$ et des marquages $M_1, \ldots, M_{k-1}$ tels que $M_0[\sigma_1(t_1)\rangle M_1 \cdots M_{k-1}[\sigma_k(t_k)\rangle M_k$.

La figure 3.1 présente un réseau coloré avec quatre marquages différents. Dans le marquage $M_\bullet$, la transition t est franchissable pour les valuations $\sigma_1 \stackrel{\text{df}}{=} \{x \mapsto 1\}$, $\sigma_2 \stackrel{\text{df}}{=} \{x \mapsto 2\}$ et $\sigma_3 \stackrel{\text{df}}{=} \{x \mapsto 3\}$. Les trois conditions de franchissement sont en effet vérifiées : (1) s_1 contient un jeton, (2) la garde est vraie et (3) $\sigma_i(x) \in \{1, 2, 3\}$ pour $1 \leq i \leq 3$. Dans ce cas, x est en fait une *variable libre* et seul le type de s_2 permet de choisir les valuations qui rendent t_1 franchissable. Le franchissement de t sous la valuation σ_3 produit le marquage M_3.

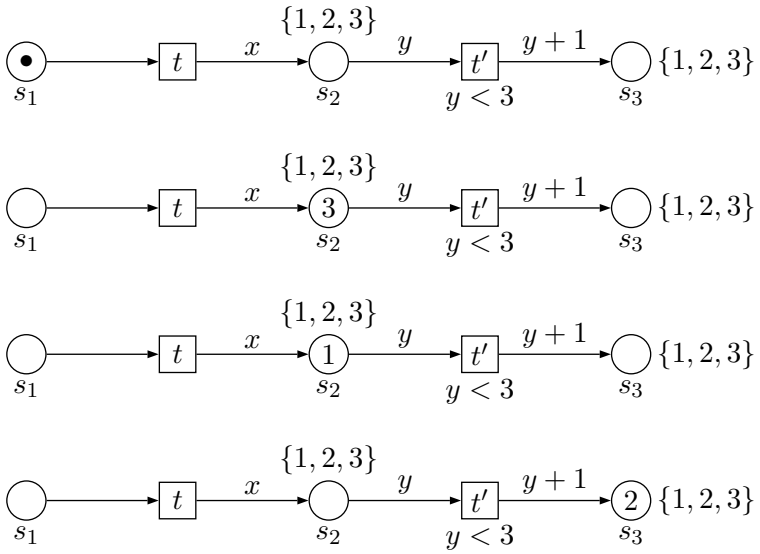


Figure 3.1 — Le réseau de Petri coloré N pour quatre différents marquages, de haut en bas : $M_\bullet$, M_3, M_1 et M_2.

Lorsque le réseau est dans le marquage M_3, aucune valuation ne permet de franchir t'. En effet, à cause de la condition de franchissement (1), la variable y maintenant nécessairement *liée* à la valeur 3 dans s_2 ce qui empêche la garde de t' d'être vraie.

Dans le marquage $M_\bullet$, on peut aussi franchir t avec la valuation σ_1. On atteint alors le marquage M_1 qui permet le franchissement de t' avec la même valuation $\sigma_1' \stackrel{\text{df}}{=} \{y \mapsto 1\}$ (puisque la garde valuée est $1 < 2$ qui s'évalue à vrai). Ce franchissement produit le marquage M_2 (puisque l'annotation d'arc $y + 1$ peut s'évaluer à 2 par σ_1'). On pourrait aussi franchir t' avec une valuation $\sigma_2' \stackrel{\text{df}}{=} \{y \mapsto 2\}$, pour cela, il faudrait au préalable franchir t sous σ_2 afin de produire le jeton 2 dans s_2.

3.2.2 Sémantique concurrente

La règle de franchissement donnée plus haut permet d'exécuter une transition à la fois et conduit à la *sémantique d'entrelacement* des réseaux de Petri dans laquelle les exécutions

sont des suites de transitions valuées. Cette sémantique peut être généralisée en autorisant un nombre arbitraire de transitions à s'exécuter en concurrence, nous obtenons alors la *sémantique steps* qui donne les suites de *steps* (des multi-ensembles de transitions valuées) que peut exécuter un réseau marqué [Gen+80, BD87, Bes96, BF88, GR83].

Considérons un réseau coloré marqué $N = (S, T, \iota, M)$. Un step du réseau marqué (N, M) est un multi-ensemble $U = \{\sigma_1(t_1), \ldots, \sigma_k(t_k)\}$, où chaque t_i est une transition dans T et σ_i est une valuation de t_i ($i \leq k$). Le step U est franchissable, ce qui est noté $M[U\rangle$, si les conditions suivantes sont réunies :

1. Chaque transition est franchissable : $\forall i \leq k, M[\sigma_i(t_i)\rangle$.
2. Le marquage est suffisant pour autoriser tous les franchissements simultanément :

$$\forall s \in S, M(s) \geq \sum_{i \leq k} \sigma_i(\iota(s, t_i)) \quad .$$

Le franchissement de U conduit à un nouveau marquage M' (nous notons alors $M[U\rangle M'$) défini pour chaque place $s \in S$ par :

$$M'(s) \stackrel{\text{df}}{=} M(s) - \sum_{i \leq k} \sigma(\iota(s, t_i)) + \sum_{i \leq k} \sigma(\iota(t_i, s)) \quad .$$

Une *linéarisation* d'un step U est une séquence de steps $U_1 \ldots U_k$ telle que $U = U_1 + \cdots + U_k$. Il s'agit bien ici d'une somme de multi-ensembles, une linéarisation peut donc être vue comme un éclatement d'un step. Nous pouvons facilement montrer que si $M[U\rangle M'$ alors il existe des marquages M_0, ..., M_k tels que $M_0 = M$, $M_k = M'$ et $M_i[U_i\rangle M_{i+1}$ pour $0 \leq i < k$.

La *sémantique concurrente* (ou *sémantique steps*) d'un réseau marqué N est l'ensemble $\mathsf{steps}(N)$ des séquences de steps franchissables dans N à partir de son marquage initial. Nous pouvons facilement montrer que la sémantique steps d'un réseau est stable par linéarisation : en remplaçant dans une séquence de $\mathsf{steps}(N)$ un step par n'importe laquelle de ses linéarisations, nous obtenons une nouvelle séquence qui est aussi dans $\mathsf{steps}(N)$.

Autres sémantiques concurrentes. Il existe d'autres sémantiques concurrentes des réseaux de Petri [Grab81, Sta81, Vog91, Vog92, Maz86]. En particulier, on peut construire des *réseaux d'occurrences généralisés* (*branching processes*) ; il s'agit de réseaux de Petri sans cycle dont les exécutions sont équivalentes à celle des réseaux qu'ils représentent [McM95]. La figure 3.2 donne un exemple de cette sémantique.

L'exécution du réseau d'occurrences à droite de la figure 3.2 correspond à celle du réseau de gauche, les différentes occurrences de chaque transition y étant distinguées (ce qui explique qu'on retrouve plusieurs fois certaines transitions). De même, la présence d'un jeton dans une place du réseau de gauche correspond au marquage de l'une des places portant le même nom dans le réseau d'occurrences ; ceci permet de différencier les jetons de la même place issus d'origines différentes.

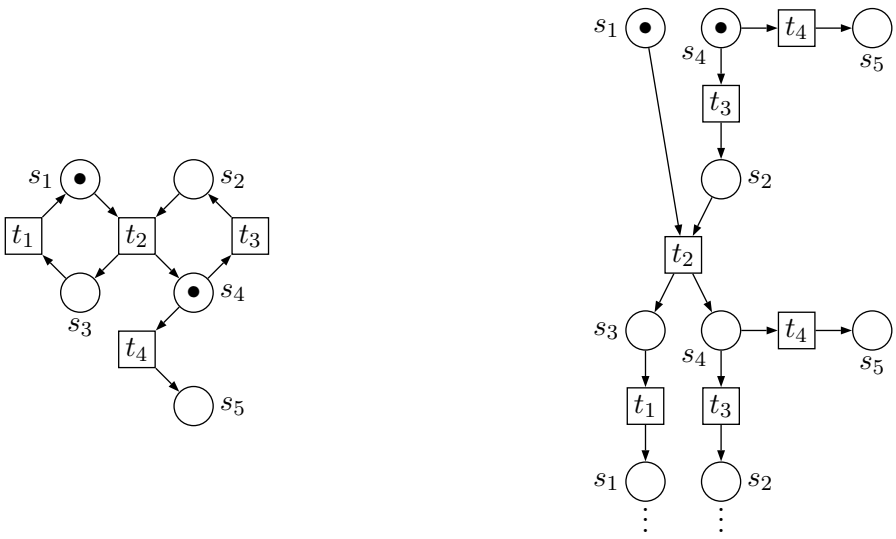


Figure 3.2 — À gauche un réseau de Petri et à droite un préfixe de son réseau d'occurrences généralisé.

Les avantages de ce type de sémantique sont multiples, nous n'en retiendrons que deux. Premièrement, comme nous venons de le voir, cela permet une connaissance très fine de l'exécution d'un réseau. Deuxièmement, et c'est un avantage considérable, on peut généralement borner le réseau à un préfixe de taille finie, et c'est sur ce préfixe fini qu'on utilise les algorithmes de vérification qui se révèlent généralement plus efficaces que sur le réseau de départ. Les préfixes finis sont en effet une représentation très compacte de la sémantique d'un réseau de Petri et donnent accès à beaucoup d'informations comme l'ordre partiel des occurrences de transitions. Un réseau d'occurrences est donc un objet à la fois simple et riche sur lequel on peut décider de nombreuses propriétés efficacement [Esp94].

Nous n'utilisons pas ce type de sémantique dans ce mémoire. La raison principale est que la sémantique d'exécution des réseaux n'est pas une question centrale de cette thèse. D'autre part, les réseaux d'occurrences ne peuvent être utilisés que dans le cadre de réseaux places/transitions. Des travaux très récents donnent des pistes pour utiliser ce genre de sémantique sur des réseaux colorés [Kou+02, FP00] et pourront certainement être utilisés dans des travaux futurs. Pour le moment, nous préférons nous en tenir à la sémantique steps qui est beaucoup plus intuitive que la sémantique par réseaux d'occurrences.

3.2.3 Dépliage

Nous montrons maintenant comment les réseaux colorés abrègent les réseaux places/transitions : pour tout réseau coloré, nous pouvons obtenir un réseau P/T équivalent. Cette opération est appelée *dépliage* et peut être utilisée pour construire des réseaux

d'occurrences de réseaux colorés dépliés.

Un réseau P/T peut être vu comme un réseau coloré en prenant $Val = \{\bullet\}$, $Var = \varnothing$ et en considérant toutes les gardes vraies. Le type de chaque place est donc $\{\bullet\}$, les annotations d'arcs correspondent alors à un nombre de jetons transportés par l'arc. Compte tenu de ces simplifications, nous remplaçons la fonction d'annotation ι par une *fonction de poids* W associant à chaque arc le nombre de jetons qu'il transporte. De même, un marquage peut être vu comme un multi-ensemble de places, chacune contenant autant de jetons qu'elle a d'occurrences dans ce multi-ensemble.

Soit $N = (S, T, \iota)$ un réseau de Petri coloré. Son dépliage est le réseau de Petri places/transitions $\mathsf{unf}(N) = (\mathsf{unf}(S), \mathsf{unf}(T), W)$ donné par :

- $\mathsf{unf}(S) \stackrel{\text{df}}{=} \biguplus_{s \in S} \mathsf{unf}(s)$, où, pour toute place $s \in S$, $\mathsf{unf}(s) \stackrel{\text{df}}{=} \{s_v \mid v \in \iota(s)\}$;
- $\mathsf{unf}(T) \stackrel{\text{df}}{=} \biguplus_{t \in T} \mathsf{unf}(t)$, où, pour toute transition $t \in T$, $\mathsf{unf}(t) \stackrel{\text{df}}{=} \{t_\sigma \mid \sigma \text{ est une valuation légale de } t\}$;
- la fonction de poids W est telle que, pour toute place $s_v \in \mathsf{unf}(S)$ et toute transition $t_\sigma \in \mathsf{unf}(T)$, $W(s_v, t_\sigma) \stackrel{\text{df}}{=} \sigma(\iota(s,t))(v)$ (chaque arc de t_σ vers s_v a pour poids le nombre de valeurs v transportées de t vers s sous la valuation σ) et nous définissons $W(t_\sigma, s_v)$ de manière similaire.

Le dépliage d'un marquage M de N est défini comme :

$$\mathsf{unf}(M) \stackrel{\text{df}}{=} \sum_{s \in S, v \in Val} M(s)(v) * s_v \quad .$$

Autrement dit, chaque place s_v dans le dépliage contient autant de jetons que la place s du réseau de départ contient d'occurrences de la valeur v.

Remarquons que si une place d'un réseau coloré possède un type infini, le dépliage produira une infinité de places pour la représenter. Si de plus une transition peut consommer ou produire un jeton arbitraire dans cette place, elle aura une infinité de valuations légales et le dépliage produira pour la représenter une infinité de transitions et d'arcs. De tels réseaux ne peuvent donc pas être dépliés en pratique et doivent être appréhendés de manière symbolique.

3.3 L'algèbre des M-nets

Afin de passer des réseaux colorés aux M-nets, nous commençons par ajouter des annotations. Un M-net est un réseau coloré étiqueté (S, T, ι) où ι est une fonction d'inscription sur les éléments du réseau, plus complexe que celle considérée jusque-là.

L'étiquette d'une place $s \in S$ est $\iota(s) \stackrel{\text{df}}{=} \lambda(s).\tau(s)$ où $\tau(s) \subseteq Val$ est le type de la place et $\lambda(s) \in \mathbb{S} \uplus \mathbb{B}$ est son *statut* indiquant la nature de cette place. Si le statut est e, la place est une *place d'entrée* ; un statut x indique une *place de sortie* (le x venant du mot *exit*) ; le statut i est celui des *places internes*. Ces trois types de places sont appelées *places de contrôle de flot* (ou simplement *places de contrôle*) puisque leur marquage reflète l'activité du M-net. Les autres étiquettes (dans $\{\mathsf{b}\} \uplus \mathbb{B}$; les lettres b viennent du mot *buffer*) indiquent des places dédiées aux communications asynchrones entre transitions,

nous les appelons les *places de liens*. Une étiquette b correspond à une *place de liens fermée*, c'est-à-dire une place de liens qui sert à des communications privées. Les autres places de liens sont *ouvertes* et on peut y ajouter des liens vers de nouvelles transitions.

Une transition $t \in T$ peut porter deux types d'étiquettes. S'il s'agit d'une *transition de communication*, c'est-à-dire un événement potentiellement communiquant, elle a l'étiquette $\iota(t) \stackrel{\text{df}}{=} \alpha(t).\gamma(t)$ où $\gamma(t)$ est la garde de la transitions et $\alpha(t)$ est un multi-ensemble *d'actions* permettant les communications synchrones. Une action est de la forme $a(x_1, \ldots, x_k)$ où $a \in \mathbb{A}$ est un symbole d'action d'arité $\mathsf{ar}(t) = k$ (si $k = 0$, les parenthèses sont omises) et chaque x_i est une valeur (dans *Val*) ou une variable (dans *Var*). Une *transition hiérarchique* est étiquetée par $\iota(t) \in \mathbb{X}$, elle est destinée à être substituée par un M-net et correspond donc à une « méta-action ».

Sur les arcs, le modèle des M-nets utilise des annotations qui peuvent être très complexes [Dev+02]. Il n'est pas utile de les présenter dans ce mémoire ; il nous suffit de savoir que, dans tous les cas, le franchissement d'une transition entraîne l'évaluation des annotations d'arcs en éléments de $\mathsf{mult}(\mathit{Val})$. Les seuls annotations que nous utiliserons seront donc des multi-ensembles de valeurs et d'expressions (souvent de simples variables), comme pour les réseaux colorés présentés auparavant.

Restrictions syntaxiques. Pour qu'un M-net soit syntaxiquement correct, il doit respecter certaines restrictions par rapport aux possibilités décrites ci-dessus. Ces restrictions ont pour but d'assurer de bonnes propriétés au modèle.

1. Un M-net doit être *ex-restreint*, c'est-à-dire posséder au moins une place d'entrée (de statut e) et une place de sortie (de statut x) de types non vides.

2. Il doit être *ex-orienté*, ce qui signifie que les arcs aboutissant à des places d'entrée ou partant de places de sortie sont interdits.

3. Il doit être *T-restreint*, ce qui correspond à forcer chaque transition à avoir au moins un arc entrant venant d'une place de contrôle de type non vide et un arc sortant allant vers une telle place.

4. Enfin, un M-net doit être *B-restreint*, c'est à dire qu'il doit posséder, pour chaque $b \in \mathbb{B}$, une et une seule place de statut b, et qu'il n'existe pas d'arc entre une transition hiérarchique et une place de liens.

Un M-net qui respecte toutes ces contraintes est dit *bien formé*.

Le rôle des deux premières contraintes est de permettre la compositions de M-nets par les opérateurs de contrôle de flot. La T-restriction permet d'assurer certaines propriétés des marquages (voir par exemple le théorème 3). Quant à la B-restriction, elle sert principalement à simplifier les définitions et les preuves.

Ensembles et marquages distingués. Étant donné un M-net N, nous distinguons plusieurs ensembles de places particuliers :

- N^{e} est l'ensemble des places d'entrée de N (celles qui ont le statut e) ;

- N^{i} est l'ensemble des places internes de N (avec le le statut i) ;
- N^{x} est l'ensemble des places de sortie de N (de statut x) ;
- N^{c} est l'ensemble des places de contrôle de flot de N (portant le statut e, i ou x) ;
- N^{f} est l'ensemble des places de liens ouvertes de N (dont le statut est dans $\mathbb{B}$, le f vient rappeller que ces places peuvent fusionner) ;
- N^{b} est l'ensemble des places de liens fermées de N (de statut b).

Nous appelons *places d'interfaces* les places de $N^{\mathsf{e}} \uplus N^{\mathsf{x}}$.

Étant donnés un marquage M d'un M-net N et $\varepsilon \in \{\mathsf{e}, \mathsf{i}, \mathsf{x}, \mathsf{c}, \mathsf{f}, \mathsf{b}\}$, nous notons M^{ε} le marquage M restreint aux places de N^{ε}.

Nous distinguons aussi trois marquages d'un M-net N :

- le marquage vide $M_{\varnothing}(N)$ dans lequel aucune place n'est marquée ;
- le marquage d'entrée $M_{\mathsf{e}}(N)$ dans lequel chaque place $s \in N^{\mathsf{e}}$ est marquée par son type $\tau(s)$, les autres places ayant le marquage vide ;
- le marquage de sortie $M_{\mathsf{x}}(N)$ dans lequel chaque place de sortie $s \in N^{\mathsf{x}}$ est marquée par $\tau(s)$, toutes les autres places étant vides.

Le comportement dynamique d'un M-net débute typiquement par le marquage d'entrée, qui est donc un état initial. Le marquage de sortie (qui peut ne pas être atteint) correspond à un état final, depuis lequel le M-net ne peut plus évoluer (puisqu'il est ex-orienté et T-restreint).

Un marquage M d'un M-net N est dit *initial* si $M^{\mathsf{e}} = M_{\mathsf{e}}(N)$ et $M^{\mathsf{x}} = M^{\mathsf{i}} = M_{\varnothing}(N)$. La différence entre le marquage d'entrée et un marquage initial, lorsqu'il y en a une, tient donc aux jetons contenus dans les places de liens. De manière similaire, un marquage est dit *final* s'il est égal au marquage de sortie de N auquel on a éventuellement ajouté des jetons dans les places de liens.

La raison pour introduire la notion de marquage initial et final en plus de celle de marquage d'entrée et de sortie est la présence des places de liens. En effet, considérons le M-net $N_{3.3}$ de la figure 3.3.

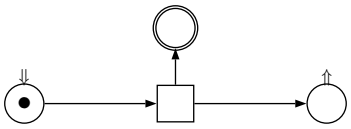


Figure 3.3 — Le M-net $N_{3.3}$ avec son marquage d'entrée.

Le réseau $N_{3.3}$ peut franchir sa transition. Il est alors dans un marquage final et non dans son marquage de sortie puisque la place de liens fermée contient un jeton. D'une manière similaire, un marquage initial différent du marquage d'entrée peut être rencontré quand un réseau tel que $N_{3.3}$ est exécuté plusieurs fois de suite (la place de liens accumulant un nouveau jeton à chaque exécution du réseau).

Représentation graphique. Nous adoptons pour les M-nets la même présentation que pour les réseaux colorés. Les transitions hiérarchiques sont mises en valeur par un double bord et nous ajoutons les simplifications supplémentaires suivantes :

- pour une transition t, l'étiquette $\gamma(t)$ n'est indiquée que si elle n'est pas vide ;
- les places de liens non marquées et non connectées à des transitions ne sont pas représentées ;
- le statut des places est donné de manière graphique plutôt que par une étiquette explicite (sauf quand cette étiquette est dans $\mathbb{B}$, auquel cas elle doit être donnée). La figure 3.4 résume ces notations ;
- les deux parties d'une étiquette de transition (actions et gardes) sont indiquées à des position différentes autour de la transition au lieu d'être connectées par un point. Cela n'entraîne pas d'ambiguïté.

Figure 3.4 — De gauche à droite : une place d'entrée, une place interne, une place de sortie, une place de liens fermée et une place de liens ouverte (pour le symbole de lien $b_1 \in \mathbb{B}$).

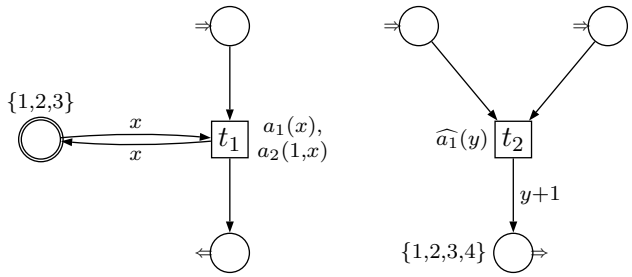
3.3.1 Communications entre transitions

Les communications dans les M-nets se font par deux moyens : la *synchronisation de transitions* et les *liens asynchrones*. Dans les deux cas, des opérateurs permettent de contrôler la façon dont les communications sont établies.

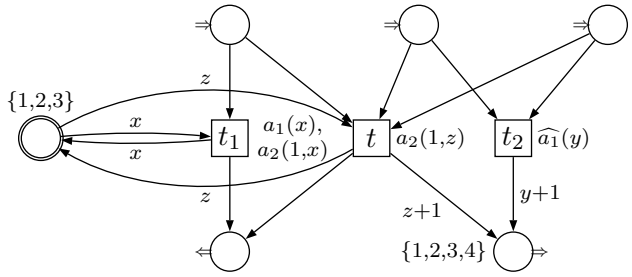
Communications synchrones. L'opérateur permettant de synchroniser des transitions est la *synchronisation*, notée sy. Si N est un M-net et $a \in \mathbb{A}$ un symbole d'action d'arité $\mathsf{ar}(a) = k$, le M-net $N\,\mathsf{sy}\,a$ est obtenu à partir de N dans lequel chaque paire de transitions (t_1, t_2) portant des actions conjuguées $a(x_1, \ldots, x_k)$ et $\widehat{a}(y_1, \ldots, y_k)$ donne lieu à une nouvelle transition correspondant à l'exécution simultanée de t_1 et t_2. Pour cela, les paramètres des actions sont unifiés et la nouvelle transition hérite des arcs de t_1 et t_2. La définition formelle de la synchronisation est complexe [Bes$^+$98, Dev$^+$98]. Dans ce mémoire, il nous suffit d'en comprendre le principe qui est illustré sur la figure 3.5.

Il est important de noter les points suivants.

- La synchronisation présentée sur la figure 3.5 implique une seule paire de transitions. En général, l'application de la synchronisation crée autant de nouvelles transitions qu'il y a de paires telles que (t_1, t_2) dans cet exemple.
- Les paramètres des actions sont unifiés dans les nouvelle transitions (dans l'exemple, on a utilisé l'unificateur $\{x \mapsto z, y \mapsto z\}$). C'est ainsi que la communication devient possible.
- On choisit un unificateur arbitraire pour chaque paire de transitions. Le résultat de la synchronisation dépend donc du choix des unificateurs, mais les différents réseaux sont tous équivalents par renommage de variables.
- Les actions utilisées pour la synchronisation d'une paire de transitions sont supprimées de l'étiquette de la transition résultante.



Le M-net $N_{3.5}$ dans lequel les transitions t_1 et t_2 peuvent être franchies indépendamment.



Le M-net $N_{3.5}\,\mathsf{sy}\,a_1$. Le franchissement de la nouvelle transition t est équivalent au franchissement simultané de t_1 et t_2.

Figure 3.5 — Synchronisation dans les M-nets.

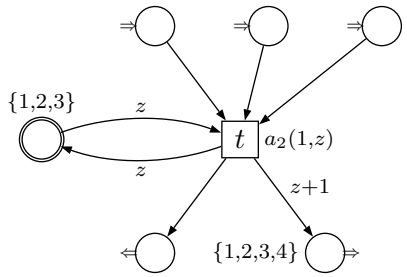
- La synchronisation, comme toutes les opérations sur les M-nets, est purement statique : on modifie la structure d'un M-net de manière à assurer que ses comportement seront ceux que nous souhaitons. Cet aspect statique est très différent de la synchronisation dynamique rencontrée dans les algèbres de processus comme CCS [Mil80, Mil83, Mil89].

La synchronisation n'est pas suffisante pour obtenir un modèle satisfaisant des communications synchrones : nous voyons dans l'exemple que le franchissement de t n'est pas obligatoire et que le réseau peut continuer à exécuter t_1 et t_2 indépendamment. Pour forcer le comportement synchrone, on utilise l'opération de *restriction d'actions* (ou simplement *restriction*) $\mathsf{rs}\,a$ qui supprime les transitions portant des actions basées sur a. La figure 3.6 présente le résultat de l'opération $\mathsf{rs}\,a_1$ sur le réseau donné au bas de la figure 3.5. Nous pouvons observer que la restriction revient simplement à supprimer des transitions avec leurs arcs adjacents.

Presque systématiquement, synchronisation et restriction sont suivies l'une de l'autre. Afin de simplifier leur utilisation, on définit l'opération de *scoping* qui combine les deux autres : étant donnés un M-net N et un symbole d'action a, on définit $N\,\mathsf{sc}\,a \stackrel{\text{df}}{=} (N\,\mathsf{sy}\,a)\,\mathsf{rs}\,a$.

Les opérations de synchronisation, restriction et scoping sont toutes commutatives (chacune avec elle même). On peut donc les étendre sans difficulté à des ensembles de symboles. Soit $\{a_1, \ldots, a_k\} \subseteq \mathbb{A}$, on définit :

$$
\begin{aligned}
N\,\mathsf{sy}\{a_1, \ldots, a_k\} &\stackrel{\text{df}}{=} N\,\mathsf{sy}\,a_1 \cdots \mathsf{sy}\,a_k \quad , \\
N\,\mathsf{rs}\{a_1, \ldots, a_k\} &\stackrel{\text{df}}{=} N\,\mathsf{rs}\,a_1 \cdots \mathsf{rs}\,a_k \quad , \\
N\,\mathsf{sc}\{a_1, \ldots, a_k\} &\stackrel{\text{df}}{=} (N\,\mathsf{sy}\,a_1\,\mathsf{sy}\,\mathsf{sc}\,a_k)\,\mathsf{rs}\,a_1 \cdots \mathsf{rs}\,a_k \quad .
\end{aligned}
$$



Dans le M-net $(N_{3.5} \,\mathsf{sy}\, a_1) \,\mathsf{rs}\, a_1$, les transitions t_1 et t_2 ayant été supprimées, seul le franchissement simultané reste possible.

Figure 3.6 — Restriction dans les M-nets.

Pour simplifier les explications, nous considérerons parfois séparément deux transitions synchronisées. Dans l'exemple de la figure 3.6, nous dirons volontiers « le franchissement de t_1 entraîne celui de t_2 » plutôt que le « franchissement du résultat de la synchronisation de t_1 et t_2 ». Même si les transitions que nous citons ont en fait disparu, les explications se décomposent plus facilement en les considérant séparément.

Communications asynchrones. L'autre aspect des communications dans les M-nets est réalisé au moyen des places de liens : certaines transitions peuvent y placer des jetons que d'autres pourront consommer plus tard, réalisant ainsi une communication asynchrone. Le principe des liens asynchrones est que les places de liens ouvertes sont automatiquement fusionnées lorsqu'on compose plusieurs M-nets au moyen des opérateurs de contrôle de flot de l'algèbre. La définition formelle de cette fusion est donc donnée dans la section suivante, avec la définition de ces opérateurs.

Intuitivement, l'application d'un opérateur de contrôle de flot juxtapose les M-nets qu'il prend en argument et combine leurs places d'interface pour forcer le comportement souhaité. Par exemple, si N_1 et N_2 sont deux M-nets, $N_1 \| N_2$ est un nouveau M-net qui s'obtient en mettant côte à côte les deux M-nets qui le composent, sans modifier leurs places de contrôle de flot ni les places de liens fermées. Par contre, les places de liens ouvertes portant le même symbole de liens sont fusionnées. Ce principe est illustré sur la figure 3.7.

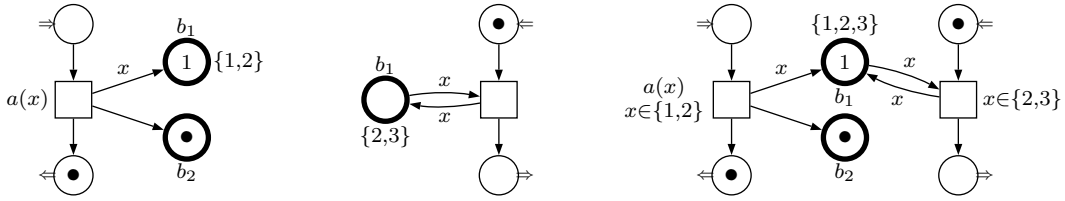


Figure 3.7 — De gauche à droite, les M-nets N_1, N_2 et $N_1 \| N_2$ dans lequel les places de liens issues des deux réseaux qui le composent ont fusionné.

Nous pouvons résumer les caractéristiques importantes de la fusion de places.

- Les marquages des places fusionnées sont ajoutés.

- Le type des places à fusionner peut être différent, dans ce cas, la place résultante à pour type l'union des types des places fusionnées.
- Lorsque les places fusionnées n'ont pas le même type, la garde des transitions est augmentée afin d'assurer qu'il n'existe pas de nouvelle valuation qui la rende franchissable.

En plus de cette fusion automatique des places de liens ouvertes, l'algèbre des M-nets propose un moyen de rendre privées les communications établies sur un lien. Pour cela, on utilise l'opération de *restriction de liens*, notée tie. Si N est un M-net et $b \in \mathbb{B}$ un symbole de lien, $N\,\mathsf{tie}\,b$ est un nouveau M-net qui est obtenu à partir de N dans lequel la place de liens de statut b reçoit le statut $\mathsf{b} \in \mathbb{S}$, et auquel une nouvelle place isolée de statut b et de type vide est ajoutée (pour conserver la B-restriction). Formellement, si $N = (S, T, \iota)$, alors $N\,\mathsf{tie}\,b$ est le réseau $N' = (S', T', \iota')$ défini par :

- $S' \stackrel{\text{df}}{=} S \uplus \{s_b\}$ et, pour toute place $s \in S'$,

$$\iota'(s) \stackrel{\text{df}}{=} \begin{cases} b.\varnothing & \text{si } s = s_b, \\ \mathsf{b}.\tau(s) & \text{si } \lambda(s) = b, \\ \iota(s) & \text{sinon}\,; \end{cases}$$

- $T' \stackrel{\text{df}}{=} T$ et pour toute transition $t \in T'$, $\iota'(t) \stackrel{\text{df}}{=} \iota(t)$;
- pour toute place $s \in S$ et toute transition $t \in T'$, $\iota'(s,t) \stackrel{\text{df}}{=} \iota(s,t)$, $\iota'(t,s) \stackrel{\text{df}}{=} \iota(t,s)$ et $\iota'(s_b,t) \stackrel{\text{df}}{=} \iota'(t,s_b) = \varnothing$.

Comme pour la synchronisation, la restriction et le scoping, nous pouvons étendre la restriction de liens à des ensembles de symboles de liens (nous pouvons facilement vérifier que tie commute avec lui-même). Pour $\{b_1, \ldots, b_k\} \subseteq \mathbb{B}$ et un M-net N, nous avons donc :

$$N\,\mathsf{tie}\,\{b_1, \ldots, b_k\} \stackrel{\text{df}}{=} N\,\mathsf{tie}\,b_1 \cdots \mathsf{tie}\,b_k \quad .$$

3.3.2 Contrôle de flot et substitution de transitions

En plus des opérateurs de communication, l'algèbre des M-nets propose des opérateurs de contrôle de flot permettant de composer des M-nets pour forcer l'ordre de leurs exécutions. Plus précisément, si N_1, N_2 et N_3 sont des M-nets, on peut les composer d'une des façons suivantes.

- *Composition parallèle :* $N_1 \| N_2$. Les deux réseaux peuvent s'exécuter en concurrence.
- *Composition séquentielle :* $N_1 ; N_2$. L'exécution du résultat commence par celle de N_1 et est suivie par celle de N_2 après la terminaison de N_1.
- *Choix :* $N_1 \,\square\, N_2$. Seul l'un des deux réseaux composés peut s'exécuter.
- *Itération :* $[N_1 * N_2 * N_3]$. L'exécution du résultat commence par celle de N_1 (initialisation), est suivie d'un nombre arbitraire d'exécutions de N_2 (répétition) et est terminée par une exécution de N_3 (terminaison).

Les opérateurs de contrôle de flot sont tous synthétisés à partir d'un opérateur de *substitution de transitions* (aussi appelé *raffinement*) qui permet de remplacer par des M-nets des transitions étiquetées par une action hiérarchique.

Substitution de transitions. Cet opérateur est très général et permet toutes sortes de substitutions sans restrictions. La substitution de transitions est un mécanisme très complexe qui a été au centre d'une thèse [Rie99] et de plusieurs articles [Dev$^+$97, Dev$^+$02, Dev$^+$98]. Il n'est pas utile dans le cadre de cette thèse de présenter toute la généralité de cette opération car nous l'utiliserons dans des cas simples uniquement. En fait, cette généralité a été introduite avec grand soin et l'opération a finalement toujours le comportement qu'on en attend intuitivement. La complexité peut donc rester cachée et nous pouvons profiter de l'opération comme d'un outil bien conçu, sans nous soucier des détails de sa réalisation.

Soient N, N_1, ..., N_k des M-nets et $\{\mathcal{X}_1, \ldots, \mathcal{X}_k\} \subseteq \mathbb{X}$ un ensemble d'actions hiérarchiques. Le M-net $N' \stackrel{\text{df}}{=} N[\mathcal{X}_1 \leftarrow N_1, \ldots, \mathcal{X}_k \leftarrow N_k]$ (aussi noté $N[\mathcal{X}_i \leftarrow N_i \mid 1 \leq i \leq k]$) est N dans lequel chaque transition hiérarchique étiquetée par $\mathcal{X}_i$ a été remplacée par une copie de N_i (pour $1 \leq i \leq k$). Pour substituer une transition t par un N_i, il faut d'abord remplacer t par une copie de N_i privé de ses places d'interfaces, ensuite, le « recollement » de N_i à N se fait de la façon suivante : les arcs partant des places de N_i^{e} sont attachés aux places de ${}^\bullet t$ et les arcs allant aux places de N_i^{x} le sont à celle de $t^\bullet$; les annotations étant modifiées en conséquence.

Dans le M-net $N[\mathcal{X}_i \leftarrow N_i \mid 1 \leq i \leq k]$, chaque transition t_i de N telle que $\iota(t_i) = \mathcal{X}_i$, est substituée par une copie de N_i. Pour effectuer cette copie, chaque transition t et chaque place interne s de N_i est renommée en $t_i.t$ et $t_i.s$ respectivement. Ce renommage est effectué pour chaque copie de N_i. Cette manière de faire garantit l'unicité des noms et donc que les copies sont bien disjointes. Signalons que les places d'interface de N_i étant supprimées, elles n'ont pas à être renommées.

Lorsqu'on remplace une transition t_i par un réseau N_i, on change les types des places de ${}^\bullet t_i$, de $t_i{}^\bullet$ et des places internes de N_i (renommées) ; les étiquettes des arcs de la copie de N_i sont aussi changées en conséquence. Le but de ces changements est de permettre deux niveaux d'exécution simultanés : le premier niveau concerne t_i qui peut éventuellement être franchie pour plusieurs valuations ; le second niveau concerne les transitions de N_i qui peuvent aussi avoir leurs propres valuations. Si par exemple t_i peut être franchie pour une valuation σ_i et qu'une transition t de N_i peut être franchie pour une valuation σ, alors les changements de types et les annotations structurées permettent de franchir $t_i.t$ pour une valuation qui combine σ_i et σ et permet de représenter les deux niveaux d'exécution de manière cohérente. Plusieurs substitutions successives augmentent la hiérarchisation suggérée ci-dessus et il n'y a aucune limite au nombre de niveaux exécutables simultanément.

Ce mécanisme permet une grande généralité dans les substitutions autorisées ; en fait, il n'y a aucune restriction. Il en existe même des versions paramétriques dans lesquelles les valeurs transportées par la transition hiérarchique sont transmises au réseau qui la remplace. Ce paramétrage permet en particulier de donner une sémantique compositionnelle aux procédures dans un langage de programmation parallèle, en autorisant différents passages de paramètres (par valeur, par retour ou par référence) ainsi que des appels récursifs ou concurrents [Kla01].

Pour résumer l'intuition, l'opération $N[\mathcal{X}_i \leftarrow N_i \mid 1 \leq i \leq k]$ remplace dans N chaque

transition hiérarchique t_i étiquetée $\mathcal{X}_i$ par une copie de N_i (pour $1 \leq i \leq k$) de telle façon que :

- le marquage par leurs types des places de ${}^\bullet t_i$ correspond au marquage d'entrée de N_i ;
- le marquage de sortie de N_i correspond au marquage des places de $t_i{}^\bullet$ par leurs types ;
- un franchissement de t_i est remplacé par une exécution de N_i, cette exécution consomme et produit des marquages correspondant au résultat du franchissement de t_i dans N ;
- les transitions et les places de N_i sont renommées en les préfixant du nom de la transition qu'elles remplacent ;
- si t_i peut être franchie simultanément avec deux valuations distinctes, N_i peut aussi exécuter deux instances concurrentes et totalement indépendantes ;
- les types de places et les annotations des arcs sont ajustés de manière à permettre une exécution cohérente à deux niveaux : celui de t_i et celui de N_i.

Prise en compte des places de liens. Jusqu'à présent, nous n'avons pas évoqué la gestion des places de liens dans le cadre de l'opération de substitution de transitions. Cette opération a été originellement définie avant l'apparition des places de liens asynchrones ; nous donnons ici les adaptations nécessaires à la prise en compte de ces nouvelles places. Pour ces définitions, notons $N\langle \mathcal{X}_i \leftarrow N_i \mid 1 \leq i \leq k \rangle$ la substitution originale, la substitution adaptée aux places de liens étant notée $N[\mathcal{X}_i \leftarrow N_i \mid 1 \leq i \leq k]$ puisque c'est cette version que nous utiliserons dorénavant.

Pour effectuer la substitution $N\langle \mathcal{X}_i \leftarrow N_i \mid 1 \leq i \leq k \rangle$, nous utilisons la définition originale en considérant les places de liens comme internes (de statut i) mais sans changer leur statut. Le résultat est un M-net qui n'est pas B-restreint en général : en effet puisque N et les N_i le sont, chaque remplacement d'une transition de N par un N_i ajoute les places de liens de N_i. Nous avons donc, pour chaque symbole de lien $b \in \mathbb{B}$, une place de statut b issue de N plus autant de telles places issues de chacun des N_i qu'il existe de transitions hiérarchiques dans N portant l'étiquette $\mathcal{X}_i$. (Le préfixage des noms est alors aussi appliqué aux places de liens puisque nous les considérons comme internes.) Pour rétablir la B-restriction, il suffit de fusionner les places de liens ouvertes de même statut (rappelons que la B-restriction de N empêche ses places de liens d'être connectées à ses transitions hiérarchiques). Cette fusion est possible même si les places n'ont pas le même type, il faut simplement en tenir compte et adapter la garde des transitions liées à ces places. La figure 3.7 illustre la définition donnée ci-dessous.

Supposons que $N = (S, T, \iota)$ et $N_i = (S_i, T_i, \iota_i)$ pour $1 \leq i \leq k$. Soit le M-net $N' = (S', T', \iota') \stackrel{\text{df}}{=} N\langle \mathcal{X}_i \leftarrow N_i \mid 1 \leq i \leq k \rangle$ obtenu en considérant les places de liens comme internes. Notons $T_{\mathcal{X}_i} = \{t \in T \mid \lambda(t) = \mathcal{X}_i\}$ pour $1 \leq i \leq k$, de plus, pour chaque $b \in \mathbb{B}$, notons s_b la place de S de statut b (elle est aussi dans S') et $S_b = \{t.s \in S' \mid t \in T_{\mathcal{X}_i}, 1 \leq i \leq k \text{ et } \lambda'(t.s) = b\}$. Alors, $N[\mathcal{X}_i \leftarrow N_i \mid 1 \leq i \leq k] \stackrel{\text{df}}{=} N'' = (S'', T'', \iota'')$ est défini de la façon suivante :

- $T'' \stackrel{\text{df}}{=} T'$;

- $S'' \stackrel{\text{df}}{=} S' \setminus \cup_{b\in\mathbb{B}} S_b$;
- pour toute place $s' \in S'$ telle que $\lambda'(s') \notin \mathbb{B}$, $\iota''(s') \stackrel{\text{df}}{=} \iota'(s')$;
  pour chaque $b \in \mathbb{B}$, $\lambda''(s_b) \stackrel{\text{df}}{=} b$ et $\tau''(s_b) \stackrel{\text{df}}{=} \tau'(s_b) \cup_{s\in S_b} \tau'(s)$;
- pour toute transition hiérarchique $t \in T''$, $\iota''(t) \stackrel{\text{df}}{=} \iota'(t)$; pour toute transition non hiérarchique $t \in T''$, $\alpha''(t) \stackrel{\text{df}}{=} \alpha'(t)$ et $\gamma''(t) \stackrel{\text{df}}{=} \gamma'(t) \bigwedge_{p\in P_t} p$, où P_t est le plus petit ensemble de prédicats tel que, pour chaque $b \in \mathbb{B}$ et chaque place $s \in S_b \uplus \{s_b\}$:
  - si $\iota'(s,t) \neq \varnothing$ et $\tau''(s_b) \neq \tau'(s)$ alors pour tout $x \in \iota'(s,t)$, $(x \in \tau_i(s)) \in P_t$,
  - si $\iota'(t,s) \neq \varnothing$ et $\tau''(s_b) \neq \tau'(s)$ alors pour tout $x \in \iota'(t,s)$, $(x \in \tau_i(s)) \in P_t$;
- pour toute transition $t \in T''$ et toute place $s \in S''$ telle que $\lambda''(s) \notin \mathbb{B}$,
  - $\iota''(s,t) \stackrel{\text{df}}{=} \iota'(s,t)$ et,
  - $\iota''(t,s) \stackrel{\text{df}}{=} \iota'(t,s)$;

  pour toute transition $t \in T''$ et tout $b \in \mathbb{B}$,
  - $\iota''(s_b,t) \stackrel{\text{df}}{=} \cup_{s\in S_b\cup\{s_b\}} \iota'(s,t)$ et,
  - $\iota''(t,s_b) \stackrel{\text{df}}{=} \cup_{s\in S_b\cup\{s_b\}} \iota'(t,s)$;

Notons que dans la définition de $\iota''(s_b,t)$ et $\iota''(t,s_b)$, les unions utilisées comportent au plus un ensemble non vide. En effet, les places de $S_b \cup \{s_b\}$ sont toutes issues de réseaux différents et chaque transition t ne peut être connectée qu'à une de ces places (au plus). Pour la même raison, l'ensemble P_t construit pour ajuster la garde d'une transition t contient au plus deux éléments par symbole de liens b, un pour chaque arc (entrant ou sortant) pouvant le relier à la place de liens de statut b dans le réseau dont t est issu.

Si N et les N_i portent les marquages M et M_i ($1 \leq i \leq k$) respectivement, en gardant les mêmes notations que ci-dessus, appelons M' le marquage de N' et M'' celui de N''. Alors, M'' est défini par :

- pour toute place $s \in S''$ telle que $\lambda''(s) \notin \mathbb{B}$, $M''(s) \stackrel{\text{df}}{=} M'(s)$;
- pour tout $b \in \mathbb{B}$, $M''(s_b) \stackrel{\text{df}}{=} M'(s_b) + \sum_{s\in S_b} M'(s)$.

Contrôle de flot. En nous basant sur la substitution de transitions, nous pouvons définir les opérateurs de contrôle de flot de l'algèbre des M-nets. Pour cela, nous utilisons les *réseaux opérateurs* présentés sur la figure 3.8. Dans ces réseaux, toutes les transitions sont hiérarchiques et chacune est remplacée par un des arguments de l'opérateur correspondant (ainsi, la transition étiquetée $\mathcal{X}_2^{\square}$ de $N_{\square}$ est substituée par une copie du second argument de l'opérateur de choix).

Munis de ces M-nets opérateurs, nous pouvons définir les opérateurs de contrôle de flot. Soient N_1, N_2 et N_3 trois M-nets, alors :

$$
\begin{aligned}
N_1 \| N_2 &\stackrel{\text{df}}{=} N_{\|}[\mathcal{X}_1^{\|} \leftarrow N_1, \mathcal{X}_2^{\|} \leftarrow N_2] \\
N_1 \,\square\, N_2 &\stackrel{\text{df}}{=} N_{\square}[\mathcal{X}_1^{\square} \leftarrow N_1, \mathcal{X}_2^{\square} \leftarrow N_2] \\
[N_1 * N_2 * N_3] &\stackrel{\text{df}}{=} N_{[**]}[\mathcal{X}_1^{[**]} \leftarrow N_1, \mathcal{X}_2^{[**]} \leftarrow N_2, \mathcal{X}_3^{[**]} \leftarrow N_3] \\
N_1 ; N_2 &\stackrel{\text{df}}{=} N_{;}[\mathcal{X}_1^{;} \leftarrow N_1, \mathcal{X}_2^{;} \leftarrow N_2]
\end{aligned}
$$

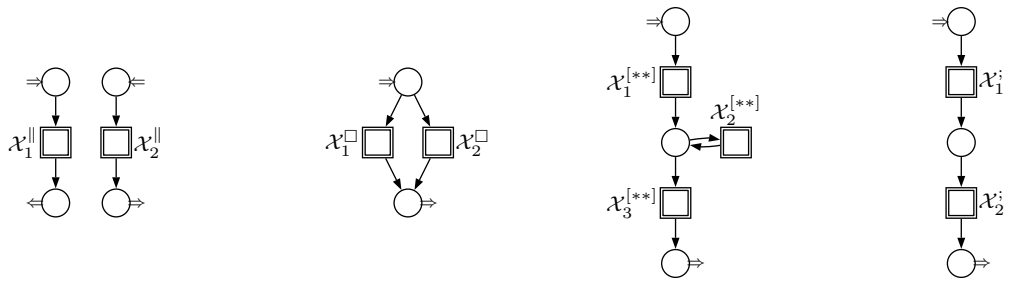


Figure 3.8 — Les M-nets opérateurs utilisés pour synthétiser les opérateurs de contrôle de flot. De gauche à droite : $N_{\parallel}$, $N_{\square}$, $N_{[**]}$ et $N_{;}$.

3.3.3 Autres opérateurs

L'algèbre des M-nets comprend encore deux opérateurs moins centraux.

Renommage. Dans un M-net, on peut renommer globalement les symboles utilisés pour les communications ou les substitutions de transitions. Étant donnés un M-net N et un couple de symboles $(x, y) \in \mathbb{A}^2 \uplus \mathbb{B}^2 \uplus \mathbb{X}^2$, le M-net $N[x \mapsto y]$ est égal à N dans lequel toutes les occurrences de x (dans les annotations) ont été substituées par y.

On peut, au lieu de $x \mapsto y$, utiliser des *fonctions de renommage* permettant de substituer plusieurs symboles à la fois. Une fonction de renommage est une fonction partielle f de l'ensemble $\mathbb{A} \uplus \mathbb{B} \uplus \mathbb{X}$ dans lui même telle que, pour tout x dans le domaine de f, on a $(x, f(x)) \in \mathbb{A}^2 \uplus \mathbb{B}^2 \uplus \mathbb{X}^2$. Le renommage des symboles est alors opéré simultanément sur tous les symboles du domaine de f. En général, il n'est pas équivalent de renommer les symboles un à un. Par exemple, si x et y apparaissent dans N en des endroits distincts, les trois M-nets suivants sont deux à deux distincts : $N[x \mapsto y, y \mapsto x]$, $N[x \mapsto y][y \mapsto x]$, et $N[y \mapsto x][x \mapsto y]$. En effet, dans le premier cas, les occurrences de x et de y sont échangée ; dans le second cas, tous les y deviennent x (les x étant laissés en place) ; dans le dernier cas, tous les x deviennent y (les y n'étant pas changés).

L'opération de renommage ainsi définie risque de briser la B-restriction. En effet, la fonction de renommage $b_1 \mapsto b_2$ amène deux places de liens de statut b_2 alors qu'il n'en reste aucune de statut b_1. Pour rétablir la B-restriction, nous pouvons utiliser la substitution de transitions qui la rétablit systématiquement. Étant donnés un M-net N et une fonction de renommage f, en notant $N\langle f \rangle$ le renommage défini ci-dessus, nous définissons un renommage qui conserve la b-restriction :

$$N[f] \stackrel{\text{df}}{=} N_{[]}[\mathcal{X}^{[]} \leftarrow N\langle f \rangle] \quad ,$$

où $N_{[]}$ est le M-net présenté sur la figure 3.9.

Récursion. Nous présentons ici l'opération de *récursion* qui est assez particulière puisqu'elle peut générer des M-nets infinis (en nombre de places et de transitions). Nous ne l'utilisons pas dans ce mémoire mais elle est présentée ici par souci d'être complet.

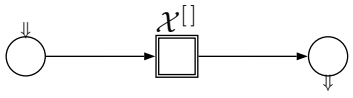


Figure 3.9 — Le M-net $N_{[]}$. (Ses places de liens étant isolées, elles ne sont pas représentées, mais ce M-net est B-restreint.)

Étant donnés un ensemble d'indices $I = \{1, \ldots, k\}$, un ensemble de M-nets $\{N_1, \ldots, N_k\}$ et un ensemble d'actions hiérarchiques $\{\mathcal{X}_1, \ldots, \mathcal{X}_k\} \subseteq \mathbb{X}$, le M-net $N_\mu \stackrel{\text{df}}{=} \mu\{\mathcal{X}_i.N_i \mid i \in I\}N$ est défini intuitivement comme une succession infinie de substitutions (la définition exacte fait appel à une notion de point fixe des substitutions récursives) :

$$N_\mu = N[\mathcal{X}_1 \leftarrow N_1, \ldots, \mathcal{X}_k \leftarrow N_k][\mathcal{X}_1 \leftarrow N_1, \ldots, \mathcal{X}_k \leftarrow N_k] \cdots$$

Si l'un des $\mathcal{X}_i$ apparaît effectivement dans le N_i correspondant et que ce dernier comporte plus d'une transition, le réseau N_μ est infini. Cette opération est bien définie théoriquement et de nombreuses propriétés lui sont attachées. En pratique, la récursion n'est pas toujours nécessaire avec les M-nets (même si certains systèmes ne peuvent être obtenus autrement) ; par exemple, la définition de la sémantique de procédures récursives [Kla01] peut se faire sans cette opération : pour autoriser une récursivité non bornée, on peut en effet utiliser des types de places infinis plutôt que des réseaux avec un nombre infini de places et de transitions.

3.3.4 L'algèbre et ses propriétés

L'algèbre des M-nets comprend les opérateurs définis dans le début de cette section, dont nous donnons un récapitulatif ci-dessous :

$N_1 \| N_2$	composition parallèle	$N_1; N_2$	séquence
$N_1 \Box N_2$	choix	$[N_1 * N_2 * N_3]$	itération
$N_1 \,\mathsf{sy}\, a$	synchronisation	$N_1 \,\mathsf{rs}\, a$	restriction d'actions
$N_1 \,\mathsf{sc}\, a$	scoping	$N_1 \,\mathsf{tie}\, b$	restriction de liens
$N_1[f]$	renommage	$N[\mathcal{X}_i \leftarrow N_i \mid 1 \leq i \leq k]$	substitution

Théorème 1.
Soient N, N_1, …, N_k des M-nets bien formés, $a \in \mathbb{A}$, $b \in \mathbb{B}$, f une fonction de renommage et $\{\mathcal{X}_1, \ldots, \mathcal{X}_k\} \subseteq \mathbb{X}$. Alors, les M-nets suivants sont bien formés :

$$N \,\mathsf{sy}\, a \quad ; \quad N \,\mathsf{rs}\, a \quad ; \quad N[\mathcal{X}_i \leftarrow N_i \mid 1 \leq i \leq k] \quad ; \quad N[f] \quad ; \quad N \,\mathsf{tie}\, b \quad .$$

Preuve. Cette propriété existe dans l'algèbre des M-nets sans places de liens. Il nous suffit donc de montrer que la B-restriction est conservée.

Le cas de la synchronisation et de la restriction est évident puisque les places ne sont pas influencées par ces opérations.

La substitution de transitions a été définie de façon à assurer la B-restriction. En effet, si N est b-restreint, alors, même si les N_i ne le sont pas, il y a dans le réseau résultant

une place de liens pour chaque symbole de liens. Ensuite, la fusion assure que chaque lien n'est représenté qu'une fois.

Le renommage est aussi correct par définition puisque nous utilisons la substitution de transitions pour rétablir explicitement la b-restriction.

La restriction de liens $N \,\mathsf{tie}\, b$ est correcte puisque la place de statut b prend le statut b et qu'une nouvelle place de statut b est aussitôt ajoutée. □

Nous déduisons du théorème 1 que l'algèbre des M-nets dans son entier conserve la propriété d'être un M-net bien formé.

Corollaire 2.
Tout M-net obtenu par scoping ou par un des opérateurs de contrôle de flot à partir de M-nets bien formés est aussi bien formé. □

Pour un entier $k \geq 1$, un marquage M d'un M-net N est *k-borné* s'il associe à chaque place un multi-ensemble avec au plus k répétitions par élément (et donc $M(s)(v) \leq k$ pour toute place s de N et toute valeur $v \in \mathit{Val}$). Un M-net marqué (N, M) est *k-borné* si M est k-borné ainsi que tous les marquages accessibles dans N à partir de M. Enfin, un M-net marqué (N, M) est *borné* s'il existe un entier $k \geq 1$ tel que (N, M) et toutes ses évolutions sont k-bornés.

Un M-net dont les places de contrôle sont 1-bornées est dit *sûr* ; par rapport à la définition habituelle, celle-ci ne considère que les places de contrôle, en effet, nous ne pouvons pas attendre des places de liens qu'elles soient bornées. Ceci est illustré par l'exemple de la figure 3.10. Cependant, nous verrons dans la suite que les M-nets sont toujours saufs et nous montrerons à la fin de ce chapitre comment obtenir des réseaux k-bornés à partir de tout M-net.

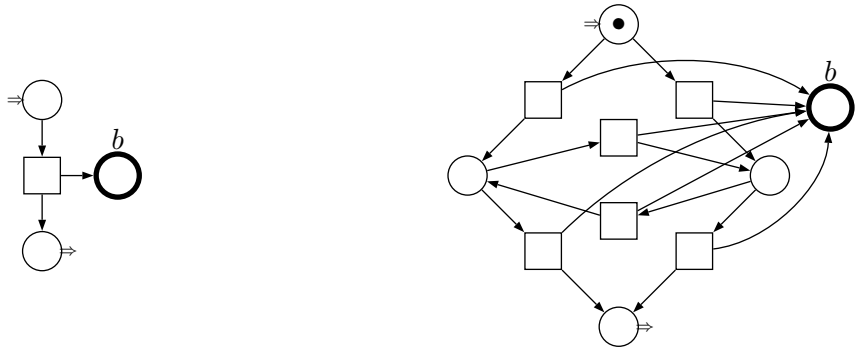


Figure 3.10 — À gauche, le M-net $N_{3.10}$ (avec $b \in \mathbb{B}$) ; à droite, le M-net $[N_{3.10} * N_{3.10} * N_{3.10}]$ avec son marquage d'entrée. Dans ce second M-net, la place de lien de statut b n'est pas bornée.

Théorème 3.
Tout M-net composé de réseaux sûrs est sur.

Preuve. Cette propriété existe dans l'algèbre des M-nets sans places de liens. Elle est conservée ici puisque nous ne nous intéressons qu'aux places de contrôle et que, par la

T-restriction les places de liens ne peuvent pas ajouter de comportement. □

Remarquons qu'une définition des réseaux sûrs basés sur le dépliage ne permettrait pas ce résultat car l'évaluation des annotations structurés par la substitution de transitions induit une perte d'information. En particulier, lorsque des places sont agrégées, les annotations des M-nets gardent le nom de ces places dans la valeur des jetons, qui sont alors différenciés. Cette information est éliminée lors du dépliage ce qui produit alors deux jetons (identiques) dans la même place [Dev$^+$97, Dev$^+$02].

Une conséquence intéressante du précédent théorème est que les steps dans les M-nets sûrs sont des ensembles de transitions valuées et non des multi-ensembles. En effet, par la T-restriction, chaque transition à un arc entrant issu d'une place de contrôle. Puisque celle-ci contient au plus un jeton de chaque valeur de son type, deux instances de la transition avec la même valuation ne peuvent pas être franchies simultanément. Interdire l'auto-concurrence des transitions garantit l'existence de sémantiques concurrentes puissantes en termes de structure d'événements [Hoo$^+$96]. Dans la suite, sauf indication du contraire, nous n'utilisons que des M-nets sûrs.

M-nets statiques et dynamiques. Un M-net N est *statique* si toute place de N^{c} a le marquage vide. Un réseau statique n'a pas forcément le marquage vide : il peut y avoir des jetons dans des places de liens. Pourtant, un tel M-net étant T-restreint, il ne peut pas franchir de transition.

Les M-nets *dynamiques* peuvent avoir des jetons dans les places de contrôle (et bien sûr aussi dans les places de liens). Cela peut leur permettre de franchir des transitions, mais ce n'est pas forcément le cas, par exemple si les jetons forment le marquage final. Le marquage d'un M-net dynamique ne peut pas être quelconque, il doit être accessible à partir d'un marquage initial du M-net. Ainsi : un M-net marqué (N, M) est *dynamique* s'il existe M', un marquage initial de N, tel que M est accessible à partir de M'.

Proposition 4.
Soit (N, M) un M-net dynamique tel que $M[U\rangle M'$ pour un step U et un marquage M' de N. Alors (N, M') est un M-net dynamique.

Preuve. Par définition. □

La classe des M-nets dynamiques est celle qui nous intéresse en dernier ressort. En effet, la modélisation d'un système avec des M-nets passe le plus souvent par la composition de M-nets statiques pour obtenir un M-net statique représentant le système entier. En lui donnant son marquage d'entrée, nous obtenons un M-net dynamique dont toutes les évolutions sont des M-nets dynamiques. Dans la suite de ce mémoire, nous nous limitons donc aux M-nets dynamiques et statiques.

3.4 Liens bornés

Nous avons vu jusqu'à présent que les places de liens peuvent ne pas être bornées (voir par exemple la figure 3.10). Pourtant, en pratique, nous n'avons jamais rencontré d'exemple concret où nous aurions eu besoin de places de liens non bornées. Dans cette section,

nous définissons une sous-classe des M-nets dont les réseaux restent toujours bornés. Nous donnons ensuite une transformation qui produit un tel réseau à partir d'un M-net (dont le marquage doit bien-entendu être borné).

Pour chaque place s, on ajoute une *place complémentaire* $\widehat{s}$; ces deux places se « partagent » des jetons dont le nombre total, noté k, ne varie pas. Lorsqu'un jeton est produit dans s, il est en même temps consommé dans $\widehat{s}$, et inversement. Cette façon de procéder interdit tout franchissement qui produirait un marquage non k-borné. En effet, pour produire $k+1$ jetons identiques dans une place de liens, il faut en puiser autant dans sa place complémentaire mais il n'y en a que k à cause de la contrainte imposée sur les marquages.

Soit un entier $k \geq 1$. Un *k-net* est un M-net $N = (S, T, \iota)$ tel qu'il existe une involution sur $N^{\mathsf{f}} \cup N^{\mathsf{b}}$, notée comme la conjugaison d'actions, qui vérifie : pour toute place de liens s, $\tau(\widehat{s}) = \tau(s)$ et, pour chaque transition $t \in T$, $\iota(s,t) = \iota(t,\widehat{s})$ et $\iota(t,s) = \iota(\widehat{s},t)$.

Un marquage M d'un k-net N est *k-valide* (ou simplement *valide* si k peut-être connu par le contexte) s'il respecte la contrainte suivante : pour toute place de liens s de N dont $\widehat{s}$ est la place complémentaire, $M(s) + M(\widehat{s}) = k * \tau(s)$. Lorsque cette contrainte est assurée par le marquage initial, elle reste vérifiée pour toute évolution. En conséquence, un k-net est toujours k-borné.

Proposition 5.
Étant donné un entier $k \geq 1$, soit N un k-net et soit M un marquage valide de N. Alors, tout marquage accessible à partir de M est valide.

Preuve. Considérons une place s et sa place complémentaire $\widehat{s}$. Puisque M est valide, nous avons $M(s) + M(\widehat{s}) = k * \tau(s)$. Soient t une transition et σ une valuation la rendant franchissable telles que $M[\sigma(t)\rangle M'$. Nous avons $M'(s) = M(s) - \iota(s,t) + \iota(t,s)$ et $M'(\widehat{s}) = M(\widehat{s}) - \iota(\widehat{s},t) + \iota(t,\widehat{s})$. Le réseau N étant un k-net, nous avons $\iota(s,t) = \iota(t,\widehat{s})$ et $\iota(t,s) = \iota(\widehat{s},t)$ et donc $M'(s) + M'(\widehat{s}) = M(s) + M(\widehat{s}) = k * \tau(s)$. □

Transformation de M-nets en k-nets. Soient $N = (S, T, \iota)$ un M-net et M un marquage k-borné de N. Nous définissons $\mathsf{knet}(k, N, M) \stackrel{\text{df}}{=} (N', M')$, où $N' = (S', T', \iota')$ est un M-net et M' est un marquage de N', de la façon suivante :

- $T' \stackrel{\text{df}}{=} T$;
- $S' \stackrel{\text{df}}{=} S \uplus \{\widehat{s} \mid s \in S \text{ et } \lambda(s) \in \mathbb{B} \uplus \{\mathsf{b}\}\}$;
- pour toute place $s' \in S'$ et toute transitions $t' \in T'$:
  - $\iota'(t') \stackrel{\text{df}}{=} \iota(t')$,
  - $\iota'(s') \stackrel{\text{df}}{=} \begin{cases} \iota(s') & \text{si } s' \in S, \\ \mathsf{b}.\tau(s) & \text{si } s' = \widehat{s}, \end{cases}$
  - $\iota'(s',t') \stackrel{\text{df}}{=} \begin{cases} \iota(s',t') & \text{si } s' \in S, \\ \iota(t',s) & \text{si } s' = \widehat{s}, \end{cases}$
  - $\iota'(t',s') \stackrel{\text{df}}{=} \begin{cases} \iota(t',s') & \text{si } s' \in S, \\ \iota(s,t') & \text{si } s' = \widehat{s}; \end{cases}$

- pour toute place $s' \in S'$,
  $M'(s) \stackrel{\text{df}}{=} \begin{cases} M(s') & \text{si } s' \in S, \\ k * \tau(s) - M(s) & \text{si } s' = \widehat{s}. \end{cases}$

Nous en arrivons à l'objet principal de cette section : nous pouvons transformer tout M-net ayant un marquage initial k-borné en un k-net, c'est à dire en un M-net dont le marquage est k-borné, de même que tous ses marquages accessibles. Nous montrons par ailleurs que cette transformation est cohérente par rapport aux évolutions k-bornées du M-net de départ. Autrement dit, un M-net et son k-net associé ont la même sémantique steps si le M-net est k-borné ; si ce n'est pas le cas, nous obtenons la sémantique steps du k-net en restreignant celle du M-net aux séquences de steps ne produisant que des marquages k-bornés.

Théorème 6.
Soient $k \geq 1$ un entier, N un M-net et M un marquage k-borné de N. Soit aussi $(N_k, M_k) \stackrel{\text{df}}{=} \mathsf{knet}(k, N, M)$. Alors :

1. *(N_k, M_k) est un M-net marqué valide.*

2. *Si $(N_k, M_k)[U\rangle(N_k, M'_k)$ alors il existe un marquage k-borné M' de N tel que $(N, M)[U\rangle(N, M')$, de plus, $\mathsf{knet}(N, M') = (N_k, M'_k)$.*

3. *Si $(N, M)[U\rangle(N, M')$ et que M' est k-borné, alors $(N_k, M_k)[U\rangle\mathsf{knet}(k, N, M')$.*

Preuve. (1) Par définition de knet, N_k est un k-net et M_k est valide. La proposition 5 nous donne que tout marquage accessible depuis (N_k, M_k) est valide et donc k-borné.

(2) L'existence de M' et le fait qu'il est k-borné sont des conséquences de la proposition 5 et de la définition de la transformation knet qui ne fait qu'ajouter des places et des arcs, et donc, ne peut pas entraîner de nouveaux franchissements. Nous montrons maintenant que $(N_k, M_k)[U\rangle\mathsf{knet}(k, N, M')$.

Commençons par remarquer que deux M-nets qui ne diffèrent que par leurs marquages sont transformés par knet en deux k-nets qui ne diffèrent aussi que par leurs marquages. D'autre part, dans la définition de knet, les places de contrôle et les arcs associés sont conservés, nous ne considérons donc que les marquages des places de liens. Appelons alors M'' le marquage de $\mathsf{knet}(N, M')$. Sans perte de généralité, nous pouvons supposer que le step U se réduit à une unique transitions valuée $\sigma(t)$ et, pour toute place de lien s dans N (qui est aussi dans N_k) avec $\widehat{s}$ sa place complémentaire dans N_k, nous avons

alors :

$$\begin{aligned}
M''(s) &= M'(s) && (a)\\
&= M(s) - \sigma(\iota(s,t)) + \sigma(\iota(t,s)) && (b)\\
&= M_k(s) - \sigma(\iota(s,t)) + \sigma(\iota(t,s)) && (a)\\
&= M'_k(s) && (b)
\end{aligned}$$

$$\begin{aligned}
M''(\widehat{s}) &= k * \tau(s) - M'(s) && (a)\\
&= M_k(s) + M_k(\widehat{s}) - M'(s) && (c)\\
&= M(s) + M_k(\widehat{s}) - M'(s) && (a)\\
&= M(s) + M_k(\widehat{s}) - M(s) + \sigma(\iota(s,t)) - \sigma(\iota(t,s)) && (b)\\
&= M_k(\widehat{s}) - \sigma(\iota(\widehat{s},t)) + \sigma(\iota(t,\widehat{s})) && (a)\\
&= M'_k(\widehat{s}) && (b)
\end{aligned}$$

d'où $M'' = M'_k$. Les indications à droite des égalités signifient : (a) par définition de knet ; (b) par la règle de franchissement ; (c) par la proposition 5 et la définition de la validité d'un marquage.

(3) Nous pouvons utiliser une preuve similaire à celle du point précédent puisque M' est supposé k-borné. □

4 Temps causal dans les M-nets

Le concept de temps causal dans les réseaux de Petri a été présenté en introduction. Rappelons qu'il s'agit de représenter la progression du temps par le franchissement d'une transition d'un sous-réseau servant d'horloge. Cet événement devient alors la référence pour l'ensemble du réseau, tout comme la pulsation de son horloge constitue l'unique référence de temps pour un ordinateur.

Le premier but de ce chapitre est de montrer comment ce principe peut être appliqué dans le cadre compositionnel des M-nets. Nous donnons trois exemples d'horloges causales qui sont de plus en plus complexes et générales.

Nous montrons ensuite comment représenter facilement des systèmes à plusieurs horloges (ce qui correspond généralement au cas des systèmes répartis) et comment ces systèmes peuvent communiquer pour synchroniser leurs horloges.

Dans une troisième section, nous montrons que le concept de temps causal est parfaitement valide sous des hypothèses temps-réel. Pour cela, nous construisons une *machine d'exécution* dont le rôle est d'exécuter un réseau contenant une horloge causale dans un environnement temps-réel de telle manière que l'écoulement du temps causal et celui du temps réel coïncident.

4.1 Horloges

Grâce à l'utilisation des M-nets, construire et surtout utiliser une horloge causale se révèle généralement assez simple. Nous pouvons construire de nouvelles horloges à volonté afin de les adapter exactement aux systèmes à temporiser ; à l'opposé, nous pouvons construire des horloges très générales pouvant être utilisées dans tous les cas. Ce n'est pas notre propos que de trancher entre ces deux approches. En effet, construire une horloge pour chaque système peut se révéler fastidieux mais chaque horloge peut être ainsi parfaitement ajustée aux besoins de ce système ; au contraire, disposer d'une horloge générique est confortable mais son utilisation peut être contraignante car pas forcément adaptée à un cas particulier. Par ailleurs, une horloge générique est généralement complexe et nous pouvons être amenés à la simplifier pour des raisons d'efficacité.

4.1.1 Une horloge très simple

La première horloge qui nous intéresse permet d'exprimer des contraintes de temps entre deux transitions de la forme « au plus *max* et au moins *min* tics doivent survenir » dans laquelle la borne *max* est une constante. Cette horloge est présentée sur la figure 4.1 (rappelons que ω est plus grand que tout entier et que $\omega + 1 = \omega$).

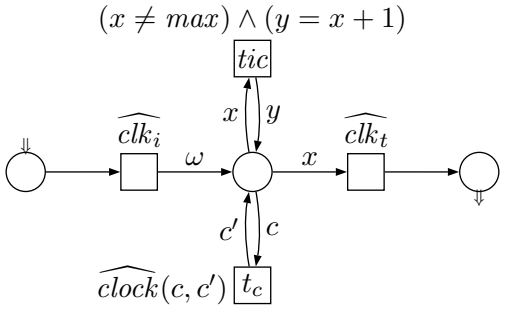


Figure 4.1 — Le M-net H_1 réalisant une horloge causale simple. Le type de la place centrale est l'ensemble $\{n \in \mathbb{N} \mid 0 \leq n \leq max\} \uplus \{\omega\}$.

L'utilisation de cette horloge est aisée : considérons à nouveau le M-net de la figure 1.8, page 22. Sa transition t_1 porte l'action $clock(c, 0)$ et peut donc se synchroniser avec la transition t_c de H_1 ; le franchissement de t_1 synchronisée à t_c met donc la valeur 0 dans la place centrale de H_1. Chaque franchissement de la transitions *tic* incrémente cette valeur, sauf si elle atteint *max* auquel cas la garde de *tic* devient fausse. Plus tard, la transition t_2, elle aussi synchronisée avec t_c, peut être franchie ; elle récupère alors dans c le nombre de tics compté depuis le franchissement de t_1 et peut l'exploiter dans sa garde (en l'occurrence, au moins un tic doit avoir été compté). Pour continuer le comptage initié par t_1, t_2 aurait dû porter l'action $clock(c, c)$ qui lui aurait permis de consulter le compteur sans le modifier. D'une manière générale, une action $clock(c, c')$ sert à récupérer le compte de tics dans c et à le positionner à c'.

Il est important de noter que le prédicat $x \neq max$ dans la garde de la transition *tic* prévient tout comptage au delà du maximum spécifié. Nous retrouvons là le couplage de l'horloge causale avec le système qu'elle temporise. En mettant ω dans la place interne de H_1 (au moyen d'une transitions étiquetée $clock(c, \omega)$ par exemple), nous laissons *tic* s'exécuter mais aucun comptage n'a lieu, et c'est cette valeur ω que nous récupérons dans c avec une autre action $clock(c, c')$.

Étant donné un M-net N_{sys}, nous devons le faire communiquer avec l'horloge et assurer l'initialisation et la terminaison de celle-ci. Nous utilisons pour cela les M-nets N_i et N_t présentés figure 1.9 (page 23) et le système complet correctement temporisé s'exprime simplement comme :

$$\Big((N_i; N_{sys}; N_t) \Big\| H_1 \Big) \,\mathsf{sc}\, \{clock, clk_i, clk_t\} \quad .$$

4.1.2 Compteurs multiples

La deuxième horloge que nous allons construire est capable de gérer plusieurs compteurs simultanément. Chaque compteur est associé à un identificateur (pris dans un ensemble I) et peut être consulté ou modifié indépendamment des autres. Par contre, tous les compteurs sont incrémentés simultanément lors du tic. Le compteur associé à l'identificateur $id \in I$ est borné par une constante m_{id} et nous définissons max comme le plus grand élément de l'ensemble $\{m_{id} \mid id \in I\}$. Cette horloge est donnée sur la figure 4.2, elle généralise légèrement celle que nous utilisons dans l'étude de cas du chapitre 5.

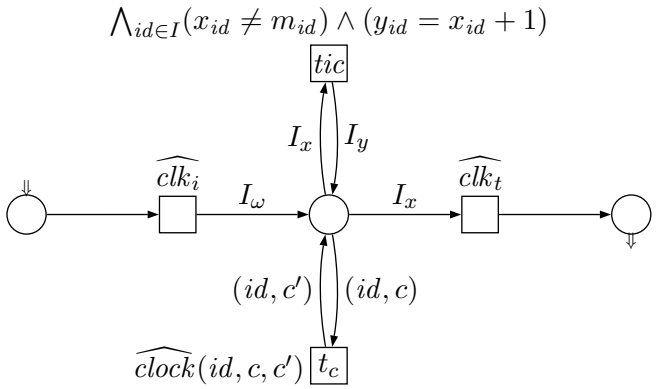


Figure 4.2 — Le M-net H_2 réalisant une horloge causale à plusieurs compteurs. Le type de la place centrale est $\{(id, i_{id}) \mid id \in I, i_{id} \in \{0, 1, \ldots, m_{id}, \omega\}\}$. De plus $I_\omega \stackrel{\text{df}}{=} \{(id, \omega) \mid id \in I\}$, $I_x \stackrel{\text{df}}{=} \{(id, x_{id}) \mid id \in I\}$ et $I_y \stackrel{\text{df}}{=} \{(id, y_{id}) \mid id \in I\}$; les x_{id} et y_{id} sont des variables deux à deux distinctes.

La place centrale de H_2 contient un ensemble de couples (id, c) où id est un identificateur et c le compteur de tics associé. La transition tic incrémente tous ces compteurs simultanément (les compteurs valant ω sont laissés à cette valeur, d'ailleurs, il ne peuvent empêcher le tic). Elle ne peut être franchie que si aucun des compteurs n'a atteint le maximum m_{id} qui lui est associé. La transition t_c est utilisée comme précédemment avec la différence que l'action $clock$ comporte un paramètre de plus pour l'identification du compteur.

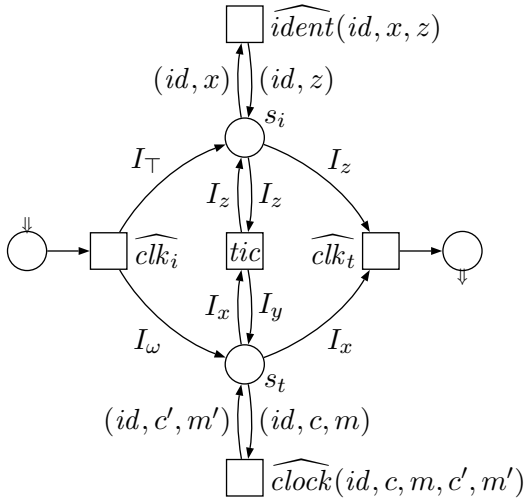
Nous obtenons un système complet à partir d'un M-net utilisant des actions sur $clock$ de la même façon que pour l'horloge H_1.

4.1.3 Contraintes dynamiques

Dans l'horloge précédente, le maximum de chaque compteur est fixé statiquement par la donnée des m_{id}. Nous définissons maintenant une horloge très générale permettant une définition dynamique de ces maximums ; elle est donnée figure 4.3. Puisque chaque compteur peut être utilisé pour n'importe quelle contrainte, cette nouvelle horloge va

aussi autoriser l'allocation dynamique des identificateurs de compteurs (toujours pris dans un ensemble I). Pour transmettre ces identificateurs entre les transitions partageant le même compteur, le moyen le plus simple est d'utiliser des liens asynchrones.

Compte tenu de l'existence des identificateurs, qui peuvent être alloués ou libérés, la valeur ω n'est plus utilisée pour identifier les compteurs non utilisé comme c'était le cas pour les horloges précédentes.



$$\gamma(tic) \stackrel{\text{df}}{=} \bigwedge_{id \in I} ((x_{id} < m_{id}) \vee (x_{id} = \omega) \vee (z_{id} = \top)) \wedge (y_{id} = x_{id} + 1)$$

$$\begin{aligned}
\tau(s_i) &\stackrel{\text{df}}{=} I \times \{\bot, \top\} \\
\tau(s_t) &\stackrel{\text{df}}{=} I \times \{0, \ldots, \omega\} \times \{0, \ldots, \omega\} \\
I_\top &\stackrel{\text{df}}{=} \{(id, \top) \mid id \in I\} \\
I_\omega &\stackrel{\text{df}}{=} \{(id, \omega, \omega) \mid id \in I\} \\
I_z &\stackrel{\text{df}}{=} \{(id, z_{id}) \mid id \in I\} \\
I_x &\stackrel{\text{df}}{=} \{(id, x_{id}, m_{id}) \mid id \in I\} \\
I_y &\stackrel{\text{df}}{=} \{(id, y_{id}, m_{id}) \mid id \in I\}
\end{aligned}$$

Figure 4.3 — Le M-net H_3 modélisant une horloge avec allocation dynamique. Les étiquettes sont données à droite ; les x_{id}, y_{id}, z_{id} et m_{id} sont des variables de *Var* toutes distinctes.

Nous pouvons reconnaître dans la partie basse de H_3 un réseau similaire à H_2. En fait, il s'agit presque du même réseau sauf que nous devons maintenant mémoriser dans la place s_t les maximums m_{id} pour chaque compteur. Dans l'horloge précédente, les m_{id} étaient des valeurs fixées ; ici, ce sont des variables du réseau. C'est aussi pour cette raison que l'action *clock* comporte plus de paramètres : en plus de l'identificateur, elle permet de lire le compteur dans c et de le positionner à c' mais aussi de lire le maximum courant dans m et de le changer en m'.

La partie haute de H_3 sert à gérer les identificateurs. Pour qu'une transition alloue un identificateur, il suffit qu'elle porte une action $ident(i, \top, \bot)$, ainsi, un identificateur libre $(id, \top)$ est choisi et il est marqué utilisé (avec $\bot$). Cet identificateur se trouve lié à la variable i par la synchronisation sur *ident*. Réciproquement, l'action $ident(i, \bot, \top)$ permet de libérer un identificateur préalablement alloué. Finalement, l'état d'un identificateur peut être consulté au moyen d'une action telle que $ident(id, x, x)$. Remarquons que l'allocation d'un identificateur peut avoir lieu simultanément à l'utilisation du compteur associé : il est tout à fait possible d'avoir sur la même transitions les actions $ident(i, \top, \bot)$ et $clock(i, c, m, 0, 5)$ par exemple ; cette combinaison alloue un nouvel identificateur qui se trouve lié à la variable i et positionne à 0 le compteur associé pour un maximum de 5 tics. De manière similaire, nous pouvons consulter un compteur et le libérer sur une seule

transition.

Remarquons finalement que le prédicat $z_{id} = \top$ dans la garde de *tic* permet de ne pas tenir compte des compteurs non alloués. Cela évite à l'utilisateur de l'horloge de devoir positionner à ω la valeur d'un compteur qu'il libère.

L'horloge H_3 comportant une action supplémentaire par rapport à la précédente, il faut en tenir compte lorsque cette horloge est ajoutée à un M-net N_{sys} qui l'exploite. Cette fois, le système complet est :

$$\Big((N_i; N_{sys}; N_t)\Big\|H_3\Big)\,\mathsf{sc}\,\{clock, ident, clk_i, clk_t\} \quad .$$

4.2 Horloges multiples

Nous avons vu comment le modèle causal du temps dans les M-nets permet de construire assez facilement différentes horloges, pour différents usages. Puisque le temps n'est pas une propriété ajoutée au modèle, il est très facile de modéliser des systèmes à plusieurs horloges, comme le sont la plupart des systèmes répartis. Ces horloges peuvent ensuite être laissées indépendantes ou bien synchronisées, directement au niveau du tic ou seulement au niveau des compteurs.

4.2.1 Horloges indépendantes

Étant donnés un M-net horloge H et un système N_{sys} faisant appel à H pour sa temporisation, nous notons $\mathsf{timed}(N_{sys}, H)$ le M-net composé du système et de l'horloge et dans lequel les communications ont été établies, ainsi que le démarrage et la terminaison de l'horloge lorsque c'est nécessaire. Il n'existe pas de définition générale de timed puisque cela dépend de chaque horloge (au niveau du démarrage, de la terminaison et des actions d'horloge), mais cette opération est toujours simple à définir si l'horloge est bien structurée. Par exemple, pour les trois horloges de la section précédente :

$$\mathsf{timed}(N_1, H_1) \stackrel{\text{df}}{=} \Big((N_i; N_1; N_t)\Big\|H_1\Big)\,\mathsf{sc}\,\{clock, clk_i, clk_t\} \quad ,$$

$$\mathsf{timed}(N_2, H_2) \stackrel{\text{df}}{=} \Big((N_i; N_2; N_t)\Big\|H_2\Big)\,\mathsf{sc}\,\{clock, clk_i, clk_t\} \quad ,$$

$$\mathsf{timed}(N_3, H_3) \stackrel{\text{df}}{=} \Big((N_i; N_3; N_t)\Big\|H_3\Big)\,\mathsf{sc}\,\{clock, ident, clk_i, clk_t\} \quad ;$$

où N_1, N_2 et N_3 sont trois M-nets utilisant les horloges H_1 (définie section 4.1.1), H_2 (section 4.1.2) et H_3 (section 4.1.3) respectivement ; les M-nets N_i et N_t sont donnés figure 1.9 (page 23).

Avec cette notation, nous pouvons envisager de composer arbitrairement des systèmes temporisés. Par exemple, pour $1 \leq j \leq k \in \mathbb{N}$, si N_j est un système utilisant une horloge H_j, nous pouvons utiliser $\mathsf{timed}(N_j, H_j)$ comme une version temporisée de N_j, par exemple, dans l'expression :

$$Sys \stackrel{\text{df}}{=} \mathsf{timed}(N_1, H_1) \| \cdots \| \mathsf{timed}(N_k, H_k) \quad ,$$

qui pourrait représenter des systèmes temporisés indépendamment évoluant en concurrence. Si les systèmes doivent communiquer, ils peuvent le faire au moyen de places de liens, ou avec des synchronisations de transitions, auquel cas nous pourrions avoir à encapsuler *Sys* dans une application de tie ou de sc. Le réseau obtenu pourrait encore être utilisé comme un système temporisé indépendant qui pourrait être intégré en tant que composant dans un système plus important encore.

Ce que nous mettons en avant ici est que la compositionnalité du modèle des M-nets est complètement conservée par l'utilisation d'horloges causales. Puisque ces dernières sont des parties des systèmes, et donc des M-nets « comme les autres », rien ne s'oppose à composer de manière arbitraire les systèmes causalement temporisés, la seule précaution à prendre étant d'encapsuler chaque système et son horloge par une application de l'opération timed appropriée.

4.2.2 Contrôle de la dérive

Par une modification simple des horloges d'un système en comportant plusieurs, nous pouvons contrôler leurs dérives respectives. Le mécanisme que nous donnons ici permet de limiter le décalage entre deux horloges quelconques, à un nombre de tics donné. Il permet uniquement synchroniser les horloges deux à deux, mais rien n'empêche qu'une horloge soit ainsi synchronisée à plusieurs autres.

Considérons deux horloges H_1 et H_2 dont les transitions de tics sont tic_1 et tic_2 respectivement. Le nombre d'occurrences d'une transition *tic* étant noté $\sharp tic$, la *dérive entre les deux horloges* est définie comme $|\sharp tic_1 - \sharp tic_2|$. À moins de forcer l'occurrence simultanée des deux tics (il suffit alors de les synchroniser sur une action de $\mathbb{A}$), cette dérive vaut en général 1 au minimum. Nous définissons l'entier $d_{1,2}$ comme la valeur maximale tolérée pour cette dérive.

Nous nous donnons ensuite deux symboles de liens, $b_{1,2}$ et $b_{2,1}$ dans $\mathbb{B}$; l'idée pour contrôler la dérive est que les places de liens associées contiennent chacune $d_{1,2}$ jetons, chaque occurrence de tic_1 consomme un jeton dans $b_{1,2}$ et le reproduit dans $b_{2,1}$, et symétriquement pour tic_2. Ainsi, en étant franchi, chaque tic donne à l'autre la possibilité d'être franchi. De plus chaque transition de tic ne peut pas être franchie plus de $d_{1,2}$ fois sans que l'autre ne le soit, ce qui assure le contrôle de la dérive. Afin que les réseaux obtenus soient sûrs, nous utiliserons des valeurs différentes pour tous ces jetons échangés.

La modification à apporter à chaque horloge est très simple : nous ajoutons juste les arcs entre sa transition de tic et les places de liens, comme indiqué sur la figure 4.4. La composition des deux systèmes temporisés assurera la fusion des places de liens leur servant à communiquer, comme montré au bas de la figure 4.4.

Il reste ensuite à initialiser le marquage des places de liens associées à $b_{1,2}$ et $b_{2,1}$, et à vider ces places lorsque le système contenant les deux horloges a terminé. l'initialisation est réalisée par le réseau donné en haut de la figure 4.5, celui au dessous permet de vider les places de liens lorsque le système a terminé.

Pour utiliser le mécanisme que nous venons de décrire, il faut encore encapsuler le système comprenant les deux horloges dans une construction similaire à timed permettant l'initialisation des places de liens de statuts $b_{1,2}$ et $b_{2,1}$. Supposons que le système en

question soit simplement la composition parallèle des deux sous-systèmes temporisés indépendamment : $\mathsf{timed}(N_1, H_1) \| \mathsf{timed}(N_2, H_2)$. Pour assurer le contrôle de la dérive, nous pouvons construire :

$$\Big(N_{1,2,i}\ ;\ \Big(\mathsf{timed}(N_1, H_1) \ \Big\|\ \mathsf{timed}(N_2, H_2)\Big)\ ;\ N_{1,2,t}\Big)\ \mathsf{tie}\ \{b_{1,2}, b_{2,1}\} \quad .$$

La restriction de liens permet de rendre privées les communications servant à contrôler la dérive entre les horloges.

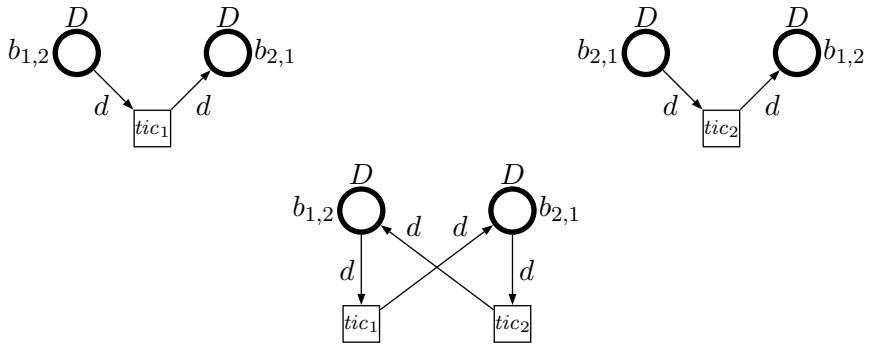


Figure 4.4 — Les modifications des horloges H_1 (en haut à gauche) et H_2 (en haut à droite) dont nous n'avons représenté que les transitions de tic. Nous définissons : $D \stackrel{\text{df}}{=} \{-d_{1,2}, \ldots, -1, 1, \ldots, d_{1,2}\}$. Nous supposons que d est une variable n'apparaissant pas dans les annotations de chaque transition de tic ni sur ses arcs environnants. Lorsque les réseaux contenant ces horloges sont composés, la fusion des places est effectuée comme dans le réseau du bas.

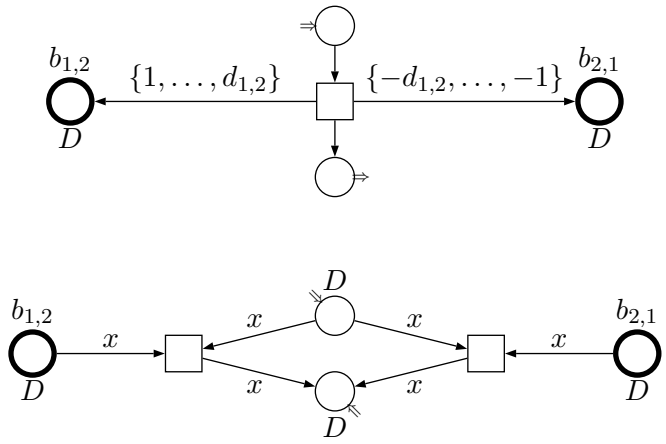


Figure 4.5 — En haut, le réseau $N_{1,2,i}$ assurant l'initialisation des places de liens ; en bas, $N_{1,2,t}$ qui assure leur terminaison (c'est-à-dire, qui les vide). Nous définissons là aussi $D \stackrel{\text{df}}{=} \{-d_{1,2}, \ldots, -1, 1, \ldots, d_{1,2}\}$.

4.2.3 Synchronisation de compteurs

Dans l'exemple précédent, nous avons synchronisé des horloges en contrôlant directement les tics. Cela revient à imposer des contraintes structurelles sur les oscillateurs des systèmes synchronisés. Ce genre de manipulations n'est généralement pas possible dans les systèmes répartis où les horloges sont physiquement indépendantes et ne peuvent pas être influencées dans leurs progressions. La solution pour synchroniser de tels systèmes est de changer la façon dont on prend en compte les tics au lieu de contraindre leurs occurrences.

Nous devons donc considérer un système à deux niveaux : d'une part, une horloge causale compte ses tics ; d'autre part un mécanisme de synchronisation communique avec une autre machine et modifie la valeur de ce compteur en fonction de la dérive qu'il constate. C'est le principe des systèmes de synchronisation tels que le standard NTP (*Network Time Protocol* [Mill89]). On y distingue un système « serveur », dont l'horloge est considérée fiable, et des systèmes « clients » qui mettent à jour leurs compteurs de tics en fonction de celui du serveur. Les systèmes sont reliés par des canaux de communication asynchrones, c'est-à-dire, sans garantie sur le temps de transmission. Dans ces conditions, la qualité de la synchronisation dépend de la capacité des canaux à communiquer rapidement les informations. De plus, le protocole NTP met en œuvre un système d'estampillage des messages échangés afin de tenir compte de cette spécificité des canaux de communications. La présentation détaillée de ce mécanisme complexe dépasse le cadre de notre exposé.

Pour toutes les horloges présentées dans les sections précédentes, on peut très simplement implanter la modification d'un compteur d'horloge au moyen de l'action *clock*. Par ailleurs, la modélisation des voies de communications peut être réalisée par des liens asynchrones ou par un M-net modélisant le réseau. Dans les deux cas, l'implantation d'un protocole tel que NTP ne pose pas de difficulté particulière puisque, pour toutes les horloges que nous avons présentées, le tic qui compte ses occurrences est séparé des transitions chargées de modifier ces compteurs. Le système à deux niveau (compteur de tic et mise à jour du compteur) est donc déjà en place.

4.3 Adéquation du temps causal et du temps réel

Nous avons jusqu'à maintenant admis l'hypothèse du monde clos : un réseau de Petri causalement temporisé s'exécute sans autre référence au temps que sa transition de tic. L'approche causale du temps reste en fait pertinente même dans un monde non clos. Pour le mettre en évidence, nous définissons une *machine d'exécution*, dotée de sa propre horloge (qui devient la référence de temps), dont le rôle est d'exécuter un réseau de Petri en garantissant la régularité des tics de l'horloge causale.

La notion de machine d'exécution est détaillée dans [Bou98] où l'auteur décrit des machines d'exécution pour les langages synchrones (en particulier pour Esterel [Ber98]). Le rôle d'une machine d'exécution est principalement de montrer comment obtenir un modèle concrètement exécutable à partir d'un modèle abstrait. Il faut pour cela résoudre

les difficultés liées aux aspects idéalisés du modèle de départ. Dans le cas des langages synchrones, les principales difficultés sont liées à *l'hypothèse synchrone* qui considère que toute réaction est simultanée à l'occurrence du stimulus qui la déclenche, ce qui suppose une exécution sur une machine infiniment rapide. La solution à ce problème passe par la construction d'un automate fini agrégeant en une seule transition les cascades d'actions/ réactions concentrées en un point du temps. Par exemple, en notant $e_1 \Rightarrow e_2$ le fait que l'événement e_1 entraîne en réaction l'événement e_2, si le système spécifie $e_1 \Rightarrow e_2$ et $e_2 \Rightarrow e_3$, nous aurons une transition portant les trois événements e_1, e_2 et e_3 dans l'automate représentant ce système. Ces événements sont tous dits *présents* à un instant donné, peu importe qu'ils aient été reçus ou émis. L'instantanéité de la réaction entraîne des difficultés pour la construction de l'automate d'un programme réactif. Par exemple, si le système spécifie $\neg e_1 \Rightarrow e_1$ (ce qui se traduit par « si e_1 n'est pas présent, on émet e_1 ») implique que e_1 est présent si et seulement si il est absent. Un tel système est dit *non réactif* et n'est pas autorisé. De même, un système comportant $e_1 \Rightarrow e_1$ ne permet pas de décider si e_1 doit être présent ou non ; il est dit *non déterministe* et n'est pas autorisé non plus. D'autres paradoxes peuvent encore être mis à jour, ces questions sont discutées en détails dans [Ber98].

Dans notre cas, nous devons aussi rejeter les systèmes non réalistes, par exemple ceux qui peuvent avoir des évolutions arbitrairement longues en des durées de temps fixées.

Un autre rôle important d'une machine d'exécution est de permettre les *entrées/ sorties* entre le modèle exécuté et l'environnement dans lequel nous le plongeons (celui de la machine). La machine permet donc d'interfacer le modèle au monde réel. Cela suppose en général de résoudre d'autres difficultés que nous mettons en évidence dans la suite.

Pour réaliser les communications, notre machine d'exécution est dotée de *ports d'entrée* et de *ports de sortie* permettant respectivement de recevoir des données de l'environnement ou de lui en envoyer. Chaque communication sur un port d'entrée peut se voir acceptée ou refusée par la machine ; cela permet de signaler à l'environnement que sa demande a pu ou non être prise en compte. La machine a aussi la possibilité de signaler une erreur lorsque le réseau pouvait lire une valeur différente de celle présentée sur un port d'entrée.

Hypothèses. Pour définir notre machine, nous supposons que les réseaux à exécuter ne comportent qu'une transition de tic. Il ne s'agit pas d'une restriction mais d'une simplification. En fait, nous pourrions considérer des réseaux à plusieurs horloges (et donc plusieurs tics), mais puisqu'elles doivent être exécutées toutes au même rythme, nous pouvons tout aussi bien les identifier directement au niveau du réseau. Par ailleurs, nous interdisons à la transition de tic d'être franchie plusieurs fois dans un même step, y compris avec des valuations différentes. Cette condition, très faible et naturelle, permet d'assurer que l'horloge se comporte comme telle.

Par ailleurs, nous nous limitons aux réseaux *tic-bornés* dans lesquels le nombre de steps non vides franchissables entre deux steps contenant un tic est borné. En effet, pour toute machine, il existe une limite à la quantité de traitements qu'elle peut exécuter un

laps de temps donné. Il s'agit donc d'une restriction naturelle.

De même, nous ne considérons que des réseaux de Petri finis et bornés. Là encore, ce choix est naturel dès que nous considérons une machine aux ressources limitées. En particulier, des marquages non bornés ne sont pas représentables dans une telle machine.

Organisation de cette section. Afin d'aboutir à la définition complète de notre machine d'exécution, nous commençons par définir une classe de réseaux colorés correspondant aux M-nets bornés et tic-bornés dans lesquels nous distinguons notamment une transition de tic. Ces réseaux sont appelés des *tic-M-nets*, ou plus simplement des *tM-nets*.

Nous montrons ensuite comment le comportement d'un tM-net peut être abstrait en un automate ne représentant que les informations nécessaires à l'exécution du réseau (compte tenu des points d'entrée/sortie que nous avons fixés).

Finalement, nous montrons comment exécuter une telle abstraction d'un tM-net en assurant la régularité des tics d'horloge.

4.3.1 Tic-M-nets (tM-nets)

Soit (N, M) un M-net marqué borné, avec $N = (S, T, \iota)$. Nous appelons *tic* la transition de T dont l'occurrence constitue le tic de l'horloge causale de N ; de plus, nous notons $\mathsf{tic} \stackrel{\text{df}}{=} \{\sigma(tic) \mid \sigma \text{ est une valuation légale de } tic\}$ l'ensemble des occurrences potentielles de *tic*.

Un *tic-step* de (N, M) est un step U tel que $M[U\rangle$ et $U \cap \mathsf{tic} \neq \varnothing$. (Remarquons que, puisque N ne comporte qu'une transition de tic, U n'en contient qu'une occurrence.) Le M-net marqué (N, M) est un *tM-net* s'il existe un entier δ tel que, pour tout marquage M' accessible à partir de (N, M), toute séquence de steps dans $\mathsf{steps}(N, M')$ comportant au moins δ steps non vides contient au moins deux tic-steps. Autrement dit, aucune exécution d'un tM-net ne franchit δ steps non vides sans rencontrer au moins deux tics d'horloge. Cet entier δ est *la distance maximale entre deux tics* du tM-net.

Entrées/sorties. Les communications d'un tM-net avec son environnement sont réalisées sur la base des actions présentes sur les transitions (il s'agit donc d'actions non restreintes). Étant donné un tM-net, nous distinguons un ensemble $\mathbb{O} \subseteq \mathbb{A}$ *d'actions de sorties* (*output*) permettant à l'environnement d'observer l'exécution du tM-net, et un ensemble $\mathbb{I} \subseteq \mathbb{A}$ *d'actions d'entrée* (*input*) permettant au tic net de recevoir des informations de son environnement. Nous supposons $\mathbb{O} \cap \mathbb{I} = \varnothing$.

Nous supposons de plus que toute action $a \in \mathbb{O} \uplus \mathbb{I}$ est d'arité $\mathsf{ar}(a) = 1$. Cette hypothèse permet de simplifier la suite sans perte de généralité : nous pouvons simuler une action n-aire par n actions unaires (puisque ces actions ne servent pas à la synchronisation) ; d'autre part, là où nous souhaiterions une action sans paramètre, nous pouvons nous contenter d'ignorer le paramètre d'une action unaire.

Les actions permettent les échanges de données entre un tM-net et l'environnement dans lequel il est exécuté. Nous restreignons les données aux valeurs d'un ensemble $\mathit{Data} \subseteq \mathit{Val}$. Cet ensemble peut être arbitrairement grand ; nous le distinguons pour

attirer l'attention sur le fait qu'il est plus réaliste de le choisir petit. En effet, les actions d'entrée et de sortie peuvent être considérées comme des ports de communication de la machine d'exécution dont il convient de limiter la taille si nous souhaitons réaliser une version concrète de la machine.

Lors de l'exécution d'un tM-net, chaque transition exécutée portant une action $a(v)$ telle que $a \in \mathbb{O}$ donne lieu à l'écriture de v sur le port de sortie qui correspond à a. Réciproquement, lorsqu'un port d'entrée de la machine (correspondant à $a \in \mathbb{I}$) reçoit une valeur v, la machine doit favoriser le franchissement d'une transition portant une action $a(x)$ pour une valuation σ telle que $\sigma(x) = v$.

En toute généralité, nous ne pouvons pas garantir qu'une telle transition existe. Même lorsque c'est le cas, nous ne pouvons pas garantir qu'elle sera franchissable dans ces conditions dans un « futur proche ». Cependant, la figure 4.6 montre comment obtenir simplement un cas favorable dans lequel le réseau peut à tout moment franchir une transition permettant de satisfaire une demande de l'environnement. Dans un tel cas, le réseau est dit *réactif* par rapport à l'action a.

Une transition t d'un tM-net est dite *réactive* par rapport à une action $a \in \mathbb{I}$ si les conditions suivantes sont réunies :

- t porte une action $a(x)$;
- pour toute valeur $v \in \mathit{Data}$, il existe une valuation σ qui rend t franchissable et telle que $\sigma(x) = v$;
- il existe une exécution maximale du tM-net (qui ne termine jamais ou aboutit à un blocage) qui ne franchit jamais t.

Une telle transition est donc toujours franchissable pour toutes les valeurs du paramètre de a mais il n'est jamais nécessaire de le faire.

Cette réactivité ne vaut que pour le marquage courant du tM-net. Nous pouvons généraliser ce point de vue au tM-net en entier. Un tM-net est dit *réactif* par rapport à l'action a si pour tous ses marquages accessibles, l'une des conditions suivantes est réalisée :

- il existe une séquence de δ steps au plus, contenant au plus un tic-step, qui permet de franchir une transition réactive par rapport à a (où δ est la distance maximale entre deux tics) ;
- toute exécution du tM-net aboutit à un blocage.

Le premier cas correspond à la situation où le réseau est « à l'écoute » de son port d'entrée. Dans le second cas, le tM-net est en train de terminer son exécution et ce port est fermé, plus aucune communication n'est alors possible dessus.

Exiger la réactivité serait une restriction, mais comme nous le voyons sur la figure 4.6, elle est très facile à obtenir. Nous pouvons considérer que des fragments de réseaux tels que celui de cette figure correspondent dans la modélisation d'un système à la représentation d'un de ses ports d'entrée. La machine d'exécution sera conçue de façon à garantir des réponses efficaces aux sollicitations de l'environnement lorsqu'une transition réactive est disponible pour cette communication.

La réactivité est le cas idéal. Il peut arriver que plusieurs communications soient demandées par l'environnement et que le tM-net ne puisse pas toutes les prendre en

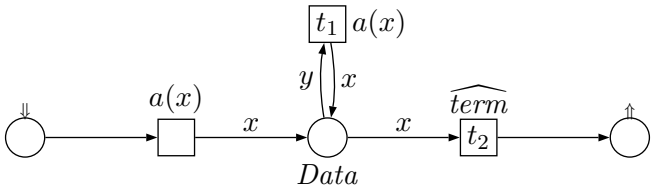


Figure 4.6 — Un tM-net contenant ce fragment est réactif par rapport à l'action a (en supposant que a n'apparaît pas ailleurs dans le réseau et que la transition t_2 n'est franchie qu'à la fin de l'exécution du réseau). Pour que ce fragment soit utilisable, il faudrait ajouter une transition pouvant consulter la valeur contenue dans la place interne (par une boucle à la manière de t_1), elle entrerait en conflit avec t_1 mais cela n'empêcherait pas la réactivité puisque le réseau peut toujours franchir t_1 en premier.

compte. Pour régler de tels conflits, nous ordonnons les actions de $\mathbb{I}$ de façon à toujours favoriser les actions les plus grandes selon cet ordre. Ce mécanisme est similaire à la façon dont un microprocesseur prend en compte ses *interruptions* ; il est cependant plus général puisque plusieurs sollicitations peuvent être considérées simultanément.

4.3.2 Compilation des tM-nets

Nous montrons maintenant comment transformer un tM-net en automate fini non déterministe possédant les mêmes exécutions (en termes d'actions d'entrées/sorties visibles). C'est cet automate qui sera en fait exécuté par la machine. Intuitivement, les états de l'automate sont des marquages du tM-net et chacune de ses transitions correspond à une séquence de steps du tM-net permettant de passer d'un marquage à l'autre. Ces séquences de steps sont telles que seul le premier step est un tic-step et nous ne retenons dans l'étiquette de la transition que les actions de communications. Ainsi, exécuter une transition de l'automate revient à franchir une séquence de steps dans le tM-net de telle façon que nous ne pouvons franchir qu'un tic, au début de la séquence. L'état initial de l'automate correspond au marquage initial du tM-net. Il n'est pas garanti que le réseau puisse immédiatement exécuter un tic depuis cet état, c'est pourquoi nous autorisons la première transition de l'automate d'un tM-net à ne pas comporter de tic.

Soit (N, M) un tM-net avec *tic* sa transition de tic, $\mathbb{O}$ ses actions de sortie et $\mathbb{I}$ ses actions d'entrée. Avant toute chose, nous éliminons des étiquettes de N toutes les actions de $\mathbb{A} \setminus (\mathbb{I} \cup \mathbb{O})$. Ainsi, compte tenu de la restriction sur l'auto-concurrence des actions d'entrée/sortie, un step U est un ensemble de transitions valuées de la forme $\sigma(t)$ (et non un multi-ensemble). Nous notons alors :

$$\lambda(U) \stackrel{\text{df}}{=} \bigcup_{\sigma(t)\in U \wedge a(x)\in\lambda(t)} a(\sigma(x)) \quad .$$

Nous construisons l'automate $A(N, M) \stackrel{\text{df}}{=} (S_A, T_A, \mathit{init}, \lambda_A)$ où S_A est l'ensemble des états, $T_A \subseteq S_A \times S_A$ celui des transitions, $\mathit{init} \stackrel{\text{df}}{=} M \in S_A$ est l'état initial et λ_A est une

fonction d'étiquetage des transitions. Nous ne distinguons pas d'états terminaux car ils le sont tous. Les éléments S_A, T_A et λ_A sont définis ci-dessous.

À partir de l'état initial, nous cherchons d'abord à atteindre un état d'où nous pourrons franchir un tic immédiatement. S'il existe un marquage M' de N et des steps U_1, …, U_k, tels que les conditions suivantes sont réunies :

- seul U_k est un tic-step ;
- $M[U_1 \cdots U_{k-1}\rangle M'[U_k\rangle$;
- chaque $a \in \mathbb{O} \cup \mathbb{I}$ apparaît au plus une fois dans chaque U_i, pour $i < k$;

alors $M' \in S_A$, $(M, M') \in T_A$ et $\lambda_A(M, M') \stackrel{\text{df}}{=} (\lambda(U_1), \ldots, \lambda(U_{k-1}))$. Remarquons que, par définition, $k \leq \delta$. La troisième condition élimine les steps où il faudrait réaliser simultanément plusieurs communications sur le même port. Il ne s'agit pas d'une contrainte puisque si un tel step existe, ses linéarisations sont disponibles et peuvent respecter la condition.

À partir d'un état quelconque $M' \neq M$, un tic peut donc être exécuté immédiatement. Nous construisons alors les transitions permettant d'atteindre un état similaire. S'il existe un marquage M'' de N et des steps U_1, …, U_k, tels que les conditions suivantes sont réunies :

- seuls U_1 et U_k sont des tic-steps ;
- $M'[U_1 \cdots U_{k-1}\rangle M''[U_k\rangle$;
- chaque $a \in \mathbb{O} \cup \mathbb{I}$ apparaît au plus une fois dans chaque U_i, pour $i < k$;

alors $M'' \in S_A$, $(M', M'') \in T_A$ et $\lambda_A(M', M'') \stackrel{\text{df}}{=} (\lambda(U_1), \ldots, \lambda(U_{k-1}))$. Remarquons, que là encore, par définition, $k \leq \delta$. Cette deuxième étape est répétée jusqu'à ce que plus rien ne soit ajouté à l'automate.

Nous devons finalement ajouter les transitions permettant de terminer l'exécution. S'il existe un marquage M'' de N et des steps U_1, …, U_k, tels que les conditions suivantes sont réunies :

- seul U_1 est un tic-step ;
- $M'[U_1 \cdots U_k\rangle M''$;
- M'' ne rend aucune transition franchissable ;
- chaque $a \in \mathbb{O} \cup \mathbb{I}$ apparaît au plus une fois dans chaque U_i, pour $i \leq k$;

alors $M'' \in S_A$, $(M', M'') \in T_A$ et $\lambda_A(M', M'') \stackrel{\text{df}}{=} (\lambda(U_1), \ldots, \lambda(U_k))$. Cette fois, nous avons $k < \delta$ car U_k n'est pas un tic-step.

Par construction, une exécution de l'automate correspond toujours à une exécution du tM-net d'où il est issu. En particulier, tous les états sont des marquages accessibles et les transitions entre les états correspondent à des suites de steps franchissables dans le tM-net. Réciproquement, si le tM-net exécute une suite de steps, l'automate correspondant peut exécuter une série de transitions qui représente cette suite de steps, sous réserve que les conditions que nous avons posées soient respectées (notamment sur la non concurrence des communications sur le même port).

4.3.3 Exécution des tM-nets

L'exécution de l'automate d'un tM-net est relativement simple. La machine d'exécution tire des transitions et les exécute. Plus précisément, étant donné M' l'état courant de l'automate, la machine exécute des cycles dont chacun correspond aux étapes suivantes :

1. Mémoriser les demandes de communications sur ses ports d'entrée.

2. Choisir une transition $(M', M'') \in T_A$ qui maximise le nombre d'actions d'entrée qui peuvent être servies (le critère de maximalité est défini ci-dessous). Cette transition porte l'étiquette $(\lambda(U_1), \ldots, \lambda(U_k))$.

3. Pour chaque valeur de i entre 1 et k (dans l'ordre) et pour chaque action $a(v) \in \lambda(U_i)$ (dans un ordre aléatoire) :
   - si $a \in \mathbb{O}$, écrire v sur le port de sortie a ;
   - si $a \in \mathbb{I}$ et que v est la valeur présente sur le port d'entrée a, marquer la communication sur a « acceptée » ;
   - si $a \in \mathbb{I}$ mais que v diffère de la valeur demandée sur le port ou qu'aucune valeur n'y est demandée, marquer la communication sur a « en erreur ».

4. Marquer « refusée » chaque communication mémorisée à l'étape 1 qui n'a pas déjà été marquée « acceptée » ou « en erreur » à l'étape 3.

5. Mémoriser le nouvel état de l'automate et attendre le début du cycle suivant.

Nous observons que la durée des cycles peut être bornée et donc fixée puisque toutes les étapes sont bornées : le nombre de ports, le nombre de transitions parmi lesquelles choisir, le nombre de steps à exécuter et le nombre d'actions dans chaque step (puisque les réseaux sont finis). La transition de tic a disparu de l'automate, mais nous nous sommes assurés qu'elle était franchie dans le premier step de chaque transition de l'automate, elle est donc virtuellement exécutée au début de chaque cycle de la machine (sauf éventuellement le tout premier) et donc ses occurrences sont régulières puisque les cycles ont une durée fixe.

Il nous reste à définir le critère de choix d'une transition. Le but est de servir un maximum de ports d'entrée tout en minimisant le nombre d'erreurs. Lorsque plusieurs choix sont possibles, nous faisons appel à l'ordre sur les actions pour choisir. La stratégie de choix peut être décrite de la façon suivante : pour chaque action a_i, de la plus grande à la plus petite selon l'ordre sur $\mathbb{I}$,

- si une communication est demandée sur a_i et qu'elle peut être servie par une transition de l'automate, éliminer toutes les transitions qui ne permettent pas de servir la communication sur a_i ;
- si aucune communication n'est demandée sur a_i et qu'il existe une transition ne comportant pas d'action a_i, éliminer toutes les transitions qui servent a_i.

Cela revient à servir les communications demandées par ordre de priorité. Lorsqu'une communication n'est pas demandée, nous évitons de plus de mettre en erreur les ports prioritaires plutôt que de servir une communication de moindre priorité.

D'autres stratégies peuvent être envisagées, mais en toute généralité, il n'en existe pas qui soit satisfaisante quand le tM-net est arbitraire. Cependant, nous pouvons facilement constater que la stratégie choisie ici garantit que si un tM-net est réactif par rapport à une action, celle-ci est toujours servie et n'est jamais mise en erreur (sauf éventuellement lors du dernier cycle de la machine qui correspond à son arrêt). Comme la réactivité est facile à obtenir, nous considérons que notre stratégie est satisfaisante.

5 Efficacité de l'approche causale du temps

Ce chapitre s'attache à l'étude de la pertinence et de l'efficacité de l'approche causale du temps en pratique. Pour cela, nous étudions un exemple de passage à niveau pour lequel nous réalisons plusieurs spécifications que nous vérifions avec différents outils logiciels. Les résultats montrent que l'utilisation d'une horloge causale peut s'avérer plus efficace que l'utilisation d'un modèle intégrant le temps de façon explicite ; cependant, cette efficacité est dépendante de la taille des constantes utilisées comme contraintes temporelles, ce qui devra être résolu à l'avenir. Nous constatons aussi que notre approche est assez simple à mettre en œuvre et plutôt intuitive.

5.1 Un passage à niveau

Nous considérons un système de passage à niveau ayant les caractéristiques suivantes :

- il comporte n voies parallèles ; sur chacune d'elle peut circuler un train ;
- l'approche d'un train déclenche un signal *app* ; lorsque le train s'éloigne, un capteur émet le signal *exit* ; ces signaux ne sont pas différenciés selon les trains ;
- les voies croisent une route gardée par une paire de barrières, celle-ci peut recevoir deux signaux, *down* et *up*, qui commandent respectivement sa fermeture et son ouverture ;
- lorsque le signal *app* est levé, le train met ensuite entre a_m et a_M unités de temps pour atteindre le niveau des barrières ; il met ensuite entre e_m et e_M unités de temps pour déclencher le signal *exit* ;
- les barrières mettent entre g_m et g_M unités de temps pour se fermer ou s'ouvrir.

Le problème est de construire un contrôleur dont le rôle est de lire les signaux résultant du passage des trains devant leurs capteurs afin de commander l'ouverture et la fermeture des barrières, tout en respectant les contraintes suivantes :

- le contrôleur réagit au minimum en c_m et au maximum en c_M unités de temps ;
- la condition de *sécurité* est toujours vérifiée : si un train se trouve entre les barrières, alors celles-ci sont forcément fermées ;
- le système n'a pas de blocage.

Pour une spécification satisfaisante, nous devrions aussi ajouter une condition de *disponibilité* spécifiant que si aucun signal *app* n'a eu lieu depuis un certain temps, les barrières doivent être ouvertes. On pourrait la vérifier au moyen de formules de logique temporelle ou bien par l'ajout de réseaux (et d'automates temporisés) de test permettant de détecter son non respect lors de l'exécution. Notre étude de cet exemple n'a pas pour but de spécifier un bon système mais de vérifier la pertinence et l'efficacité de l'approche causale. Nous nous en tenons donc à cette version minimale.

5.2 Modèles et outils considérés

La modélisation du passage à niveau peut être faite avec une multitude de modèles, nous avons choisi de nous limiter, autant que possible, aux plus répandus et de vérifier les spécifications avec des outils ayant fait leur preuves.

5.2.1 M-nets avec temps causal

L'approche causale a été implantée à base de M-nets en utilisant l'outil PEP [GB96, Gra97] pour la partie modélisation et MARIA [Mäk99] pour la vérification.

PEP est un outil tout particulièrement dédié aux M-nets et aux modèles de cette famille. Il regroupe dans une interface graphique différents éditeurs, des convertisseurs entre les modèles et des outils de vérification. PEP a été dès le départ le choix évident pour cette étude. Pourtant, nous n'avons pas pu l'utiliser du début à la fin. En effet, les outils de vérification proposés avec PEP travaillent sur des réseaux places/transitions et PEP fait appel au dépliage pour permettre la vérification des M-nets. Nous avons constaté que notre horloge causale était trop complexe pour être dépliée par PEP. Ce problème a été signalé aux responsables du projet et des travaux sont en cours pour améliorer le déplieur de PEP et lui permettre de traiter des réseaux plus complexes.

En attendant que ces développements aboutissent, nous avons dû nous tourner vers un autre outil pour la vérification. Le dépliage s'avérant limité, nous avons cherché un outil permettant de travailler sur les réseaux colorés que sont les M-nets. MARIA est un outil assez récent qui permet de générer des graphes de marquages de réseaux colorés et de réaliser en cours de génération (« à la volée ») des tests d'accessibilité, de chercher les blocages et de vérifier des formules de logique temporelle.

Nous avons donc réalisé la spécification avec PEP puis nous avons traduit les M-nets qu'il produisait en réseaux colorés adaptés à MARIA. À présent, nous pouvons générer et vérifier de nombreuses variantes de la spécification en nous passant totalement de PEP, mais celui-ci s'est toutefois révélé très utile pour aboutir à une spécification en nous donnant accès aux compositions de M-nets.

5.2.2 Automates temporisés

Compte tenu de la popularité de ce modèle, il nous a semblé nécessaire d'utiliser les automates temporisés. Il en existe de nombreuses versions : nous avons choisi de nous

limiter à celles utilisées par les outils UPPAAL [Lar+97] et Kronos [Yov97] qui sont considérés comme deux des meilleurs outils disponibles.

La réalisation de la spécification pour UPPAAL a été assez directe ; en effet, cet outil dispose d'une interface graphique très maniable et efficace et nous avons pu rapidement aboutir à une spécification utilisable et vérifiable. Comme dans le cas précédent, nous sommes capables de générer facilement de nombreuses variantes de notre spécification.

L'utilisation de Kronos s'est avérée plus problématique. Cet outil est efficace mais le format dans lequel on doit lui fournir les automates temporisés est de très bas niveau et on ne dispose pas d'interface graphique pour s'affranchir de l'édition de ces fichiers. En plus de cela, Kronos suppose une synchronisation de toutes les transitions portant une même action, à la place d'une synchronisation binaire comme dans UPPAAL (et dans les M-nets). Cette approche s'est révélée particulièrement malcommode dans le cas du système de passage à niveau et il nous a fallu multiplier les noms d'actions pour avoir des synchronisations satisfaisantes. La définition d'un système non trivial en utilisant Kronos est donc une opération très délicate et les risques d'erreurs sont importants.

5.2.3 Réseaux de Petri temporisés

Nous nous sommes aussi penchés sur les différents modèles de réseaux de Petri temporisés. Cependant, nous avons été surpris de ne pas trouver d'outil nous permettant d'implanter correctement une spécification de notre système.

Pour commencer, nous avons considéré PEP qui est capable dans ses versions récentes de vérifier des réseaux temporisés (*timed Petri nets*, dans lesquels on assigne aux transitions un intervalle de temps durant lequel elles doivent être franchies). Cependant, PEP ne permet pas de spécifier des transitions sans intervalle de temps ou avec un intervalle $[0, \infty]$. Cette limitation a été signalée aux concepteurs de PEP qui travaillent actuellement à la lever.

Un appel à la communauté réseaux de Petri, par la liste de diffusion *Petri Nets* (voir `http://www.daimi.au.dk/PetriNets/mailing-lists`), nous a amené à considérer deux autres outils : INA [RS01] et DesignCPN [Jan+99].

Le premier s'est avéré extrêmement difficile à utiliser : il dispose d'une interface particulièrement rudimentaire et, plus grave dans notre cas, les fichiers sur lesquels il travaille sont absolument illisibles. Nous n'avons donc pas pu générer différentes versions de notre spécification ; il a fallu toutes les créer indépendamment. De plus, avec INA, la vérification doit se faire en direct par manipulation de l'interface. Ces deux défauts empêchent toute tentative de lancer des tests en série pour observer les performances. Nous avons donc dû nous contenter de vérifier quelques exemples. Dans tous les cas, INA s'est révélé largement moins efficace que MARIA ; c'est pourquoi nous avons décidé de ne pas le prendre en considération plus avant.

DesignCPN propose une interface graphique qui permet d'éditer simplement des réseaux colorés temporisés. Cependant, le modèle proposé permet de tester l'âge des jetons, mais il n'offre aucun moyen de forcer le franchissement d'une transition. Il a donc été impossible de modéliser notre système de passage à niveau faute de pouvoir spécifier les contraintes de temps que les différents éléments doivent respecter structurellement.

Pour le moment, les tentatives de comparaison avec un modèle de réseaux de Petri temporisé sont donc suspendues, jusqu'à ce que PEP (ou un autre outil) soit capable de répondre à nos besoins. Ce constat est décevant mais il a au moins un aspect positif : puisque nous ne nous comparons pas à des modèles proposant des sémantiques de vraie concurrence, nous pouvons simplifier la spécification et nous limiter à un contrôleur purement séquentiel, ce qui résulte dans l'entrelacement des événements qui auraient pu être concurrents. Non seulement cela simplifie la spécification, mais en plus, cela renforce la pertinence de la comparaison avec les automates temporisés qui ont une sémantique d'entrelacement.

Nous avons l'intention de faire dès que possible la comparaison de notre approche avec des réseaux de Petri temporisés et dans ce cas, nous prendrons soin de définir une spécification la plus concurrente possible ce qui devrait améliorer les performances des outils qui en tirent parti (notamment PEP qui travaille sur les réseaux d'occurrences).

5.3 Modélisations

5.3.1 Automates temporisés

La figure 5.1 présente la modélisation d'une des voies parallèles et celle des barrières au moyen d'automates temporisés. Les automates présentés ici peuvent être directement implantés pour UPPAAL ; afin de les tester avec Kronos, un travail d'adaptation est nécessaire et nous obtenons des automates plus complexes qu'il ne nous paraît pas utile de présenter ici.

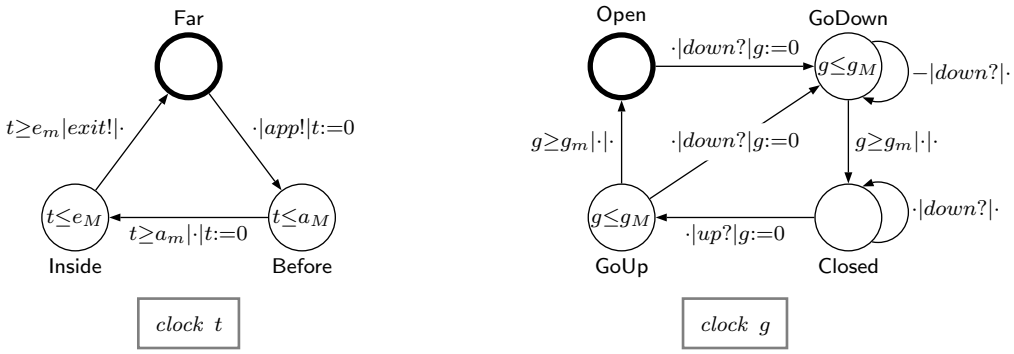


Figure 5.1 — À gauche, l'automate *Train*, modélisant une voie et le train associé, qui utilise un chronomètre t. À droite, l'automate *Gates*, représentant les barrières ; il utilise un chronomètre g.

L'automate pour une voie démarre dans l'état Far (les états initiaux sont dessinés en gras). Lorsqu'un train passe au niveau du premier capteur, l'automate passe dans l'état Before en émettant le signal *app*! et en remettant à zéro le chronomètre t. Il peut rester dans cet état tant que $t \leq a_M$; à $t = a_M$ au plus tard, il doit passer dans l'état Inside qui modélise le fait que le train se trouve entre les barrières ; cette transition n'est possible

que si $t \geq a_m$ et remet t à zéro sans émettre aucun signal. De même, l'automate passe à nouveau à l'état Far pour $t \in [e_m, e_M]$ en émettant le signal *exit*!.

Le fonctionnement de l'automate spécifiant les barrières est similaire. Remarquons que cet automate peut recevoir un signal *down*? depuis n'importe lequel de ses états. Par contre, il ne peut recevoir *up*? que lorsque la barrière est fermée. Le nom des états indique sans ambiguïté ce qu'ils représentent au niveau du système concret. Ceci n'est pas réaliste du point de vue de la modélisation mais constitue plutôt une propriété qu'on devrait vérifier. Dans ce cas et dans celui du contrôleur, certains signaux incontrôlables ne sont pas pris en compte dans tous les états (alors qu'ils devraient l'être). Nous avons fait le choix de cette version peu réaliste afin de garder une spécification simple nous permettant d'illustrer notre propos sans alourdir l'aspect technique.

Le contrôleur doit tenir compte des spécificités de la barrière au niveau de la réception des signaux. Son fonctionnement est assez simple : chaque fois qu'il reçoit un signal *app*? (qui se synchronise avec le *app*! émis par un automate *Train*), il incrémente un compteur et envoie un signal *down*! aux barrières dans un délai compris dans $[c_m, c_M]$. Lorsqu'il reçoit un signal *exit*?, le contrôleur décrémente la même variable, si elle tombe à zéro, le dernier train vient de quitter les barrières et un signal *up*! leur est envoyé. La figure 5.2 donne l'automate modélisant le contrôleur.

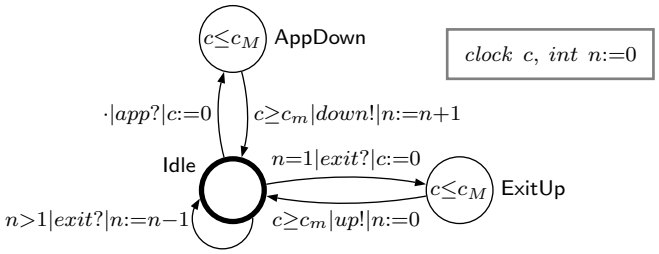


Figure 5.2 — L'automate temporisé *Controller* représentant le contrôleur. Il utilise une variable entière n initialisée à zéro et un chronomètre c.

Le système complet est obtenu comme le produit synchronisé de n copies de l'automate *Train* (nommées $Train_i$, pour $1 \leq i \leq n$) et des automates *Gates* et *Controller*. Nous obtenons un automate permettant l'exécution entrelacée de tous ceux qui le composent avec synchronisation sur les actions telles que *app*! et *app*?.

La propriété de sécurité et d'absence de blocage peut s'exprimer par une formule de logique temporelle (dans une variante de CTL) :

$$\forall\Box\left(\neg\,\mathsf{deadlock}\right) \wedge \Big(\big(\mathit{Train}_1.\mathsf{Inside} \vee \cdots \vee \mathit{Train}_n.\mathsf{Inside}\big) \Rightarrow \mathit{Gates}.\mathsf{Closed}\Big) \quad ,$$

qui peut se traduire ainsi : « Il est toujours vrai que le système n'est pas bloqué et que si l'un des automates $Train_i$ est dans l'état Inside alors l'automate *Gates* est dans l'état Closed. » La formulation pour Kronos est similaire.

5.3.2 M-nets

La spécification avec des M-nets est calquée sur celle avec les automates temporisés donnée plus haut. Pour chaque état, nous aurons une place (un jeton dans celle-ci correspondant à l'activation de l'état) et pour chaque transition de l'automate, nous aurons une transition d'un M-net déplaçant un jeton d'une place à une autre. Les transitions portent les mêmes actions et les mêmes gardes. Il existe quand même quelques différences notables :

- nous pouvons représenter toutes les voies (avec les trains) par un seul M-net, les différents trains étant représentés par différents jetons ;
- le contrôleur n'utilise pas une variable entière mais stocke le nombre de trains dans ses places (ce qui a le même effet) ;
- nous avons en plus une horloge causale et les transitions portent des actions pour communiquer avec elle ;
- les contraintes de temps spécifiées dans les états avec les automates sont ici spécifiées au niveau de l'horloge ;
- les M-nets devant être ex-restreints, le système global démarre avec une action silencieuse synchronisant l'initialisation de tous les M-nets le composant.

Nous utilisons une horloge causale très similaire à celle présentée figure 4.2 (page 67) mais légèrement adaptée à notre problème. En particulier, puisque le système n'est pas destiné à s'arrêter, l'horloge n'a pas de transition pour terminer, et sa transition d'initialisation est synchronisée avec les transitions initialisant les autres M-nets composant le système. De plus, nous fixons le nombre de compteurs que l'horloge peut manipuler et nous les assignons de la façon suivante :

- le M-net représentant le contrôleur à l'identificateur 0, nous fixons donc $m_0 \stackrel{\text{df}}{=} c_M$;
- le M-net modélisant les barrières utilise l'identificateur 1 et nous posons $m_1 \stackrel{\text{df}}{=} g_M$;
- chaque train utilise deux identificateurs puisqu'il nécessite deux maximums (a_M et e_M) qui sont fixés dans l'horloge. Les trains étant numérotés de 1 à n, le train i utilise le compteur $2i$ pour son approche et le compteur $2i+1$ pour son départ, ce qui implique $m_{2i} \stackrel{\text{df}}{=} a_M$ et $m_{2i+1} \stackrel{\text{df}}{=} e_M$.

L'horloge causale que nous utilisons est donnée figure 5.3.

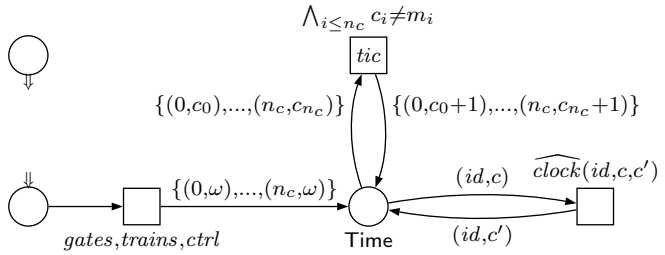


Figure 5.3 — L'horloge causale H_r utilisée pour le passage à niveau. Nous avons $n_c \stackrel{\text{df}}{=} 2n+1$ où n est le nombre de voies parallèles et le type de la place interne est $\{(id, i_{id}) \mid 1 \le id \le n_c, i_{id} \in \{0, 1, \ldots, m_{id}, \omega\}\}$.

Les autres éléments de la spécification sont donnés sur la figure 5.4. Nous pouvons y reconnaître les automates transformés comme nous l'avons décrit plus haut.

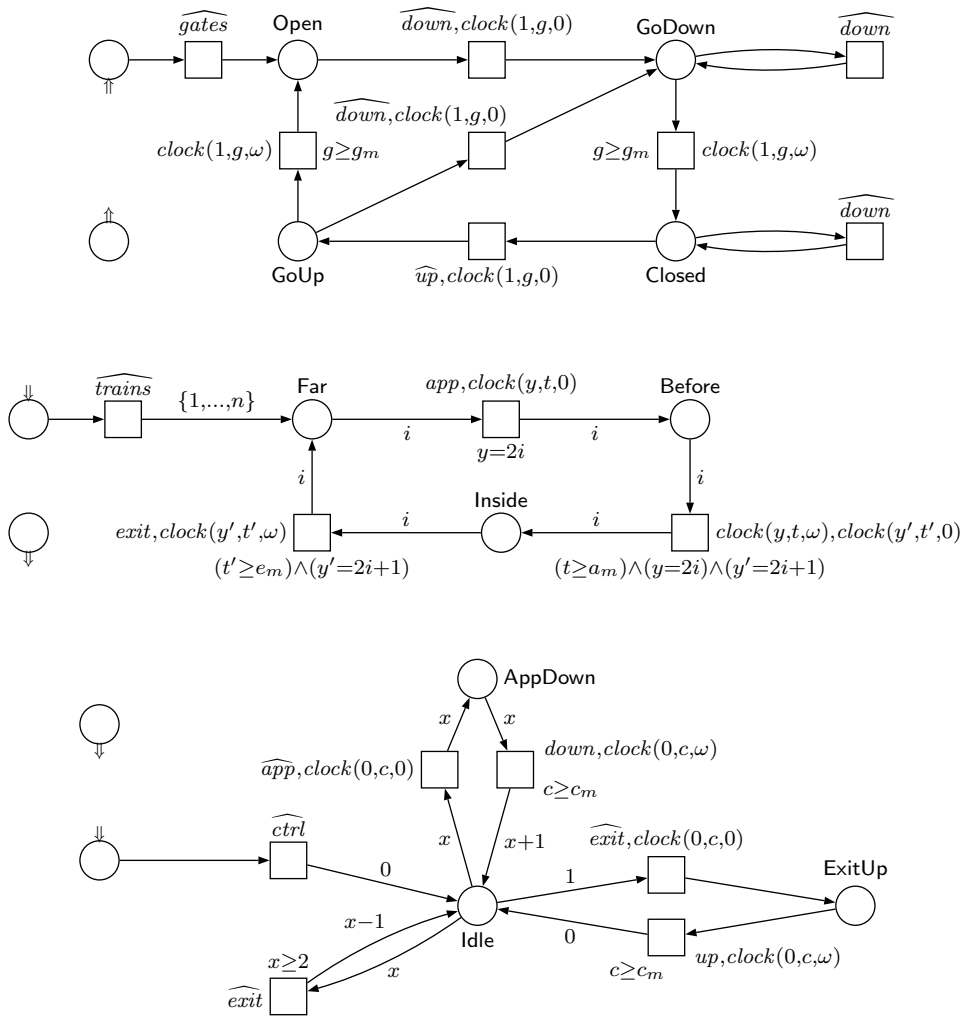


Figure 5.4 — De haut en bas, les M-nets N_g, N_t et N_c spécifiant respectivement les barrières, les voies avec leurs trains et le contrôleur. Les places Far, Before et Inside ont le type $\{1, \ldots, n\}$ et les places AppDown et Idle ont le type $\{0, \ldots, n\}$.

Nous obtenons le système complet en composant en parallèle tous ces réseaux et en synchronisant sur toutes les actions visibles :

$$N_r = (N_g \| N_t \| N_c \| H_r) \,\mathsf{sc}\, \{clock, down, up, app, exit, gates, trains, ctrl\} \quad .$$

Par rapport à la spécification à base d'automates temporisés, celle-ci peut sembler plus complexe ; en fait, une bonne part de la complexité vient de la représentation graphique

des réseaux de Petri, pour lesquels les transitions sont des nœuds alors qu'il ne s'agit que d'arcs dans les automates. Cependant, il est vrai qu'il faut ajouter les actions pour communiquer avec l'horloge et que les valeurs des paramètres sont calculées dans les gardes. La modélisation est donc un peu plus complexe avec les M-nets ; en revanche nous pouvons représenter la concurrence (et non seulement l'entrelacement) alors que c'est impossible avec les automates. Cet exemple n'exploite pas cette possibilité car nous avons cherché à calquer les M-nets sur les automates correspondants pour mettre en valeur la ressemblance des spécifications.

Nous vérifions la sécurité du système en testant qu'il n'existe pas de marquage M accessible à partir du marquage d'entrée de N_r tel que

$$(M(\mathsf{Inside}) \neq \varnothing) \wedge (M(\mathsf{Closed}) = \varnothing) \quad ,$$

ce qui s'exprime dans le formalisme de MARIA par :

```
reject !(place Inside equals empty)
       && (place Closed equals empty) && fatal;
```

L'absence de blocage peut aussi être directement exprimée par la clause :

```
deadlock fatal;
```

Ces deux clauses sont testées à mesure que le graphe de marquage est généré ; si l'une ou l'autre est violée, la génération est stoppée et l'erreur est signalée (c'est l'utilité du mot-clef `fatal`).

5.4 Résultats

Nous donnons maintenant les résultats obtenus pour les différents outils lors de la vérification de diverses versions de la spécification. Tous les tests ont été menés sur une station Sun Sparc à 440 Mhz, équipée d'un giga-octet de mémoire vive et d'un giga-octet supplémentaire de mémoire virtuelle. Nous avons travaillé depuis le répertoire `/tmp` de la machine qui est situé dans la mémoire du système d'exploitation (en utilisant le système TPMFS de Sun [Sny]), de cette façon tout le travail s'est effectué en mémoire, y compris les accès disque. En effet, MARIA stockant les graphes de marquages dans des fichiers, le faire travailler directement sur disque l'aurait lourdement pénalisé et aurait faussé la pertinence des résultats.

Lorsqu'une phase de pré-compilation a été nécessaire, le temps consommé est inclus dans les résultats présentés. Cela a été le cas pour Kronos, qui a besoin de calculer les produits synchronisés des automates avant de les vérifier, et pour MARIA, qui peut créer des bibliothèques en langage C, chargées par le programme à l'exécution, qui permettent d'évaluer les gardes des transitions de façon plus efficace.

Les temps indiqués sont ceux donnés par l'utilitaire Unix `time`, nous n'avons gardé que le temps « réel » consommé par les processus.

Nous avons vérifié les systèmes respectant la condition de sécurité et n'ayant pas de blocage pour un à six trains, en utilisant en particulier les constantes suivantes : $a_m = 4$,

$a_M = 5$, $c_m = 0$, $c_M = 1$, $e_m = 4$, $e_M = 6$, $g_m = 0$ et $g_M = 2$. Les temps constatés dans ce cas sont indiqués dans le tableau ci-dessous :

trains	1	2	3	4	5	6
MARIA	0,2s	0,2s	0,9s	12s	4m	1h12m
Kronos	0s	0,1s	1,6s	20s	5m	1h36m
UPPAAL	0,3s	0,5s	0,7s	27s	57m	–

Avec six trains, après environ 12h30m, UPPAAL a épuisé toute la mémoire vive et le système d'exploitation a commencé à utiliser la mémoire virtuelle sur disque de manière intensive. Le taux d'utilisation du processeur est tombé à moins de 1% et nous avons donc décidé de stopper le test car les résultats n'auraient pas été significatifs.

Les systèmes violant la condition de sécurité peuvent être générés avec les mêmes valeurs de constantes que celles présentées ci-dessus, sauf pour g_M qui a été augmenté à 3. De cette façon, les barrières ont la possibilité de se fermer trop lentement et un train peut les atteindre avant qu'elles ne soient complètement baissées. Les temps mesurés sont alors les suivants :

trains	1	2	3	4	5	6
MARIA	0,1s	0,2s	0,2s	0,2s	0,3s	0,4s
Kronos	0s	0,2s	1,7s	21,4s	6m57s	5h59m
UPPAAL	0s	0s	0s	0s	0s	0s

Remarquons que la dernière ligne du tableau est correcte : UPPAAL s'est révélé incroyablement rapide sur les systèmes comportant des erreurs (violation de la sécurité ou blocage).

Les systèmes avec blocage peuvent être produits à partir des systèmes sûrs en retirant dans la spécification de la barrière les transitions lui permettant de recevoir un signal *down* alors qu'elle est déjà en train de se baisser ou qu'elle est fermée. Remarquons que de cette façon, lorsqu'un seul train est utilisé, il ne peut pas y avoir de blocage ; mais dans tout les autres cas, un blocage est possible. Les temps mesurés sont alors les suivants :

trains	1	2	3	4	5	6
MARIA	0,1s	0,2s	0,2s	0,2s	0,3s	0,4s
Kronos	0s	0,2s	1,7s	21,4s	6m55s	6h02m
UPPAAL	0s	0s	0s	0s	0s	0,1s

5.4.1 Cohérence des résultats

Dans notre étude de cas, nous n'avons pas pris en considération le fait que le temps causal est un temps d'horloge (un nombre de tics) alors que les automates temporisés supposent un temps continu. Nous avons utilisé les mêmes constantes et les mêmes contraintes dans les différentes modélisations sans tenir compte de cette différence. Nous avons préféré utiliser pour chaque modèle la spécification la plus naturelle.

Outre les performances constatées, l'étude de cas présentée dans ce chapitre nous amène à croire qu'il existe une classe non triviale d'automates temporisés qui peuvent systématiquement être traduits en M-nets causalement temporisés. Nous pensons pouvoir identifier cette classe et définir une traduction systématique tout en garantissant une

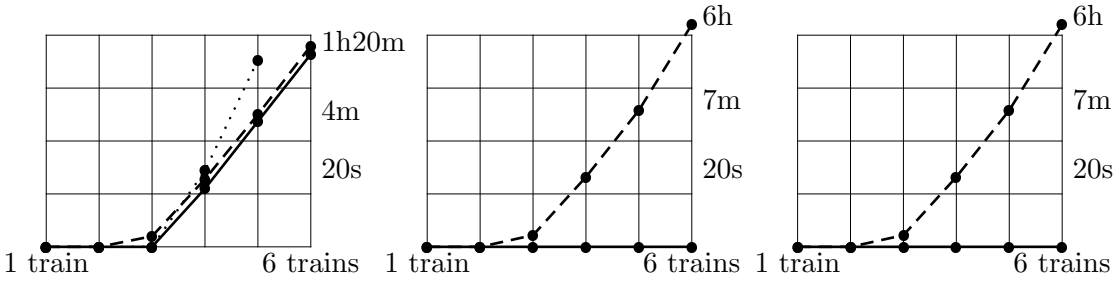


Figure 5.5 — Les temps consommés par la vérification avec MARIA (lignes continues), Kronos (tirets) et UPPAAL (pointillés), pour différents systèmes ; de gauche à droite : systèmes corrects, systèmes sans sécurité et systèmes avec blocage. Les échelles sont logarithmique. Dans les deux derniers cas, les courbes pour MARIA et UPPAAL sont confondues.

cohérence au niveau des exécutions : nous pouvons vraisemblablement établir une bisimilarité entre les exécutions des automates et celles des M-nets obtenus par traduction. Des travaux dans cette direction ne sont pas encore commencés mais devraient l'être à court terme.

À l'appui de cette conjecture, cette section expose les observations que nous avons pu mener au cours de l'étude de cas présentée plus haut. Nous avons généré et testé de nombreuses versions des spécifications, en utilisant différentes valeurs des constantes, avec ou sans blocage, et avec ou sans sécurité. Nous avons pu observer que les comportements des systèmes étaient toujours cohérents : des spécifications générées dans les mêmes conditions (valeurs des constantes et absence ou présence des transitions recevant *down*) étaient vérifiées avec la même conclusion par tous les outils. Cela conforte notre hypothèse sur la pertinence de la comparaison que nous avons présentée, même si elle n'a pas été faite sur des systèmes identiques.

Considérons par exemple le système sans sécurité, à trois trains, pour lequel nous avons présenté les temps d'exécution plus haut. Lorsqu'il rencontre un marquage incorrect, MARIA indique une séquence de transitions permettant de l'atteindre ; elle correspond à :

- Initialisation.
- Le jeton 1 dans la place Far, qui correspond au premier train, est déplacé dans la place Before, pendant ce temps, le jeton du contrôleur est déplacé de Idle à AppDown.
- Le tic d'horloge est franchi.
- Le jeton du contrôleur est déplacé de AppDown à Idle et celui des barrières de Down à GoDown.
- Le tic d'horloge est franchi trois fois.
- Le train 1 se déplace de Before à Inside.

Avec UPPAAL, nous obtenons une trace qui correspond à :

- L'automate $Train_1$ passe de l'état Far à l'état Before pendant que le contrôleur passe de Idle à AppDown.

- Attente d'une unité de temps.
- Le contrôleur va de AppDown à Idle et les barrières de Open à GoDown.
- Attente de trois unités de temps.
- $Train_1$ passe de Before à Inside.

Nous observons très clairement que ces deux exécutions sont très similaires (mise à part l'initialisation qui est spécifique à la modélisation M-nets). La trace donnée par Kronos est elle aussi similaire : un tic dans les M-nets correspond à exactement une unité de temps *continu* dans les automates (alors que la correspondance théorique serait qu'un tic correspond à un délai dans $d \in]1, 2[$ sur l'échelle de temps continue).

5.4.2 Explosion de l'espace des états

Les résultats présentés ci-dessus indiquent que les performances obtenues avec MARIA pour les M-nets causalement temporisés sont généralement meilleures que celles obtenues avec les autres outils pour les automates temporisés. Cet optimisme doit cependant être un peu tempéré. En effet, en s'appuyant sur MARIA, l'approche causale du temps souffre du problème bien connu d'*explosion de l'espace des états* : si nous augmentons les constantes utilisées pour borner les comptages de tics, le nombre de marquages accessibles croît très vite. Comme MARIA génère explicitement tous ces marquages, ses performances deviennent très mauvaises.

Avec les automates temporisés, ce qui est effectivement testé n'est pas les automates eux mêmes mais des *automates de régions* qui confondent les états équivalents des automates de départ. Les régions sont nécessaires dans cette approche car le temps étant supposé continu, un système qui utiliserait au moins un chronomètre aurait une infinité non dénombrable d'états distincts. Une telle notion de régions n'existe pas dans le cas du temps causal. Nous envisageons de mener des recherches dans cette direction afin de lever la limitation actuelle.

D'autre part, les techniques permettant d'alléger le problème d'explosion de l'espace de états pour les réseaux de Petri sont typiquement basées sur l'indépendance de certaines actions et reposent le plus souvent sur des sémantiques d'ordres partiels des réseaux. Par exemple, l'espace des états peut être représenté implicitement par un préfixe du réseau d'occurrences généralisé [McM95]. Ces techniques sont pour le moment limitées aux modèles basés sur les réseaux places/transitions, mais des recherches récentes dans le domaine montrent qu'il est possible de les étendre à des réseaux colorés pour produire des préfixes de haut niveau qui peuvent être analysés [Kou+02, FP00]. Il est même possible d'améliorer considérablement l'efficacité des vérifications en définissant une équivalence sur les marquages, ce qui permet de regrouper de nombreux états dans les préfixes générés. Cela revient à abstraire les données du réseau coloré pour ne les représenter que lorsqu'elles influent sur son exécution. Dans notre spécification, par exemple, la place Time n'apparaîtrait dans le préfixe que lorsque les valeurs qu'elle contient seraient discriminantes pour l'évolution du système. De même, la plupart des occurrences du tic d'horloge seraient simplement absentes du préfixe.

Ce type d'approches mènera certainement rapidement à une solution satisfaisante du problème d'explosion de l'espace des états. Dans ce cas, non seulement nous lèverions les limites actuelles, mais en plus, les performances constatées pourraient se voir encore améliorées puisque la vérification des préfixes est généralement beaucoup plus efficace que la génération des graphes de marquage. En fait, le gain en efficacité est proportionnel au degré de concurrence du réseau vérifié. Dans notre cas, il faudra réaliser un contrôleur permettant plus de concurrence pour améliorer grandement nos résultats. Remarquons quand même que les constantes que nous avons utilisées ne sont pas spécialement petites et l'utilisation de ces techniques réduirait considérablement la taille de la représentation de l'espace des états, même avec la spécification présentée ici.

6 Préemption

Ce chapitre est dédié à l'introduction de la préemption dans le modèle des M-nets. Cette introduction est motivée par la recherche d'un modèle complet pour le temps-réel et nous avons vu au chapitre 2 que la préemption est un concept central du temps-réel, au même titre que la représentation du temps.

Nous avons aussi vu que la préemption impérative nécessite une vue globale de la tâche à interrompre. Pour modéliser la suspension notamment, il faut donc se doter d'un mécanisme permettant de contrôler la franchissabilité d'un ensemble de transitions (celles qui composent la tâche à suspendre) de manière centralisée. Dans les réseaux de Petri, chaque transition n'a qu'une vue locale du système (ses places en entrée) et son franchissement ne dépend que de cet environnement restreint. La franchissabilité d'un ensemble de transitions est donc un critère réparti et non centralisé comme nous en aurions besoin. Ce modèle se révèle donc à priori peu adapté à l'introduction de la suspension.

Afin de surmonter cette difficulté, nous augmentons le modèle des M-nets avec des priorités entre les transitions. Ajouter des priorités aux réseaux de Petri peut augmenter leur expressivité et permettre de représenter les machines de Turing. C'est le cas en général lorsque les réseaux sont non bornés. Les places de liens n'étant pas bornées en général, l'ajout des priorités augmente strictement la puissance d'expression des M-nets. Cependant, comme nous l'avons vu à la section 3.4, il est possible de garantir que les places de liens restent bornées par l'ajout de places complémentaires. Nous pouvons alors montrer que les réseaux ainsi bornés sont des abréviations de réseaux places/transitions et que leur taille est compactée (avec un facteur exponentiel) grâce à l'utilisation des priorités.

L'ajout des priorités nous permet de définir deux nouveaux opérateurs autorisant respectivement la suspension/reprise et l'avortement d'un M-net à priorités. Nous respectons ainsi l'aspect composable de notre modèle de départ et nous obtenons un nouveau modèle, baptisé *algèbre des M-nets préemptibles* (aussi appelés *PM-nets*). L'importance de l'indépendance de la préemption par rapport aux autres aspects, comme le temps, le contrôle de flot et les communications est détaillée dans [Ber93]. En l'intégrant sous forme d'opérateurs pouvant être librement utilisés (c'est-à-dire comme tous les autres

opérateurs de l'algèbre), nous garantissons sa complète orthogonalité au reste du modèle.

Afin d'aboutir à la définition de l'algèbre des PM-nets, nous procédons en plusieurs étapes correspondant aux sections de ce chapitre.

1. Nous commençons par ajouter aux M-nets des priorités entre leurs transitions. Il en résulte des *M-nets à priorités*. Cette classe de réseaux de Petri peut être vue comme une version de haut niveau des *systèmes à priorités* définis et étudiés dans [BK92].

2. Les opérateurs de l'algèbre des M-nets sont ensuite étendus pour tenir compte des priorités. Les M-nets à priorités sont alors dotés de la même structure algébrique que les M-nets présentés au chapitre 3.

3. Nous définissons deux nouveaux opérateurs, π_s et π_a, autorisant respectivement la suspension/reprise et l'avortement de M-nets à priorités.

4. Nous définissons enfin l'algèbre des PM-nets grâce à des contraintes syntaxiques sur les M-nets à priorités qui permettent d'assurer de bonnes propriétés au modèle ainsi obtenu (en particulier, nous montrons qu'ils abrègent les réseaux P/T dans les cas finis).

Remarquons que l'approche consistant à définir un modèle général et puissant sur lequel on impose des restrictions syntaxiques a déjà été utilisée pour les M-nets (voir les restrictions syntaxiques imposées à la section 3.3 page 48).

6.1 M-nets à priorités

Soit $N = (S, T, \iota)$ un M-net. Une *relation de priorités* sur les transitions de N est une relation binaire $\rho \subseteq T \times T$. Pour $(t_1, t_2) \in \rho$, nous notons $t_1 \prec_\rho t_2$ (ou simplement $t_1 \prec t_2$ s'il n'y a pas d'ambiguïté sur ρ) ; cela signifie intuitivement que le franchissement de t_2 est toujours prioritaire sur celui de t_1 lorsque les deux transitions peuvent être franchies. D'autre part, si t_1 et t_2 sont en conflit sur une place, la priorité résout le conflit en faveur de t_2 ; si les deux transitions sont indépendantes, et donc potentiellement franchissables dans un même step, seule t_2 peut s'exécuter car t_1 est inhibée.

Le *graphe* d'une relation de priorités ρ sur T a pour nœuds les transitions de T et, pour chaque paire $(t, t') \in \rho$, il existe un arc $t \leftarrow t'$ dans le graphe. La figure 6.1 donne un exemple de tel graphe.

Une relation de priorités ρ est *bien formée* si elle ne spécifie rien d'incohérent avec l'intuition donnée ci-dessus. Par exemple, s'il n'existe pas de transitions t_1 et t_2 telles que $t_1 \prec t_2$ et $t_2 \prec t_1$. Plus précisément, ρ est bien formé si le graphe de ρ ne contient pas de cycle (en particulier, il faut avoir $\rho \cap \{(t, t) \mid t \in T\} = \varnothing$ qui interdit les cycles ne contenant qu'un nœud). Cela revient à dire que nous pouvons compléter ρ pour former un ordre partiel, ce qui correspond bien à la perception intuitive d'une relation de priorités. La relation de la figure 6.1 est bien formée.

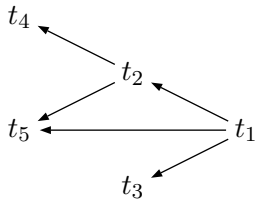


Figure 6.1 — Le graphe de la relation de priorités $\{(t_2, t_1), (t_3, t_1), (t_5, t_1), (t_4, t_2), (t_5, t_2)\}$.

Un *M-net à priorités*, ou *ρ-net*, est une paire $P = (N, \rho)$ où N est un M-net et ρ est une relation de priorités sur les transitions de N. Un ρ-net $P = (N, \rho)$ est *bien formé* si N et ρ le sont. Nous appelons N la partie réseau de P et ρ sa relation de priorités.

Remarquons que nous n'imposons pas à un ρ-net d'être bien formé. Nous verrons dans la suite que c'est une propriété qui sera obtenue, sur une sous-classe des ρ-nets, par des restrictions syntaxiques.

Un ρ-net est *statique*, respectivement *dynamique*, si sa partie réseau l'est. Plus généralement, le vocabulaire associé aux propriétés des M-nets et de leurs marquages est étendu aux ρ-nets en considérant leurs parties réseaux.

Franchissement et sémantique concurrente. La règle de franchissement des ρ-nets doit prendre en compte les priorités entre transitions. Soit $P = (N, \rho)$ un ρ-net tel que $N = (S, T, \iota)$ et soit M un marquage de N (par extension, nous dirons aussi que c'est un marquage de P). Une transition $t \in T$, franchissable sous le marquage M, est *ρ-franchissable*, ce que nous notons $M[t\rangle_\rho$, s'il n'existe pas de transition $t' \in T$ telle que $M[t'\rangle$ et $t \prec t'$. Nous voyons ici que l'ajout de priorités peut rendre non franchissable une transition qui aurait pu l'être avec la règle de franchissement des M-nets, mais que le contraire n'est pas vrai. Nous avons donc toujours $M[t\rangle_\rho \Rightarrow M[t\rangle$ mais pas la réciproque.

Comme la règle de franchissement, la notion de steps doit être adaptée aux priorités. Sans cela, la sémantique steps des M-nets à priorités contiendrait des incohérences. Considérons par exemple le ρ-net $P = (N, \rho)$ présenté figure 6.2 (cet exemple est repris de [BK92]).

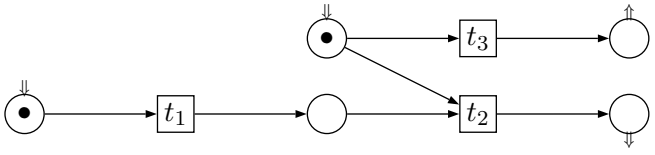


Figure 6.2 — Un ρ-net pour lequel $\rho = \{(t_3, t_2)\}$.

En considérant uniquement sa partie réseau N, nous obtenons :

$$\mathsf{steps}(N) = \{\varnothing, \{t_1\}, \{t_3\}, \{t_1, t_3\}, \{t_1\}\{t_3\}, \{t_3\}\{t_1\}, \{t_1\}\{t_2\}\} \quad ,$$

où $\varnothing$ représente la séquence vide. Cette sémantique contient la séquence $\{t_1\}\{t_3\}$ qui viole la relation de priorités ; en effet, si t_1 est franchie, t_2 et t_3 deviennent toutes deux franchissables et, puisque $t_3 \prec t_2$, t_3 n'est pas ρ-franchissable. Il faut donc supprimer cette séquence de la sémantique. Ce n'est pourtant pas suffisant car la sémantique contient encore le step $\{t_1, t_3\}$ dont $\{t_1\}\{t_3\}$ est une linéarisation. Pour conserver la cohérence de la sémantique par rapport à la linéarisation, il faut aussi supprimer $\{t_1, t_3\}$. La *sémantique steps cohérente* d'un ρ-net $P = (N, \rho)$, notée $\mathsf{steps}(P)$, est donc le plus grand sous-ensemble de $\mathsf{steps}(N)$ tel que chaque séquence de $\mathsf{steps}(P)$ et chacune de ses linéarisations respecte ρ. Pour l'exemple ci-dessus, nous obtenons :

$$\mathsf{steps}(P) = \{\varnothing, \{t_1\}, \{t_3\}, \{t_3\}\{t_1\}, \{t_1\}\{t_2\}\} \quad .$$

Dans [BK92], les auteurs considèrent cette sémantique comme l'une des « plus concurrentes » qu'on peut obtenir pour des réseaux de Petri à priorités.

6.2 Extension des opérateurs

L'extension des opérateurs M-nets aux ρ-nets est immédiate pour la plupart d'entre eux. Cependant, dans le cas de la synchronisation et de la substitution de transitions, plusieurs définitions peuvent être envisagées. Notre but n'est pas de définir une algèbre générale de M-nets à priorités ; nous voulons simplement établir une base sur laquelle définir les M-nets préemptibles (PM-nets), en particulier les opérateurs de la section 6.3. Nos définitions ne sont certes pas les plus générales (si tant est qu'il en existe) mais elles nous permettent d'arriver à nos fins tout en assurant de bonnes propriétés au modèle des PM-nets.

Substitution de transitions. Soient P, P_1, …, P_k des ρ-nets tels que $P = (N, \rho)$ avec $N = (S, T, \iota)$ et $P_i = (N_i, \rho_i)$ avec $N_i = (S_i, T_i, \iota_i)$ pour $1 \leq i \leq k$, et soient $\mathcal{X}_1$, …, $\mathcal{X}_k$ des symboles d'actions hiérarchiques. Nous définissons $P[\mathcal{X}_i \leftarrow P_i \mid 1 \leq i \leq k] \stackrel{\text{df}}{=} (N[\mathcal{X}_i \leftarrow P_i \mid 1 \leq i \leq k], \rho')$, avec :

$$\begin{aligned}\rho' \stackrel{\text{df}}{=} & \{(t, t') \in \rho \mid \iota(t) \neq \mathcal{X}_i \neq \iota(t') \text{ pour } 1 \leq i \leq k\} \\ & \uplus \{(t_{\mathcal{X}_i}.t, t_{\mathcal{X}_i}.t') \mid t \prec_{\rho_i} t',\ t_{\mathcal{X}_i} \in T, \iota(t_{\mathcal{X}_i}) = \mathcal{X}_i \text{ et } 1 \leq i \leq k\} \\ & \uplus \{(t_{\mathcal{X}_i}.t, t') \mid t_{\mathcal{X}_i} \prec_{\rho} t',\ \iota(t_{\mathcal{X}_i}) = \mathcal{X}_i \neq \iota(t'),\ t \in T_i \text{ et } 1 \leq i \leq k\} \quad .\end{aligned}$$

La première ligne de cette union permet de reporter dans ρ' les priorités qui existent dans ρ entre les transitions non hiérarchiques. La deuxième ligne permet de reporter les priorités de chaque réseau P_i. La dernière ligne fait en sorte de propager les priorités hiérarchiques aux réseaux P_i : si dans P, une transition t a la priorité sur t' et que t' est remplacée par un réseau P_i, alors t obtient la priorité sur toutes les transitions issues de P_i. (Nous n'autorisons pas la propagation dans l'autre sens car cela permettrait d'introduire des cycles dans une relation de priorités, ce qui n'est pas souhaitable.)

Nous pouvons symboliser ce que nous obtenons au niveau du graphe de ρ' comme sur la figure 6.3 où nous prenons $k = 2$ pour plus de clarté. Chaque ovale en pointillés

représente le graphe d'une relation ; ceux de ρ_1 et ρ_2 ont été dupliqués autant de fois que de transitions substituées dans P et toutes ces copies sont disjointes. Les arcs dessinés indiquent que certaines transitions du graphe de ρ peuvent avoir la priorité sur toutes les transitions d'une ou plusieurs copies de ρ_1 ou de ρ_2. Cette représentation montre clairement que notre définition de la substitution de transition ne permet pas d'introduire de cycle dans une relation de priorités (bien qu'elle puisse conserver des cycles déjà présents).

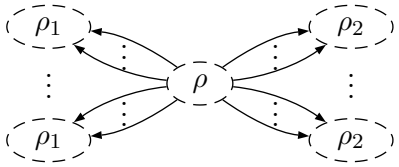


Figure 6.3 — Une vue simplifiée du graphe de la relation de priorités de $P[\mathcal{X}_1 \leftarrow P_1, \mathcal{X}_2 \leftarrow P_2]$.

Synchronisation. La définition de la synchronisation impose que les priorités d'une transition crée par synchronisation sont héritées des transitions dont elle est issue. La figure 6.4 donne une illustration de cette idée.

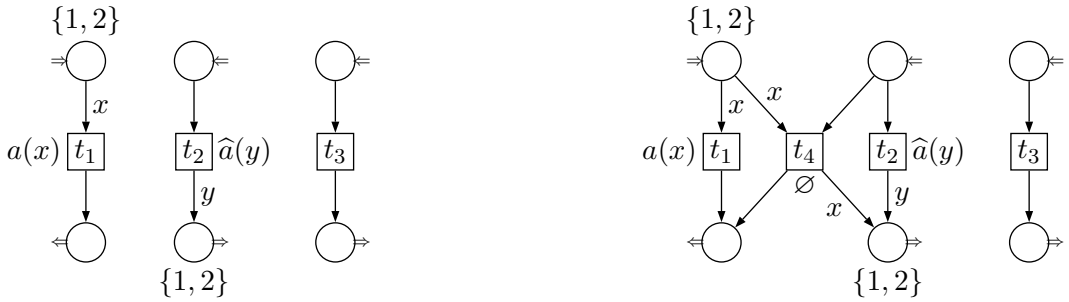


Figure 6.4 — À gauche, un ρ-net dont la relation de priorités est $\{(t_3, t_2)\}$. À droite, le même réseau après synchronisation sur a : la relation de priorités est devenue $\{(t_3, t_2), (t_3, t_4)\}$.

Formellement, soit $P = (N, \rho)$ un ρ-net tel que $N = (S, T, \iota)$ et soit $a \in \mathbb{A}$ un symbole d'action. Alors, $P \,\mathsf{sy}\, a \stackrel{\text{df}}{=} (N', \rho')$ où $N' \stackrel{\text{df}}{=} N \,\mathsf{sy}\, a = (S', T', \iota')$. La relation de priorités ρ' est définie comme le plus petit ensemble contenant ρ tel que, pour chaque transition $t \in T'$ résultant d'une synchronisation de deux transitions t_1 et t_2, et, pour chaque transition $t' \in T'$,

- si $t_1 \prec_{\rho'} t'$ ou si $t_2 \prec_{\rho'} t'$ alors $t \prec_{\rho'} t'$;
- si $t' \prec_{\rho'} t_1$ ou si $t' \prec_{\rho'} t_2$ alors $t' \prec_{\rho'} t$.

Les différents cas possibles sont schématisés sur la figure 6.5 où nous avons restreint les graphes des relations de priorités aux seuls nœuds t_1, t_2, t et t' utilisés ci-dessus

(rappelons que t_1 et t_2 se synchronisent pour donner t). Dans cette figure nous indiquons par une flèche à double pointe les arcs qui pourraient être des chemins si le graphe contenait plus de nœuds (dans ce cas, il faut considérer que t' indiqué sur la figure n'est pas celui de la définition). Par exemple, dans le dernier cas (en bas à droite de la figure), s'il existe avant la synchronisation un chemin $t_1 \to \cdots \to t_2$ passant par le nœud t' indiqué, la synchronisation crée un chemin $t \to \cdots \to t$ (passant toujours par t') qui forme un cycle.

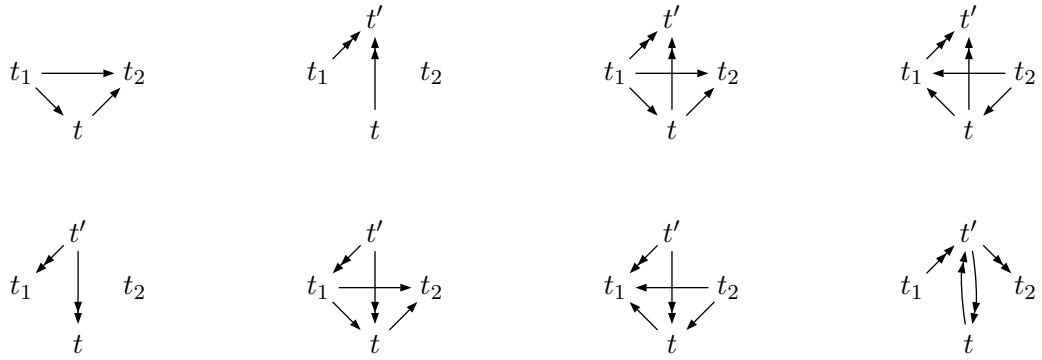


Figure 6.5 — Les différentes configurations après synchronisation (les sous-graphes avant synchronisation s'obtiennent en supprimant le nœud t et ses arcs adjacents). Dans le premier cas, nous avons $t' \in \{t_1, t_2\}$. On obtient les autres cas en échangeant t_1 et t_2.

De tous les cas présentés sur la figure 6.5, seul le dernier peut produire un cycle. D'une manière générale, ce cas survient lorsqu'il existe une chaîne de priorités $t_1 \prec \ldots \prec t_k$ (avec $k \geq 3$) et que t_1 et t_k peuvent se synchroniser. Nous verrons par la suite qu'une telle situation n'arrive jamais avec les PM-nets.

Autres opérateurs. La définition des opérateurs de contrôle de flot faisant appel à des réseaux opérateurs (voir figure 3.8 page 58), nous devons définir des ρ-nets opérateurs. Pour cela, nous ajoutons simplement une relation de priorités vide aux M-nets opérateurs. Nous pouvons alors définir les opérateurs de contrôle de flot comme pour les M-nets.

Pour la restriction, étant donnés $P = (N, \rho)$ et un symbole d'action $a \in \mathbb{A}$, nous définissons $P \,\mathsf{rs}\, a \stackrel{\text{df}}{=} (N', \rho')$ avec $N' = (S', T', \iota') \stackrel{\text{df}}{=} N \,\mathsf{rs}\, a$ et $\rho' \stackrel{\text{df}}{=} \rho \cap (T' \times T')$.

Pour les autres opérations, nous prenons un ρ-net $P = (N, \rho)$, un symbole d'action $a \in \mathbb{A}$, un symbole de lien $b \in \mathbb{B}$ et une fonction de renommage f et alors : $P \,\mathsf{sc}\, a \stackrel{\text{df}}{=} (P \,\mathsf{sy}\, a) \,\mathsf{rs}\, a$, $P \,\mathsf{tie}\, b \stackrel{\text{df}}{=} (N \,\mathsf{tie}\, b, \rho)$ et $P[f] \stackrel{\text{df}}{=} (N[f], \rho)$.

6.3 Introduction de la préemption

Cette section définit deux nouveaux opérateurs, π_s et π_a. Le premier autorise la suspension/reprise d'un ρ-net arbitraire, le second autorise son avortement. Ces deux opérateurs sont très simples à mettre en œuvre et peuvent être utilisés directement. Nous pouvons

aussi les considérer comme des primitives de bas niveau au dessus desquelles des opérateurs plus complexes peuvent être construits. Nous donnons une illustration de ce point de vue en définissant des opérateurs permettant d'initier la préemption plus librement que ne le permettent π_s et π_a.

6.3.1 Suspension et reprise

Grâce aux priorités, il est assez simple de suspendre un ρ-net P : intuitivement, nous embarquons P dans un environnement qui contient une transition t_s ayant la priorité sur toutes les transitions de P. En rendant t_s franchissable, les transitions de P sont inhibées. Pour que le franchissement de t_s n'interrompe pas la suspension, il suffit de reproduire le jeton consommé.

Considérons le ρ-net P_s présenté sur la figure 6.6 et un ρ-net P arbitraire. Alors :

$$\pi_s(P) \stackrel{\text{df}}{=} P_s[\mathcal{X}_s \leftarrow (P \,\mathsf{sc}\, \mathit{resume})]\,\mathsf{sc}\, \mathit{sleep} \quad .$$

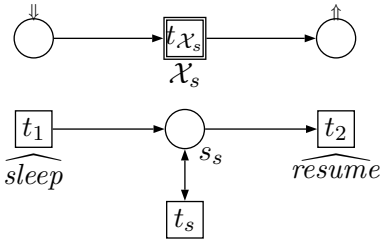


Figure 6.6 — Le ρ-net P_s dont la relation de priorités est $\rho_s \stackrel{\text{df}}{=} \{(t_{\mathcal{X}_s}, t_s)\}$.

La construction de $\pi_s(P)$ peut donc se décomposer de la façon suivante.

1. Nous appliquons à P un scoping sur l'action *resume*. En effet, cette action permet la reprise d'un réseau suspendu (voir le point 3 ci-dessous). S'il existe dans P un sous-réseau $\pi_s(P')$, nous permettons ainsi sa reprise après suspension depuis une autre partie de P franchissant une transition portant l'action *resume*. Le scoping synchronise cette transition avec t_2 issue de l'opération $\pi_s(P')$, rendant la reprise effective.

2. Nous substituons $t_{\mathcal{X}_s}$ par $P \,\mathsf{sc}\, \mathit{resume}$ dans le réseau P_s. Grâce à la priorité $t_{\mathcal{X}_s} \prec t_s$, toutes les transitions de $P \,\mathsf{sc}\, \mathit{resume}$ ont maintenant une priorité plus faible que t_s.

3. Un scoping sur *sleep* permet au réseau $\pi_s(P)$ de se suspendre. Si P a une transition portant l'action *sleep*, elle se synchronise avec t_1 dans P_s et le franchissement du résultat de cette synchronisation place un jeton dans s_s. La transition t_s est alors franchissable ce qui interdit l'exécution de toutes les transitions issues de P. Le franchissement de t_2 par la suite permet la reprise après suspension.

Remarquons que le réseau P_s n'est pas T-restreint. L'une des propriétés des PM-nets est de permettre à la T-restriction d'être restaurée simplement (ce sera l'objet du théorème 7(2) donné page 105).

Suspension externe. Avec cette définition de π_s, la suspension ne peut être initiée que depuis P, par le franchissement d'une transition synchronisée avec t_1 de P_s par l'opération $\mathsf{sc}\, sleep$ lors de l'application de π_s. Nous pouvons définir au dessus de π_s un nouvel opérateur, π'_s, permettant à la suspension de venir de l'extérieur :

$$\pi'_s(P) \stackrel{\text{df}}{=} \pi_s\Big(\Big((P \,\mathsf{sc}\, suspend\ ;\ P_t) \,\Big\|\, P_c\Big) \,\mathsf{sc}\, term\Big) \quad ,$$

où P_t et P_c sont présentés figure 6.7. Le scoping initial sur *suspend* a le même rôle que le scoping sur *resume* dans la définition de π_s.

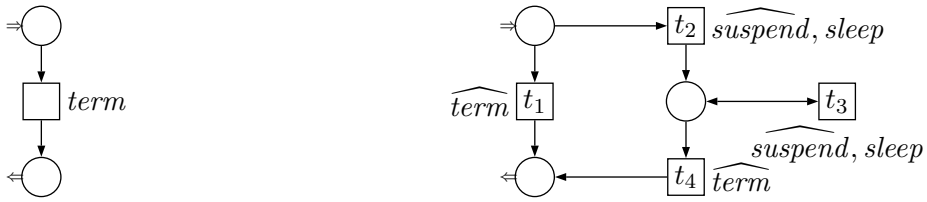


Figure 6.7 — À gauche le ρ-net P_t dont la relation de priorités est vide. À droite, P_c qui a aussi une relation de priorités vide.

Le rôle de P_c est de « convertir » chaque action *suspend* venant de l'extérieur de $\pi'_s(P)$ en une action *sleep* qui pourra être prise en compte par π_s. Le rôle de P_t est de permettre la terminaison de P_c lorsque P termine.

La figure 6.8 schématise le fonctionnement de π'_s en mettant en valeur les synchronisations possibles.

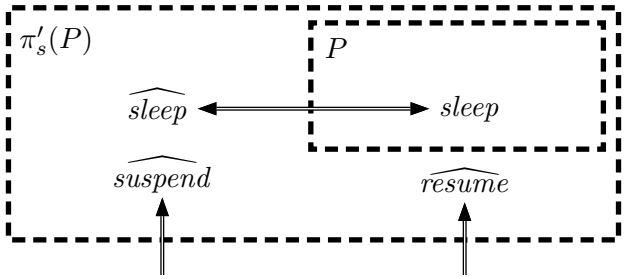


Figure 6.8 — Une vue schématique de $\pi_s(P)$. Les arcs doublés représentent les synchronisations (effectives ou potentielles) ; les boites en pointillés symbolisent les différents réseaux (dont le nom est donné dans le coin supérieur gauche de chaque boite).

Exemple. Considérons un ρ-net *Print* modélisant le processus d'impression d'une imprimante. Nous voulons permettre la suspension de l'impression lorsque le bac à papier de l'imprimante est ouvert. Pour cela, nous utilisons l'opérateur π'_s et nous définissons un nouveau processus :

$$Print_s \stackrel{\text{df}}{=} \Big((\pi'_s(Print); P_t) \,\Big\|\, Tray\Big) \,\mathsf{sc}\, \{suspend, resume, term\} \quad ,$$

où *Tray*, présenté figure 6.9, modélise le comportement du bac à papier (P_t étant donné figure 6.7).

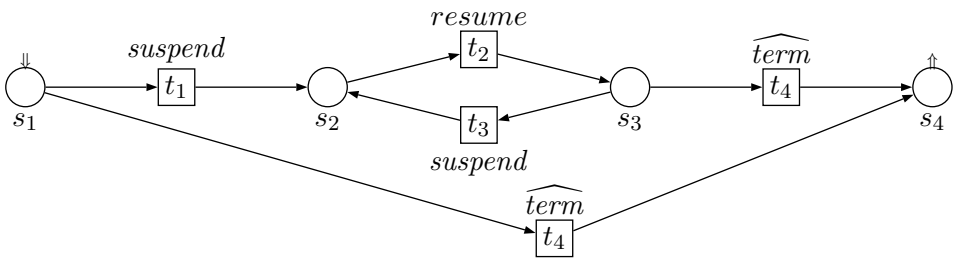


Figure 6.9 — Le ρ-net *Tray* dont la relation de priorités est vide. Les places s_1 et s_3 correspondent à l'état « bac fermé » et la places s_2 à l'état « bac ouvert ». Les transitions entre ces places modélisent l'ouverture et la fermeture du bac.

6.3.2 Avortement

Par rapport à la suspension, l'avortement présente une difficulté supplémentaire : il faut empêcher tout redémarrage du réseau avorté. Le suspendre et interdire sa reprise n'est pas une solution satisfaisante. En effet, la suspension peut être considérée comme une pause dans l'exécution d'un réseau. Nous pouvons d'ailleurs noter que le modèle des réseaux de Petri ne force pas le franchissement des transitions et qu'elles peuvent toujours attendre avant d'être franchies. De ce point de vue, la suspension ne sert qu'à forcer l'attente. L'avortement cependant n'est pas une attente mais une terminaison. Nous en déduisons immédiatement que l'avortement d'un ρ-net doit produire son marquage de sortie qui correspond effectivement à sa terminaison.

Pour réaliser cela, nous procédons en trois étapes. Tout d'abord, le réseau est suspendu (puisque nous avons affaire à un avortement impératif) ; ensuite, nous supprimons tous les jetons qu'il contient ; enfin, nous produisons son marquage de sortie. La suppression des jetons est réalisée par une boucle qui s'arrête quand il n'y a plus de jeton à consommer (les priorités nous permettent de détecter cette condition assez simplement).

Nous pourrions réaliser ces étapes en une seule de la façon suivante. Étant donné un ρ-net P, pour chaque marquage M de P accessible à partir d'un marquage initial, nous devrions ajouter à P une transition t_M ayant la priorité sur toutes les transitions de P ; le franchissement de cette transition devrait consommer M et produire le marquage de sortie de P. Il faudrait ensuite assurer que, lorsque plusieurs de ces nouvelles transitions seraient franchissables, celle qui consommerait le plus grand marquage (et qui viderait alors P) serait toujours choisie ; nous ajouterions donc une priorité $t_M \prec t_{M'}$ dès que M est un sous-marquage de M'.

Cette façon de faire n'est pas seulement inélégante, elle est aussi irréaliste puisqu'elle nécessite d'ajouter une infinité de transitions. En effet, les places de liens n'étant pas bornées, un ρ-net possède une infinité de marquages accessibles (comme nous l'avons vu sur la figure 3.10, page 60) et il faut pouvoir tous les consommer.

Notre solution en trois étapes permet donc de contourner cette difficulté : puisque la suppression des jetons ne peut être faite par un seul franchissement, la suspension est nécessaire pour empêcher le réseau en cours d'avortement d'évoluer. Ainsi modélisé, l'avortement n'est pas une action *élémentaire*, mais c'est une action *atomique* puisque le réseau avorté est suspendu pendant qu'elle a lieu et qu'il ne peut donc pas interagir avec le reste du système. De plus, l'avortement ne peut pas être interrompu autrement que par une autre préemption de plus haut niveau. S'il s'agit d'une suspension, l'avortement continuera à la reprise, s'il s'agit d'un autre avortement, le système qui initialement avortait le sera donc de toute façon.

Nous voyons ici que l'avortement est réalisé sans nouvelle extension du modèle (c'est-à-dire, sans ajouter autre chose que les priorités déjà nécessaires à la suspension). Cette remarque coïncide avec l'un des résultats principaux de [Pin+98] qui montre que la suspension est une opération primitive alors que l'avortement ne l'est pas (autrement dit, nous pouvons obtenir le second en nous basant sur la première, mais pas le contraire).

Suppression des jetons. Pour permettre la suppression des jetons, nous définissons un opérateur auxiliaire, clr, qui ajoute pour chaque place d'un réseau une transition pouvant y supprimer des jetons. Ces transitions seront par la suite synchronisées avec une boucle qui assurera la suppression de tous les jetons dans toutes les places. L'opérateur clr fait perdre l'ex-orientation qui sera donc rétablie systématiquement par l'opérateur d'avortement.

Soit $P = (N, \rho)$ un ρ-net tel que $N = (S, T, \iota)$. Nous définissons $\mathsf{clr}(P) \stackrel{\text{df}}{=} (N', \rho')$, avec $N' = (S', T', \iota')$, de la façon suivante :

- $S' \stackrel{\text{df}}{=} S$;
- $T' \stackrel{\text{df}}{=} T \uplus \{t_a^s \mid s \in S\}$;
- pour toute place $s \in S'$ et toute transition $t \in T'$:
  - $\iota'(s) \stackrel{\text{df}}{=} \iota(s)$,
  - $\iota'(t) \stackrel{\text{df}}{=} \begin{cases} \iota(t) & \text{si } t \in T, \\ \{clear\}.\{\top\} & \text{sinon,} \end{cases}$
  - $\iota'(s,t) \stackrel{\text{df}}{=} \begin{cases} \iota(s,t) & \text{si } t \in T, \\ \{y\} \subset Var & \text{si } t = t_a^s, \\ \varnothing & \text{sinon,} \end{cases}$
  - $\iota'(t,s) \stackrel{\text{df}}{=} \begin{cases} \iota(t,s) & \text{si } t \in T, \\ \varnothing & \text{sinon ;} \end{cases}$
- $\rho' \stackrel{\text{df}}{=} \rho \uplus \{(t, t_a^s) \mid s \in S \text{ et } t \in s^\bullet \cap T\}$.

L'opérateur d'avortement. Nous sommes maintenant en mesure de définir l'opérateur π_a permettant de rendre avortable un ρ-net arbitraire P. Le principe est presque le même que pour la suspension et nous utilisons le ρ-net P_a, donné sur la figure 6.10, qui ressemble beaucoup à P_s. La principale différence ici est que nous exploitons la boucle

sur la transition t_a pour supprimer un à un les jetons de P. Formellement :

$$\pi_a(P) \stackrel{\text{df}}{=} P_a[\mathcal{X}_a \leftarrow \mathsf{clr}(P); P_\varnothing]\,\mathsf{sc}\{\mathit{clear}, \mathit{abort}\} \quad ,$$

où $P_\varnothing$ est aussi donné sur la figure 6.10.

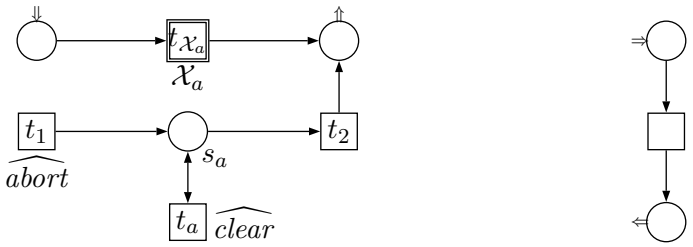


Figure 6.10 — À gauche, le ρ-net P_a, sa relation de priorités étant $\rho_a \stackrel{\text{df}}{=} \{(t_2, t_a)\}$. À droite, le réseau $P_\varnothing$ dont la relation de priorités est $\varnothing$.

Grâce au scoping sur *abort*, le franchissement dans P d'une transition portant cette action place un jeton dans s_a. Grâce au scoping sur *clear* ce jeton rend franchissables toutes les transitions t_a^s ajoutées par clr, pour chaque place s non vide. La priorité $t_2 \prec_{\rho_a} t_a$ garantit qu'elles seront toutes franchies avant que t_2 ne puisse produire le marquage de sortie.

La séquence avec le réseau $P_\varnothing$ a pour but d'assurer que $\pi_a(P)$ reste un réseau exorienté. En effet, l'application $\mathsf{clr}(P)$ ajoute des transitions t_a^s pour puiser les jetons dans toutes les places de P, y compris ses places de sortie. Le résultat n'est donc pas exorienté. En ajoutant $P_\varnothing$ en séquence, nous rétablissons cette propriété. Notons qu'il est nécessaire de vider les places de sorties de P car il est possible que certaines contiennent des jetons alors que d'autres pas encore. Tout vider nous assure que nous produirons un marquage de sortie correct.

Remarquons que, comme dans le cas de la suspension, le réseau P_a n'est pas T-restreint. Nous verrons dans la preuve du théorème 7(2) que cela n'a pas de conséquence sur la T-restriction d'un réseau $\pi_a(P)$.

Dans la définition de π_a, nous n'avons pas fait de différence entre une terminaison provoquée par un avortement est une terminaison « normale » (lorsque P n'est pas avorté). Nous verrons à la section 7.4.1 que, puisque P initie son propre avortement, il peut à ce moment exécuter une action lui permettant de prévenir son environnement. (Dans ce cas, il s'agira de déclencher le traitant d'une exception alors que le bloc de programme qui la lève est avorté.)

Avortement externe. Comme pour la suspension, nous pouvons définir au-dessus de π_a un nouvel opérateur π'_a autorisant un réseau à être avorté de l'extérieur. Étant donné un ρ-net P, nous définissons :

$$\pi'_a(P) \stackrel{\text{df}}{=} \pi_a\Big(\Big((P\ ;\ P_t) \Big\|\ P_k\Big)\,\mathsf{sc}\ \mathit{term}\Big) \quad ,$$

où P_t est donné figure 6.7 et P_k, donné figure 6.11, est chargé de « convertir » une action *kill* externe en action *abort*. Comme un seul avortement peut avoir lieu, il n'est pas utile ici d'utiliser une boucle.

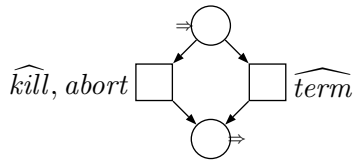


Figure 6.11 — Le ρ-net P_k, sa relation de priorités est vide.

La figure 6.12 schématise le fonctionnement de π'_a comme nous l'avons fait pour π'_s sur la figure 6.8. Cette fois, il n'y a pas d'action comme *resume* qui soit visible de l'extérieur de $\pi'_a(P)$ puisque l'avortement entraîne la terminaison du réseau.

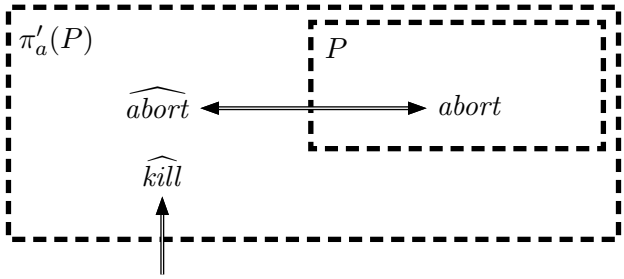


Figure 6.12 — Une vue schématique de $\pi_a(P)$.

Exemple. Considérons à nouveau le processus d'impression $Print_s$ qui nous a servi à illustrer la suspension. Nous souhaitons ajouter la modélisation d'un bouton d'annulation de l'impression. En utilisant l'opérateur π'_a, nous obtenons facilement :

$$Print_a \stackrel{\text{df}}{=} \Big((\pi'_a(P_1)\ ;\ P_t) \ \Big\|\ Cancel \Big)\ \mathsf{sc}\,\{kill, term\} \quad ,$$

où P_t est donné figure 6.7 et *Cancel*, donné figure 6.13, modélise le bouton d'annulation.

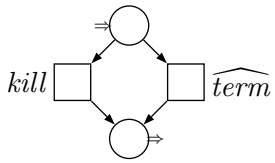


Figure 6.13 — Le ρ-net *Cancel* (sa relation de priorités est vide). La transition de gauche correspond à l'utilisation du bouton alors que celle de droite est franchie quand l'impression se termine normalement.

6.4 L'algèbre des M-nets préemptibles (PM-nets)

L'algèbre des PM-nets est définie par des restrictions syntaxiques sur la classe des M-nets à priorités. Un M-net à priorités $P = (N, \rho)$ est un *M-net préemptible* (ou *PM-net*) si l'une des conditions suivantes est vérifiée :

- N est un M-net bien formé statique ou dynamique et $\rho = \varnothing$.
- P est un ρ-net statique ou dynamique défini comme $\pi_s(P_0)$, $\pi_a(P_0)$, $P_0 \,\mathsf{sy}\, a$, $P_0 \,\mathsf{rs}\, a$, $P_0 \,\mathsf{tie}\, b$, $P_0[\mathcal{X}_i \leftarrow P_i \mid 1 \leq i \leq k]$ ou $P_0[f]$, pour des PM-nets P_i $(0 \leq i \leq k)$, un symbole d'action $a \in \mathbb{A}$, un symbole de liens $b \in \mathbb{B}$, des symboles d'actions hiérarchiques $\mathcal{X}_i \in \mathbb{X}$ $(1 \leq i \leq k)$ et une fonction de renommage f.

Remarquons que $P = P_0 \,\mathsf{sc}\, a$ est aussi un PM-net. De même, un réseau construit avec l'un des opérateurs de contrôle de flot à partir de PM-nets est aussi un PM-net (puisque ces opérateurs sont basés sur la substitution de transitions). Par ailleurs, la définition des opérateurs π'_s et π'_a n'utilise que des constructions autorisées dans l'algèbre des PM-nets, et donc, l'application de ces opérateurs à des PM-nets produit des PM-nets.

Remarquons enfin que tout M-net peut être vu comme un PM-net : il suffit de lui ajouter une relation de priorités vide. D'autre part, si nous n'utilisons pas les opérateurs π_s et π_a, la relation de priorités d'un PM-net reste vide (car ces opérateurs sont les seuls à introduire des priorités). Sous l'angle de ces deux remarques, nous pouvons considérer que l'algèbre des PM-nets contient celle des M-nets. C'est donc un modèle aussi riche que celui des M-nets ; nous pouvons en particulier y utiliser les horloges causales définies au chapitre précédent.

6.4.1 Propriétés

Théorème 7.
Soit $P = (N, \rho)$ un PM-net. Alors :

1. *La relation de priorités ρ est bien formée.*
2. *Le PM-net $P \,\mathsf{rs}\{resume\}$ est bien formé.*

Preuve. La première propriété se prouve par induction sur la structure des PM-nets. La propriété est évidente pour $P = (N, \varnothing)$. On suppose par hypothèse d'induction que c'est aussi le cas pour les P_i $(1 \leq i \leq k)$. Les autres cas sont donnés ci-dessous, nous reprenons les notations de la définition en supposant de plus que $P_i = (N_i, \rho_i)$ avec $N_i = (S_i, T_i, \iota_i)$ (pour $0 \leq i \leq k$).

(1) $P = P_0 \,\mathsf{rs}\, a$. Évident car $\rho \subseteq \rho_0$, la restriction pouvant seulement supprimer des transitions de T_0.

(2) $P = P_0[\mathcal{X}_i \leftarrow P_i \mid 1 \leq i \leq k]$. Nous avons montré à la section 6.2 que la substitution ne peut pas créer de cycle (voir figure 6.3). Par hypothèse d'induction, il n'en préexiste pas.

(3) $P = \pi_s(P_0)$. Le graphe de ρ est schématisé à gauche de la figure 6.14, il ne contient pas de cycle puisque, par hypothèse d'induction, le graphe de ρ_0 n'en contient pas.

(4) $P = \pi_s(P_0)$. Le graphe de ρ est schématisé à droite de la figure 6.14, il ne contient pas de cycle puisque, par hypothèse d'induction, le graphe de ρ_0 n'en contient pas et que le graphe de chaque $\rho_0|_{\bullet s_i}$ y est inclus (pour $1 \leq i \leq k$).

(5) $P = P_0 \,\mathsf{sy}\, a$. Nous avons montré à la section 6.2 que la seule façon d'introduire un cycle avec la synchronisation est qu'il existe une chaîne de priorités $t_1 \prec \cdots \prec t_n$ (avec $n \geq 3$) telle que t_1 et t_n peuvent se synchroniser. Avec les PM-nets, les seules transitions telles que t_n pouvant être introduites sont t_s dans le réseau P_s, et t_a ainsi que les transitions t_a^s dans P_a, utilisées respectivement pour définir π_s et π_a. Or, t_s ne peut pas être synchronisée car elle ne porte pas d'action et nous avons déjà montré au point (4) que la synchronisation cachée dans l'opérateur π_a n'introduit pas de cycle et qu'elle est suivie d'une restriction qui supprime t_a et les t_a^s.

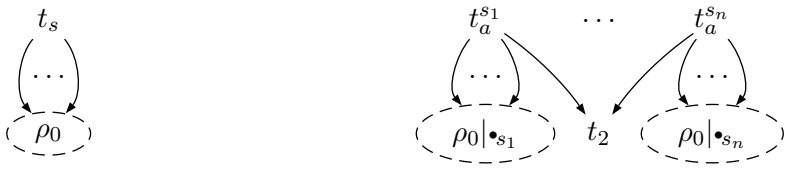


Figure 6.14 — À gauche, le graphe de la relation de priorités de $\pi_s(P)$; la transition t_s est celle de P_s. À droite le graphe de la relation de $\pi_a(P)$. La notation $\rho_0|_{\bullet s_i}$ représente la relation ρ_0 restreinte aux transitions de ${}^\bullet s_i$; les $t_a^{s_i}$ sont les transitions résultant de la synchronisation sur *clear*. Les graphes des $\rho_0|_{\bullet s_i}$ ne sont pas nécessairement disjoints.

Pour prouver le second point du théorème, il suffit de montrer que $N \,\mathsf{rs}\, \mathit{resume}$ est un M-net bien formé puisque nous venons de prouver que ρ est nécessairement bien formée. Au cours de la définition des M-nets à priorités, seule la T-restriction a été violée (l'exorientation l'a été durant la définition de π_a mais c'était temporaire), et seulement dans les réseaux P_s et P_a (figure 6.6 et 6.10). Toutes les transitions non T-restreintes portent donc l'une des actions $\widehat{\mathit{sleep}}$, $\widehat{\mathit{abort}}$ ou $\widehat{\mathit{resume}}$. Nous examinons ces cas séparément.

(1) $\widehat{\mathit{sleep}}$. Il s'agit d'une transition t_1 issue du réseau P_s lors d'une application de π_s. Pour un PM-net P, l'application $\pi_s(P)$ comporte un scoping sur *sleep*, c'est-à-dire une synchronisation suivie d'une restriction. La transition t_1 peut se synchroniser avec une transition t de P qui porte l'action *sleep*. D'après la remarque ci-dessus, cette transition est T-restreinte. Le résultat de la synchronisation de t_1 et t l'est donc aussi. Enfin, la restriction sur *sleep* supprime t_1.

(2) $\widehat{\mathit{abort}}$. Ce cas est similaire au précédent, la seule différence est qu'il concerne l'opérateur π_a.

(3) $\widehat{\mathit{resume}}$. La restriction sur *resume* supprime ces transitions. □

Nous montrons maintenant que les opérateurs π_s et π_a sont bien définis. Pour un PM-net P, cela signifie que $\pi_s(P)$ se comporte comme P sauf qu'il peut être suspendu et que $\pi_a(P)$ se comporte comme P sauf qu'il peut être avorté.

Plus précisément, soit t une transition créée par la synchronisation sur *sleep* lors de l'opération $\pi_s(P)$. Si t est franchie, alors, les seules transitions franchissables dans $\pi_s(P)$

sont t_s et t_2 issues de P_s. En effet, $\pi_s(P)$ démarre son exécution à partir d'un marquage initial dans lequel la place s_s est vide. Elle ne devient marquée que lorsque t (ou une autre transition créée par la synchronisation sur *sleep*) est franchie ce qui rend t_s et t_2 franchissable. Ensuite, tant que t_2 n'est pas franchie, t_s peut l'être et donc, grâce à la priorité $t_{\mathcal{X}} \prec_{\rho_s} t_s$ et à la substitution de $t_{\mathcal{X}}$ par P, toutes les transitions issues de P sont inhibées. Le franchissement de t_s ne change rien à cette situation. Il est important de noter que t_s ne peut pas être franchie en même temps qu'une autre transition t' (qui ne peut pas être t_2 à cause du conflit sur s_s). En effet, le step $\{t_s, t'\}$ serait incohérent puisque $\{t'\}\{t_s\}$ est une de ses linéarisations et qu'elle viole la priorité $t' \prec_{\rho_s} t_s$.

En ce qui concerne l'avortement, soit t une transition créée par la synchronisation sur *abort* lors de l'opération $\pi_a(P)$. Si t est franchie, alors $\pi_a(P)$ atteint son marquage de sortie sans franchir une seule transition issue de P. En effet, lors du franchissement de t, un jeton est placé dans s_a ce qui a pour effet de rendre franchissables les transitions t_a^s (ajoutées lors de l'opération $\mathsf{clr}(P)$) telles que s contient au moins un jeton. Ainsi, chaque transition $t' \in s^\bullet$ qui aurait été franchissable est inhibée par la franchissabilité de t_a^s. Tant que s contient des jetons, la transition t_a^s reste franchissable puisqu'elle est synchronisée à t_a qui boucle sur la place s_a. La même remarque que pour t_s s'applique sur l'impossibilité de franchir une transition en concurrence avec t_a^s. De cette façon, tous les jetons de $\pi_a(P)$ vont être consommés, à l'exception de celui qui se trouve dans s_a. À ce moment, la transition t_2 peut être franchie et produire le marquage de sortie de $\pi_a(P)$. Elle ne peut pas être franchie plus tôt puisque nous avons $t_2 \prec_{\rho_a} t_a$.

Théorème 8.
Soit $P = (N, \rho)$ un PM-net. Alors, N est sûr.

Preuve. En ce qui concerne les marquages, la seule différence entre les M-nets et les PM-nets concerne les opérateurs π_s et π_a. Examinons ces cas séparément.

(1) Dans le cas de π_s, le seul problème pourrait venir de la transition t_1 dans P_s qui n'a pas d'arc entrant. Dans la définition de π_s, le scoping sur *sleep* nous assure que cette transition n'engendre que des transitions T-restreintes et que t_1 est supprimée. Par ailleurs, il ne peut y avoir d'auto-concurrence de ces transitions car le step résultant serait incohérent (puisqu'un jeton dans s_s rend t_s franchissable). Enfin, la production d'un second jeton dans s_s par une de ces transitions est impossible puisque le premier jeton rend t_s franchissable et suspend donc toutes les transitions susceptibles de produire des jetons dans s_s.

(2) Le cas de la transition t_1 impliquée dans la définition de π_a est similaire. D'autre part, cet opérateur manipule directement le marquage lors d'un avortement. Nous avons montré qu'il était bien défini et que l'avortement conduit toujours au marquage de sortie. Donc, aucune accumulation de jetons n'est possible entre deux exécutions d'un réseau avortable (au sein d'une boucle). Remarquons pour finir qu'au cours de l'avortement, aucun jeton ne peut être créé à part lors du franchissement de t_2 qui supprime le jeton dans s_a pour en produire un dans la place de sortie du réseau avorté. □

Expressivité du modèle à priorités. Nous montrons maintenant que le modèle des PM-nets n'est pas plus expressif que celui des M-nets mais qu'il en constitue une abréviation (exponentielle en taille). Cette propriété n'est valable que pour des PM-nets finis, en nombre de places et de transitions, mais aussi n'utilisant que des types finis pour leurs places. En effet, l'utilisation des types infinis permet d'obtenir le pouvoir d'expression des machines de Turing sans même avoir recours aux priorités. De même, les priorités alliées à des places non bornées amènent la même augmentation d'expressivité, c'est pourquoi, dans tout ce paragraphe, nous fixons un entier $k \geq 1$ et nous ne considérons que des PM-nets dont les parties réseaux sont des k-nets (voir section 3.4) n'utilisant que des types finis.

Pour commencer, nous étendons le dépliage présenté à la section 3.2.3 pour tenir compte des priorités. Nous considérons ici les PM-nets comme des réseaux colorés à priorités (les seules annotations considérées sont donc les types des places, les gardes des transitions et les annotations des arcs). Soit $P = (N, \rho)$ un PM-net tel que $N = (S, T, \iota)$, le dépliage $\mathsf{unf}(P)$ est la paire $(\mathsf{unf}(N), \mathsf{unf}(\rho))$, où $\mathsf{unf}(N)$ est le dépliage du réseau coloré N et :

$$\mathsf{unf}(\rho) \stackrel{\text{df}}{=} \{(t_\sigma, t'_{\sigma'}) \mid t \prec_\rho t',\, t_\sigma \in \mathsf{unf}(t) \text{ et } t'_{\sigma'} \in \mathsf{unf}(t')\} \quad .$$

Grâce au dépliage ainsi étendu, nous pouvons obtenir à partir de tout PM-net fini et borné un réseau places/transitions équipé d'une relation de priorités. Ce réseau est fini et borné puisque le PM-net l'est aussi. Ces réseaux sont ceux considérés dans [BK92] et les auteurs montrent qu'ils peuvent être transformés en réseaux finis et bornés sans priorités possédant la même sémantique cohérente [BK92, section 7]. Par ailleurs, il est prouvé dans [BW00] qu'un réseau fini et borné peut être transformé en réseau sûr en préservant sa sémantique *pomset* (multi-ensembles d'ordres partiels) dont la sémantique steps est une restriction.

Nous voyons donc qu'un PM-net fini et borné peut être transformé en un réseau place/transition fini et 1-borné ayant la même sémantique concurrente (séquences de steps cohérents). Cette transformation (en particulier la dernière étape) grandit exponentiellement le nombre de places et de transitions. Néanmoins, elle permet de prouver que nous disposons bien d'une abréviation, pour peu qu'on se restreigne à des réseaux finis et bornés. Cette restriction est très légère puisqu'elle correspond à ne considérer que des systèmes finis, ce qui est généralement ce qu'on veut étudier.

7 Sémantique des langages de programmation

Ce chapitre est consacré à la sémantique des langages de programmation. Nous présentons un langage de programmation parallèle dont la sémantique est traditionnellement donnée en termes de M-nets. Ensuite, nous montrons comment les différentes adjonctions faites à ce modèle (liens asynchrones, temps causal et préemption) permettent d'étendre le langage de programmation avec de nouvelles constructions, en particulier avec des tâches, des exceptions et des délais. Nous aboutissons après ces extensions à un langage temps-réel complet.

L'une des motivations de notre travail dans les M-nets est de pouvoir les utiliser pour donner la sémantique formelle de langages de programmation. Ce chapitre illustre cette intention en décrivant plusieurs applications au langage *B(PN)²* (*Basic Petri Net Programming Notation*) [BH93]. Il s'agit d'un langage de programmation à la syntaxe simple et dont la sémantique est depuis le début donnée en termes de réseaux de Petri composables (avec des réseaux places/transitions au départ, puis avec des M-nets). B(PN)^2 propose les constructions classiques des langages de programmation (boucles, conditionnelles, séquence), des constructions liées au parallélisme et aux communications entre blocs parallèles (mise en parallèle, canaux FIFO, variables partagées) mais aussi des procédures qui peuvent être appelées récursivement ou en concurrence (avec des paramètres passés par valeur, par retour ou par références).

B(PN)^2 est intégré au système PEP [GB96, Gra97] qui permet, entre autres choses, d'éditer des programmes dans ce langages, d'obtenir automatiquement leur sémantique réseau de Petri et de vérifier des propriétés sur les réseaux ainsi obtenus (par *model checking* ou par simulation).

Le langage B(PN)^2 *n'est pas* un langage de programmation utilisé dans des applications concrètes. Jusque-là, il a été utilisé plutôt comme notation pour spécifier des systèmes et les vérifier [Bes97, Gra95]. Cependant, nous considérons B(PN)^2 comme un langage de test : en appliquant nos idées à ce langage, nous pouvons les mettre en pratique et vérifier leur pertinence. Cette application donne des pistes pour transférer les approches éprouvées sur B(PN)^2 à d'autres langages de programmation. Pour le moment, seul B(PN)^2 a été étudié, mais nous pensons que notre approche est adaptable au langage Ada que nous envisageons d'étudier.

Dans la suite de ce chapitre, nous commençons par présenter B(PN)^2, puis nous donnons la sémantique de ses constructions clefs. (Sans donner la sémantique complète car elle est pour certains points assez complexe et ce degré de détail nuirait à notre propos qui est d'illustrer les applications possibles de nos extensions des M-nets.) Nous donnons ensuite quelques applications tirant parti des contributions présentées dans les chapitres précédents.

7.1 Le langage B(PN)^2 et sa sémantique

B(PN)^2 est un langage de programmation parallèle, sa syntaxe est simple et il propose la plupart des constructions classiques pour ce type de langages :

- constructions de contrôle de flot : séquence, boucles, conditionnelles et mise en parallèle ;
- variables typées (le contrôle de type revient à assurer la compatibilité des valeurs) ;
- canaux de communication FIFO ;
- procédures avec paramètres (passés par valeur, par retour ou par référence) pouvant être appelées récursivement et aussi en parallèle ;
- blocs permettant de limiter la portée des déclarations ;
- actions atomiques.

La figure 7.1 donne l'extrait de la syntaxe de B(PN)^2 sur lequel nous travaillons dans la suite.

program	::=	**program** block
block	::=	**begin** scope **end**
scope	::=	com | decl ; com
com	::=	⟨expr⟩ | proc-call | block
	|	com ‖ com | com ; com | **do** alt-set **od** | (com)
decl	::=	**var** *name* : *set* | **var** *name* : **chan** *k* **of** *set*
	|	**procedure** *name* (fpl) *block* | decl , decl
proc-call	::=	*name* (epl)
alt-set	::=	com ; **repeat** | com ; **exit** | alt-set alt-set

Figure 7.1 — Un extrait de la syntaxe de B(PN)^2. Les mots-clefs sont indiqués en **gras**, les non-terminaux de la grammaire en sans sérif et on trouve en *italique* les valeurs fournies par le programme.

Un programme B(PN)^2 débute par le mot clef **program** et continue par un bloc. Le bloc est la construction de base ; il est délimité par les mots-clefs **begin** et **end** ; il comporte généralement des déclarations (decl) suivies d'une commande (com).

Les déclarations sont locales au bloc où elles sont faites ; pour déclarer une variable on lui donne un nom (*name*) et on indique l'ensemble de valeurs qu'elle peut prendre (*set*, une partie de l'ensemble *Val* utilisé pour les M-nets). Pour déclarer un canal de communication (**chan**) on indique son nom (*name*), sa capacité (k, un entier naturel ou

la valeur ∞ pour un canal non borné) et le type des valeurs qu'il peut transporter (*set*, aussi inclus dans *Val*). Notons que la communication sur un canal de capacité nulle est synchrone : la lecture et l'écriture doivent se faire simultanément. On peut enfin déclarer des procédures ayant un nom, une liste de paramètres formels (fpl) et un bloc constituant leur corps (avec leurs déclarations).

Une commande (com) peut être une action atomique ($\langle$expr$\rangle$) qui permet de lire et écrire des variables et des canaux. Une variable v peut y apparaître en tant que ${}'v$ ou v' qui représentent respectivement sa valeur avant et après l'exécution de l'action atomique. Un canal c peut apparaître comme $c?$ ou $c!$ qui représentent respectivement une valeur reçue ou envoyée sur le canal. Ces notations sont combinées sous la forme d'une expression booléenne qui doit pouvoir être évaluée à vrai pour que l'action atomique puisse s'exécuter. Par exemple, l'action $\langle {}'v > 0 \wedge v' = c? \rangle$ correspond à une communications gardée : pour l'exécuter, il faut que la valeur de la variable v soit strictement positive (${}'v > 0$) et qu'une lecture soit possible sur le canal c ($c?$) ; l'exécution de l'action assigne alors à v la valeur lue sur c (avec $v' = c?$).

Une commande peut aussi être un appel de procédure (proc-call) qui est fait en donnant le nom de la procédure suivi de la liste des paramètres effectifs (epl). On peut aussi utiliser un bloc comme commande, ce qui autorise à introduire de nouvelles déclarations pouvant masquer celles qui existent (comme on en a l'habitude avec les langages orientés blocs). Les commandes plus complexes sont formées au moyen des différentes constructions de contrôle de flot qui peuvent être arbitrairement imbriquées en utilisant des parenthèses. On dispose d'une composition parallèle ($\|$), d'une composition séquentielle (;) et d'une construction permettant de représenter toutes sortes de boucles et de conditionnelles : on place entre les mots-clefs **do** et **od** une liste de clauses (alt-set). Chaque clause peut être une répétition (**repeat**), ce qui entraîne après son exécution une nouvelle évaluation du bloc **do**...**od**, ou bien une terminaison (**exit**) dont l'exécution termine le bloc **do**...**od**. Chaque clause **repeat** ou **exit** débute typiquement par une action atomique dont l'exécutabilité détermine celle de la clause entière. Lorsque plusieurs clauses sont exécutables, le choix entre elles est non déterministe.

7.1.1 Sémantique M-net de B(PN)^2

La définition de la sémantique M-net des programmes B(PN)^2 est donnée dans [BH93] par une fonction sémantique mnet, récursive sur la structure syntaxique des programmes.

La sémantique d'un bloc s'obtient en juxtaposant (en parallèle) la sémantique de ses déclarations et celle de sa commande. La sémantique de la commande est suivie (en séquence) par des M-nets permettant de terminer les déclarations (leur rôle est similaire au M-net N_t utilisé pour terminer une horloge causale, voir par exemple la section 4.1.1 page 66). On applique ensuite un scoping sur toutes les actions M-net permettant au M-net de la commande de communiquer avec les M-nets des variables du bloc. Plus formellement, cette construction est définie ainsi :

$$\mathsf{mnet}(\textbf{begin}\ \mathit{decl}\ ;\ \mathit{com}\ \textbf{end}) \stackrel{\text{df}}{=} \Big((\mathsf{mnet}(\mathit{com}); \mathsf{term}(\mathit{decl})) \Big\| \mathsf{mnet}(\mathit{decl})\Big)\, \mathsf{sc}\,(\mathsf{act}(\mathit{decl}))\quad , \tag{111}$$

où $\mathsf{term}(decl)$ permet la terminaison des déclarations *decl* et $\mathsf{act}(decl)$ est l'ensemble des symboles d'actions de $\mathbb{A}$ permettant la communication entre les déclarations *decl* et la commande *com* ainsi que les actions permettant les terminaisons.

L'accès à une variable v est représenté par l'action $V(v^i, v^o)$ qui correspond au changement de la valeur stockée par v de v^i (*input*) à v^o (*output*). Chaque déclaration de variable a pour sémantique un M-net de donnée du type correspondant ; le réseau à gauche de la figure 7.2 en donne un exemple. Nous pouvons en déduire le M-net permettant de terminer une variable : il est donné à droite de la figure.

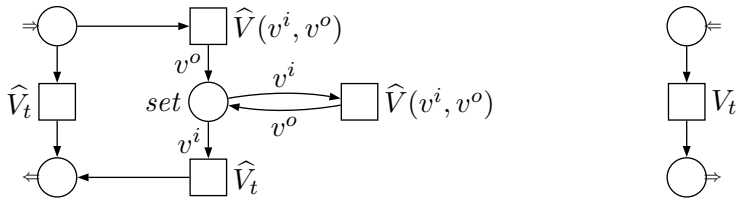


Figure 7.2 — À gauche, le M-net $\mathsf{mnet}(\textbf{var}\ v : set)$ donnant la sémantique de la déclaration de la variable v de type *set*. À droite, le M-net $\mathsf{term}(\textbf{var}\ v : set)$ permettant de terminer le M-net de gauche.

La sémantique d'un canal est plus complexe mais suit un principe similaire : au niveau d'une action atomique l'écriture sur le canal c est représentée par une action M-net $C!(x)$ et la lecture par une action $C?(x)$. La sémantique des canaux est donnée figure 7.3 pour les différentes capacités possibles. Le M-net permettant la terminaison du M-net sémantique d'un canal est similaire à celui présenté à droite de la figure 7.2 : il suffit d'y changer V_t en C_t.

La sémantique d'une action atomique est donnée par un M-net comportant une seule transition portant les actions M-net permettant de communiquer avec les variables et les canaux impliqués dans l'action atomique ; la garde est obtenue à partir de l'expression dans laquelle on remplace les notations $\mathrm{B(PN)}^2$ par les noms de variables M-net correspondant. La figure 7.4 donne la sémantique de l'action atomique $\langle {}'v > 0 \wedge v' = c? \rangle$ présentée plus haut.

La sémantique de la composition parallèle de deux commandes est la composition parallèle des sémantiques des commandes ; de même, la sémantique de deux commandes en séquence est obtenue comme la composition séquentielle des M-nets donnant les sémantiques des commandes. La sémantique de la construction **do**. . .**od** fait intervenir les opérateurs d'itération et de choix : les sémantiques des commandes des différentes clauses **repeat** sont composées par l'opérateur de choix et placées dans la partie répétition de l'itération ; les sémantiques des commandes des clauses **exit** sont aussi composées avec le choix et placées dans la partie terminaison de l'itération ; pour la partie initialisation de l'itération, on utilise une action silencieuse.

La sémantique des déclarations et des appels de procédure est complexe (voir [Kla00, Kla01] pour tous les détails). Nous ne détaillons pas cet aspect car l'introduction de la préemption va nous amener à le simplifier considérablement (voir section 7.4.3).

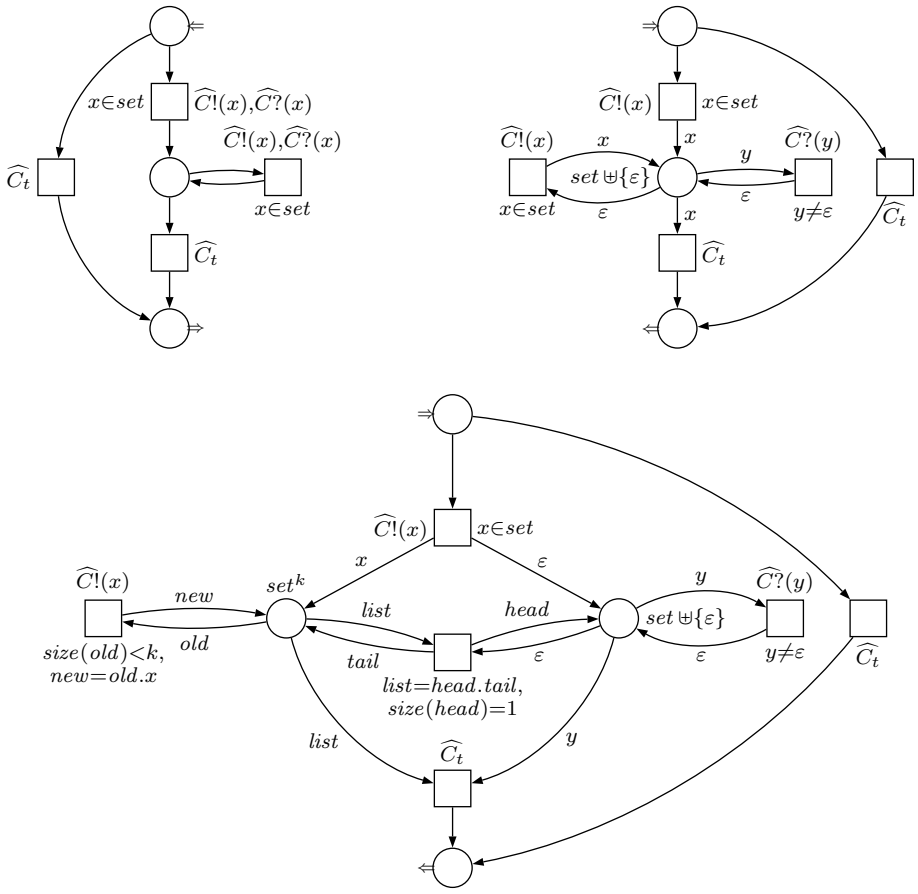


Figure 7.3 — Les différentes sémantiques pour un canal c de capacité k avec des valeurs dans set. En haut à gauche pour $k = 0$, en haut à droite pour $k = 1$ et en bas pour $k > 1$.

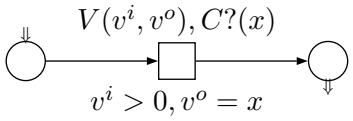


Figure 7.4 — Le M-net $\mathsf{mnet}(\langle' v > 0 \wedge v' = c?\rangle)$.

7.2 Amélioration de la sémantique

La motivation première pour ajouter les liens asynchrones aux M-nets a été de permettre une transmission simple et efficace des identificateurs associés aux compteurs d'horloges causales entre différentes parties d'un réseau temporisé. Il est apparu par la suite qu'une autre application était envisageable : les liens asynchrones permettent de séparer dans un réseau les données du contrôle. En utilisant les places de liens pour stocker toutes les données, les places de contrôle sont alors toutes construites par les opérateurs de contrôle

de flot et ont toujours le type $\{\bullet\}$. En se basant sur cette idée, la sémantique complète de B(PN)^2 a pu être donnée grâce à une sous-classe des M-nets ne faisant appel qu'à des réseaux très simples et aux opérateurs de l'algèbre. Nous avons introduit cette idée dans [2] et l'avons appliquée à la sémantique des canaux. Ces principes ont ensuite été généralisés dans [Kla00] où l'algèbre de M-nets simplifiée est formellement définie (il ne s'agissait que d'une notation dans [2]) et utilisée pour donner la sémantique complète de B(PN)^2. Nous présentons ici l'application aux canaux FIFO.

Nous notons $\alpha.\beta.\gamma$, un M-net de la forme de celui de la figure 7.5, où $\alpha.\gamma$ est l'étiquette de la transition et β est une liste de liens asynchrones représentant les arcs entre t et les places de liens. Un lien de la forme $b^+(x)$ représente un arc étiqueté x allant de la transition t à la place de liens de statut b et un lien $b^-(x)$ représente l'arc équivalent mais dans l'autre sens. Nous choisissons comme types des places de liens les ensembles minimaux permettant de franchir la transition de $\alpha.\beta.\gamma$ sans que les places de liens n'ajoutent ou ne suppriment de comportements à cause de leurs types.

Avec cette notation, nous pouvons représenter simplement la sémantique d'une action de base (qui ne fait pas intervenir de lien asynchrone), nous aurons par exemple :

$$\mathsf{mnet}(\langle {}'v > 0 \wedge v' = c? \rangle) = \{V(v^i, v^o), C?(x)\}.\varnothing.\{v^i > 0, v^o = x\} \quad .$$

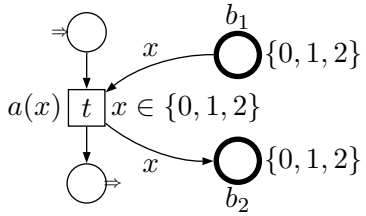


Figure 7.5 — Le M-net noté $\{a(x)\}.\{b_1^-(x), b_2^+(x)\}.\{x \in \{0,1,2\}\}$, où $a \in \mathbb{A}$ et b_1 et b_2 sont dans $\mathbb{B}$.

7.2.1 Sémantique des canaux

La sémantique d'un canal de capacité nulle ne fait pas appel aux liens asynchrones ; en effet, puisque la communication est synchrone, il n'y a pas de donnée à stocker et il suffit d'assurer la synchronisation sur une seule transition. Nous définissons alors :

$$\begin{aligned}\mathsf{mnet}(\mathbf{var}\ c\ \mathbf{chan}\ 0\ \mathbf{of}\ set) \stackrel{\text{df}}{=} \{\widehat{C_t}\}.\varnothing.\{\top\} \,\square\, \Big[\ & \{\widehat{C!}(x), \widehat{C?}(x)\}.\varnothing.\{x \in set\} \\ & * \ \{\widehat{C!}(x), \widehat{C?}(x)\}.\varnothing.\{x \in set\} \\ & * \ \{\widehat{C_t}\}.\varnothing.\{\top\} \ \Big] \quad .\end{aligned}$$

Le M-net correspondant est exactement celui présenté en haut à gauche de la figure 7.3.

Pour les canaux de capacité infinie, nous pouvons stocker les données envoyées sous forme de paires (x, n) où $x \in set$ est la donnée et $n \in \mathbb{N}$ est son numéro d'ordre. La sémantique d'un canal est alors composée de deux itérations en parallèle communiquant

par une place de liens où sont stockées les valeurs numérotées. Chaque itération maintient dans une place de liens le numéro de la prochaine valeur à recevoir ou envoyer. Nous définissons :

$$\mathsf{mnet}(\mathbf{var}\ c\ \mathbf{chan}\ \infty\ \mathbf{of}\ set) \stackrel{\text{df}}{=} \Big(\{\widehat{C_t}\}.\varnothing.\{\top\};\{\widehat{C'_t}\}.\varnothing.\{\top\}\Big) \\ \square\ \Big(core\ \mathsf{sc}\ \{I,T\}\ \mathsf{tie}\ \{d,s,r\}\Big) \quad ,$$

où les places de liens de statuts d, s, r servent respectivement à stocker les données, le numéro de la prochaine donnée à envoyer et celui de la prochaine à recevoir. Les actions I et T permettent de synchroniser l'initialisation et la terminaison des parties dédiées à la réception et à l'envoi de données. Ces deux parties sont des itérations parallèle comme le montre la définition de *core* :

$$core \stackrel{\text{df}}{=} [init * send * terminate] \parallel [wait_i * receive * wait_t]$$

L'itération de gauche est dédiée aux envois de données sur le canal ; elle débute par une initialisation qui a lieu lors du premier envoi ; la partie répétée permet d'autres envois ; la partie de terminaison sert à vider toutes les places de liens. L'itération de droite est dédié à la réception de données : son initialisation et sa terminaison servent uniquement à se synchroniser avec celles de l'autre partie, au moyen des actions I et T ; la partie répétée effectue toutes les réceptions. Nous définissons alors :

$$\begin{aligned}
init &\stackrel{\text{df}}{=} \{I, \widehat{C!}(x)\}.\{r^+(0), s^+(1), d^+((x,0))\}.\{x \in set\} \\
wait_i &\stackrel{\text{df}}{=} \{\widehat{I}\}.\varnothing.\{\top\} \\
send &\stackrel{\text{df}}{=} \{\widehat{C!}(x)\}.\{s^-(n), s^+(n+1), d^+((x,n))\}.\{x \in set, n \in \mathbb{N}\} \\
receive &\stackrel{\text{df}}{=} \{\widehat{C?}(x)\}.\{r^-(n), r^+(n+1), d^-((x,n))\}.\{x \in set, n \in \mathbb{N}\} \\
wait_t &\stackrel{\text{df}}{=} \{\widehat{T}\}.\varnothing.\{\top\} \\
terminate &\stackrel{\text{df}}{=} \Big[\ \{\widehat{C_t}\}.\varnothing.\{\top\}\ *\ \varnothing.\{r^-(n), r^+(n+1), d^-((x,n))\}.\{x \in set, n \in \mathbb{N}\} \\
&\qquad * \{\widehat{C'_t}\}.\{r^-(n), s^-(m)\}.\{n = m, n \in \mathbb{N}\}\ \Big] \quad .
\end{aligned}$$

L'envoi d'une valeur x sur le canal est pris en compte par l'action $\widehat{C!}(x)$: à ce moment, la valeur numérotée (x,n) est stockée dans la place de liens de statut d ; le compteur n pour les envois est pris dans la place de liens de statut s, incrémenté et remis dans la place de liens. Le premier envoi est similaire mais initialise les deux compteurs au lieu d'incrémenter celui des envois. La réception est symétrique et utilise la place de liens de statut r pour son compteur ; la réception peut être simultanée à l'envoi d'une valeur. La terminaison (avec *terminate*) est initiée par une synchronisation sur l'action C_t, puis elle simule la réception de valeurs de façon répétée, la fin de l'itération récupère les deux compteurs et ne peut avoir lieu que s'ils sont égaux, c'est-à-dire si toutes les valeurs envoyées ont été récupérées par la simulation de réception.

Pour terminer un canal, nous devons maintenant utiliser le réseau $\{C_t\}.\varnothing.\{\top\};\{C'_t\}.\varnothing.\{\top\}$ qui permet la synchronisation avec le démarrage et la terminaison de la partie *terminate*.

Afin d'implanter un canal de capacité k avec $0 < k < \infty$, nous simulons une file circulaire : les paires (x, n) deviennent des triplets (x, n, z) où $z \in \{\top, \bot\}$ indique si la cellule d'indice n est vide ($z = \bot$) ou déjà utilisée par une valeur ($z = \top$). De cette façon, nous évitons une comparaison entre les compteurs (réalisés par les liens sur s et r) ce qui supprimerait la concurrence entre une réception et un envoi. Ces compteurs sont gérés modulo k. Nous avons alors :

$$\mathsf{mnet}(\textbf{var } c \textbf{ chan } k \textbf{ of } set) \stackrel{\text{df}}{=} \left(\{\widehat{C_t}\}.\varnothing.\{\top\}\right) \,\square\, \left(core\, \mathsf{sc}\, \{I, T\}\, \mathsf{tie}\, \{d, s, r\}\right) \quad ,$$

avec :

$$\begin{aligned}
core &\stackrel{\text{df}}{=} [init * send * terminate] \parallel [wait_i * receive * wait_t] \\
init &\stackrel{\text{df}}{=} \{I, \widehat{C!}(x)\}.\{r^+(0), s^+(1), d^+((x,0,\top)), d^+((x,1,\bot)), \ldots, \\
&\qquad d^+((x,k-1,\bot))\}.\{x \in set\} \\
wait_i &\stackrel{\text{df}}{=} \{\widehat{I}\}.\varnothing.\{\top\} \\
send &\stackrel{\text{df}}{=} \{\widehat{C!}(x)\}.\{s^-(n), s^+((n+1) \bmod k), d^-((y,n,\bot)), d^+((x,n,\top))\} \\
&\qquad .\{x \in set, 0 \le n < k\} \\
receive &\stackrel{\text{df}}{=} \{\widehat{C?}(x)\}.\{r^-(n), r^+((n+1) \bmod k), d^-((x,n,\top)), d^+((x,n,\bot))\} \\
&\qquad .\{x \in set, 0 \le n < k\} \\
terminate &\stackrel{\text{df}}{=} \{T, \widehat{C_t}\}.\{r^-(n), s^-(m), d^-((x_0,0,z_0)), \ldots, d^-((x_{k-1},k-1,z_{k-1}))\} \\
&\qquad .\{0 \le n < k, 0 \le m < k, x_i \in set, z_i \in \{\bot,\top\} \mid 1 \le 1 < k\} \\
wait_t &\stackrel{\text{df}}{=} \{\widehat{T}\}.\varnothing.\{\top\}
\end{aligned}$$

Cette fois, la terminaison peut se faire sans itération car nous savons exactement combien de jetons doivent être retirés dans chaque place de liens.

7.2.2 Avantages de cette sémantique

Le premier avantage de la sémantique que nous venons de présenter est qu'elle est donnée par des expressions ce qui n'était pas possible auparavant. Nous pouvons d'ailleurs envisager de ne pas présenter du tout de réseaux de Petri et de rester au niveau des expressions. Nous avons constaté, lors des différentes présentations en public de ces travaux, que les expressions sont mieux comprises et par un plus large public que les M-nets.

Le second avantage, non négligeable, est que nous avons considérablement réduit la taille des réseaux donnant la sémantique des canaux. Sans tenir compte des canaux de taille infinie qui nécessitent de toute façon des types de places infinis, considérons le M-net au bas de la figure 7.3 ; les données du canal sont stockées dans une place de type set^k, le nombre de places dans le dépliage sera donc de l'ordre de $|set|^k$. La sémantique que nous avons donné pour les canaux de capacité k fait appel à des places de contrôle,

de type $\{\bullet\}$, deux places de liens pour les compteurs, de type $\{1,\ldots,k\}$, et une place de liens pour les données dont type est $set \times \{0,\ldots,k-1\} \times \{\bot,\top\}$. Le nombre de places dans le dépliage sera donc de l'ordre de $2k|set|$.

Notre sémantique possède encore un avantage : une valeur envoyée sur un canal est immédiatement disponible en réception (mais pas simultanément puisqu'il s'agit d'une communication asynchrone). Avec l'ancienne sémantique, il est nécessaire d'attendre que la transition au centre du réseau du bas de la figure 7.3 ait extrait la tête de la liste stockée dans la place à sa gauche pour la déposer dans la place à sa droite. Tant que cette transition n'est pas franchie, aucune réception n'est possible. Ce défaut n'existe pas avec notre sémantique.

7.3 Extension temporisée

Dans cette section, nous montrons comment intégrer simplement le temps causal à B(PN)^2 avec une nouvelle instruction **delay** permettant d'attendre qu'un certain nombre de tics aient été comptés. On utilise cette instruction en lui donnant en paramètre une valeur entière (« **delay** 5 », par exemple) ou le nom d'une variable de type entier (par exemple « **delay** v » si v a été déclarée avec un type compris dans $\mathbb{N}$).

Nous commençons par changer la sémantique globale d'un programme pour ajouter une horloge causale :

$$\mathsf{mnet}(\textbf{program}\ block) \stackrel{\text{df}}{=} \Big(H_3 \,\Big\|\, (N_i; \mathsf{mnet}(block); N_t)\Big)\,\mathsf{sc}\,\{clk_i, clk_t, clock, ident\} \quad ,$$

où le réseau H_3 est présenté section 4.1.3 page 67 ; N_i et N_t étant donnés figure 1.9 page 23.

Ensuite, pour $n \in \mathbb{N}$ et v une variable B(PN)^2 dont le type *set* ne contient que des entiers, nous pouvons définir simplement :

$$\begin{aligned}
\mathsf{mnet}(\textbf{delay}\ n) \stackrel{\text{df}}{=} \Big(&\{clock(i,c,m,0,n), ident(i,\top,\bot)\}.\{b^+(i)\}.\{G\}\ ; \\
&\{clock(i,c,m,\omega,\omega), ident(i,\bot,\top)\}.\{b^-(i)\}.\{c=m,G\}\Big)\,\mathsf{tie}\,b \\
\mathsf{mnet}(\textbf{delay}\ v) \stackrel{\text{df}}{=} \Big(&\{clock(i,c,m,0,v^i), ident(i,\top,\bot), V(v^i,v^o)\}. \\
&\qquad\qquad \{b^+(i)\}.\{v^i=v^o, v^i \in set, G\}\ ; \\
&\{clock(i,c,m,\omega,\omega), ident(i,\bot,\top)\}.\{b^-(i)\}.\{c=m,G\}\Big)\,\mathsf{tie}\,b
\end{aligned}$$

où les liens sur $b \in \mathbb{B}$ permettent de communiquer les identificateurs de compteurs entre les deux transitions de chacun des réseaux ; pour les quatre gardes, nous posons $G \stackrel{\text{df}}{=} (i \in Ident) \wedge (c,m \in \mathbb{N})$. Le type des places de liens contient exactement les valeurs possibles pour ces identificateurs. En appliquant la restriction de liens, nous rendons privée la communication des identificateurs.

On peut envisager d'autres façons d'introduire le temps dans B(PN)^2, par exemple au moyen de déclarations de chronomètres pouvant être consultés ou remis à zéro. Un tel ajout ne pose pas de difficulté à partir du moment où la sémantique du programme

est dotée d'une horloge causale très générique. La simplicité avec laquelle nous pouvons ajouter une instruction de délai montre bien que d'autres extensions ne poseraient pas de difficultés particulières.

7.4 Extensions liées à la préemption

Dans cette section, nous tirons parti de l'ajout de la préemption aux M-nets pour introduire de nouvelles possibilités dans B(PN)^2. Il s'agit cette fois d'ajouter un mécanisme d'exceptions et un système de tâches. Bien que techniques, ces extensions se révèlent en fin de compte assez simples à réaliser, ce qui appuie la pertinence de notre choix d'opérateurs de préemption. Nous présentons ici des versions améliorées des applications publiées dans [4] et [5].

Nous allons donner à B(PN)^2 une sémantique en termes de PM-nets, et non plus simplement en termes de M-nets, au moyen d'une fonction sémantique pm. Dans la suite, nous définissons explicitement la sémantique des constructions que nous ajoutons au moyen de cette fonction. Pour certaines constructions existantes (par exemple les blocs), nous donnons une nouvelle sémantique qui remplace l'ancienne. Le reste de la sémantique est obtenu en ajoutant aux M-nets une relation de priorités vide lorsque c'est nécessaire ou en mimant le comportement de mnet (par exemple, on aura encore $\mathsf{pm}(com_1 \| com_2) \stackrel{\text{df}}{=} \mathsf{pm}(com_1) \| \mathsf{pm}(com_2)$). Dans le même esprit d'extension, la notation $\alpha.\beta.\gamma$ utilisée pour les M-nets représente maintenant un PM-net dont la partie réseau est le M-net représenté par $\alpha.\beta.\gamma$ et dont la relation de priorité est vide.

7.4.1 Sémantique des exceptions

Afin d'introduire des exceptions dans B(PN)^2, nous introduisions une nouvelle commande « **throw** e » qui permet de lever l'exception e, qui est une valeur arbitraire dans *Val*. Nous autorisons aussi la syntaxe « **throw** v », où v est une variable B(PN)^2 ; dans ce cas, c'est la valeur de v qui est utilisée. Ensuite, nous changeons la syntaxe des blocs de la façon suivante :

block	::=	**begin** scope **end**
	|	**begin** scope catch-list **end**
catch-list	::=	catch-clause
	|	[catch-clause **or**] **catch-others** [w] [**then** com]
catch-clause	::=	**catch** e [**then** com]
	|	catch-clause **or** catch-clause

où w est une variable de type *Val*, visible depuis le point où elle est utilisée. Les parties entre crochets sont facultatives.

Chaque clause « **catch** e » permet de capturer une exception levée par une clause « **throw** e » quelque part dans le même bloc. En utilisant la forme longue « **catch** e **then** com », on peut indiquer une commande qui est exécutée lorsque l'exception e est capturée. Cette commande constitue donc le traitant de l'exception. La clause **catch-others** est similaire et permet de capturer n'importe quelle exception. Elle permet en

outre de stocker la valeur de l'exception dans une variable w dont le type doit être *Val* (puisqu'une exception est une valeur quelconque de cet ensemble).

La sémantique d'un bloc est alors :

$$\mathsf{pm}(\textbf{begin } \mathit{decl}\ \mathit{com}\ \mathit{cc}\ \textbf{end}) \stackrel{\text{df}}{=} \Big(\Big(\pi_a\Big(\mathsf{pm}(\mathit{com})\ ;\ \{\mathit{term}\}.\varnothing.\{\top\}\Big)\ ;\ \mathsf{term}(\mathit{decl})\Big) \Big\| \Big(\mathsf{pm}(\mathit{cc}) \,\square\, \{\widehat{\mathit{term}}\}.\varnothing.\{\top\}\Big) \Big\| \mathsf{pm}(\mathit{decl})\Big) \; \mathsf{sc}\Big(\{\mathit{catch}, \mathit{term}\} \cup \mathsf{act}(\mathit{decl})\Big)\ [\mathit{throw} \mapsto \mathit{catch}]$$

Ce qui ressemble à ce qu'on avait précédemment sauf que nous avons ajouté la sémantique des clauses **catch** *cc*, en parallèle aux sémantiques de la commande (rendue préemptible avec l'opérateur π_a) et des déclarations. Nous ajoutons aussi de quoi les terminer, grâce à la synchronisation sur *term* lorsqu'aucune exception n'est levée. La sémantique de *cc* est différente selon qu'on utilise **catch-others** ou non. En effet, dans le premier cas, toutes les exceptions sont capturées alors que dans le second cas, il faut propager toute exception qui ne serait pas capturée. Afin de la définir de manière compositionnelle, nous utilisons une fonction sémantique auxiliaire, pm', qui prend en second argument l'ensemble des exceptions déjà capturées. Si $cc = cc_1$ **or** $\ldots$ **or** cc_k, nous définissons alors

$$\mathsf{pm}(cc) \stackrel{\text{df}}{=} \mathsf{pm}'(cc_1 \textbf{ or } \ldots \textbf{ or } cc_k, \varnothing) \quad ,$$

avec :

$$\mathsf{pm}'(cc_i \textbf{ or } cc_{i+1} \ldots, E) \stackrel{\text{df}}{=} \mathsf{pm}(cc_i) \,\square\, \mathsf{pm}'(cc_{i+1} \ldots, E \cup \{e_i\})$$
où e_i est l'exception capturée par cc_i

$$\mathsf{pm}'(cc_k, E) \stackrel{\text{df}}{=} \begin{cases} \mathsf{pm}'(cc_k, E), & \text{si } cc_k \text{ est un } \textbf{catch-others} \\ \mathsf{pm}(cc_k) \,\square\, \{\widehat{\mathit{catch}}(x), \mathit{throw}(x), \mathit{abort}\}.\varnothing.\{x \notin E\}, & \text{sinon} \end{cases}$$

$$\mathsf{pm}'(\textbf{catch-others}, E) \stackrel{\text{df}}{=} \{\widehat{\mathit{catch}}(x)\}.\varnothing.\{x \notin E\}$$

$$\mathsf{pm}'(\textbf{catch-others } w, E) \stackrel{\text{df}}{=} \{W(w^i, w^o), \widehat{\mathit{catch}}(x)\}.\varnothing.\{w^o = x, x \notin E\}$$

$$\mathsf{pm}'(\textbf{catch-others then } \mathit{com}, E) \stackrel{\text{df}}{=} \mathsf{pm}'(\textbf{catch-others}, E); \mathsf{pm}(\mathit{com})$$

$$\mathsf{pm}'(\textbf{catch-others } w \textbf{ then } \mathit{com}, E) \stackrel{\text{df}}{=} \mathsf{pm}'(\textbf{catch-others } w, E); \mathsf{pm}(\mathit{com})$$

Finalement, nous définissons :

$$\mathsf{pm}(\textbf{catch } e) \stackrel{\text{df}}{=} \{\widehat{\mathit{catch}}(e)\}.\varnothing.\{\top\}$$

$$\mathsf{pm}(\textbf{catch } e \textbf{ then } \mathit{com}) \stackrel{\text{df}}{=} \mathsf{pm}(\textbf{catch } e); \mathsf{pm}(\mathit{com})$$

$$\mathsf{pm}(\textbf{throw } e) \stackrel{\text{df}}{=} \{\mathit{catch}(e), \mathit{abort}\}.\varnothing.\{\top\}$$
où e est une valeur dans *Val*.

$$\mathsf{pm}(\textbf{throw } v) \stackrel{\text{df}}{=} \{V(v^i, v^o), \mathit{catch}(x), \mathit{abort}\}.\varnothing.\{x = v^o = v^i\}$$
où v est une variable $\mathrm{B(PN)}^2$ de type *Val*.

Le fonctionnement global est assez simple. L'exécution d'une instruction « **throw** e » entraîne l'avortement de la commande depuis laquelle l'exception est levée (grâce à l'utilisation de π_a et à l'action *abort* dans la sémantique du **throw**). En même temps, la

synchronisation sur *catch* permet de capturer l'exception à l'extérieur de π_a, en parallèle. S'il existe, le traitant correspondant est déclenché à la suite de la capture. Une clause **catch-others** peut servir à capturer et traiter toute exception non déjà prise en compte. Si cette clause est omise et qu'une exception non capturée est levée, la transmission est assurée par le réseau ajouté automatiquement dans un tel cas : après le scoping sur *catch* nous renommons l'action *throw* en *catch* et donc, le réseau ajouté se comporte exactement comme une instruction « **throw** e ».

Lorsque plusieurs exceptions sont levées simultanément, le choix de celle qui est effectivement capturée est non déterministe, les autres exceptions étant ignorées. Ce n'est pas en général satisfaisant mais il est tout à fait possible d'implanter des mécanismes de résolution sophistiqués au dessus du mécanisme de base que nous fournissons.

7.4.2 Sémantique d'une extension multi-tâches.

Nous étendons maintenant B(PN)^2 avec un système de tâches. Une tâche est considérée comme une ressource, au même titre qu'une procédure ou qu'une variable. Les déclarations de tâches peuvent donc avoir lieu à tous les niveaux et même être imbriquées. On peut démarrer une tâche, lui envoyer des signaux ou vérifier qu'elle s'exécute ; elle peut aussi s'avorter ou se suspendre elle-même. Nous étendons la syntaxe avec une nouvelle déclaration et des nouvelles commandes :

$$
\begin{array}{lll}
\mathsf{decl} & ::= & \cdots \mid \textbf{task } \mathit{name}\ \mathsf{block}\ \mathsf{signal} \\
\mathsf{signal} & ::= & \textbf{await } \mathit{sig}\ \textbf{then}\ \mathsf{com} \mid \mathsf{signal}\ \mathsf{signal} \\
\mathsf{com} & ::= & \cdots \mid \textbf{start } \mathit{name} \mid \textbf{signal } (\mathit{id}, \mathit{sig}) \mid \textbf{abort} \mid \textbf{sleep} \\
 & & \mid \textbf{wait } \mathit{id} \mid \textbf{active } \mathit{id}
\end{array}
$$

Une clause « **task** ... » déclare une nouvelle tâche en lui donnant un nom, un corps sous forme d'un bloc et une liste de signaux auxquels elle peut réagir. On peut démarrer la tâche *name* avec l'instruction « **start** *name* » qui crée une nouvelle instance de la tâche, qui commence aussitôt à exécuter son corps. Contrairement à un appel de procédure, qui attend la fin de l'exécution de la procédure, le lancement d'une instance de tâche termine immédiatement et la tâche s'exécute indépendamment. Les tâches peuvent communiquer avec le reste du programme en utilisant les moyens fournis en standard par B(PN)^2. À son démarrage, chaque instance se voit allouer un identificateur (pris dans un ensemble I_{task}) qui lui est accessible comme une ressource *id* en lecture seule. Chaque instance de tâche est responsable de la transmission de son identificateur (contrairement à ce qui se passe, par exemple, dans un système Unix lorsqu'on exécute un *fork*). Les commandes **abort** et **sleep** sont utilisables par une instance de tâche pour avorter et se suspendre respectivement. La commande « **wait** *id* » peut s'exécuter si une instance de tâche désignée par son identificateur a terminé. La commande « **active** *id* » fait exactement le contraire puisqu'elle ne peut s'exécuter que quand une instance tâche est active (mais potentiellement suspendue). Placé dans une séquence, un **wait** permet d'attendre la fin de l'exécution ; en utilisant un **do**...**od**, on peut tester la fin de l'exécution d'une instance. Par exemple, le fragment de programme ci-dessous exécute com_1 si l'instance *id* a terminé et com_2 sinon.

do
  wait id ; com_1 ; **exit**
  active id ; com_2 ; **exit**
od

On peut envoyer un signal à une tâche avec la commande « **signal** (id, sig) » où id identifie la tâche visée et sig est le signal à transmettre (qui peut être n'importe quelle valeur dans Val). Une tâche peut réagir aux signaux qui lui sont envoyés si elle comporte une liste (éventuellement vide) de clauses « **await** sig **then** com », durant cette réaction, le corps de la tâche est suspendu et il reprend son exécution quand la réaction est terminée. Si un signal n'est pas dans la liste, il est simplement ignoré. Par ailleurs, nous supposons qu'il existe des signaux KILL, SUSPEND et RESUME qui ne peuvent pas être capturés par une tâche et qui servent respectivement à avorter, suspendre et reprendre une instance de tâche. Ces signaux seront automatiquement pris en compte par la sémantique.

Afin de gérer les identificateurs de tâches, nous définissons un gestionnaire P_{TIM} qui correspond à une ressource globale du programme. Sa partie réseau est donnée figure 7.6 et sa relation de priorité est vide. Le fonctionnement est très simple : les identificateurs libres sont stockés dans la place i_1 et ceux qui sont alloués sont dans la place i_2 ; une synchronisation sur l'action *alloc* permet d'allouer un nouvel identificateur et une synchronisation sur *free* le libère ; les actions *wait* et *active* permettent de tester l'état d'un identificateurs et seront utilisées pour la sémantique des commandes correspondantes. En limitant la taille de I_{task} nous pouvons limiter le nombre maximum d'instances actives dans un programme.

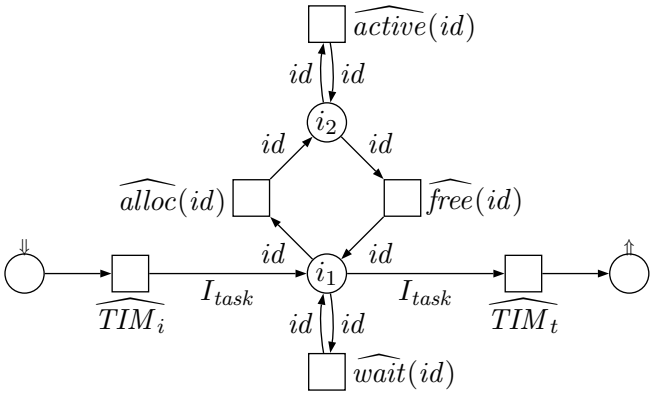


Figure 7.6 — La partie réseau du gestionnaire d'identificateurs P_{TIM}, les places i_1 et i_2 ont le type I_{task}.

Avec ce gestionnaire d'identificateurs, nous pouvons définir une nouvelle sémantique aux programmes :

$$\mathsf{pm}(\textbf{program}\ block) \stackrel{\text{df}}{=} \Big(\Big(\{TIM_i\}.\varnothing.\{\top\}; \mathsf{pm}(block); \{TIM_t\}.\varnothing.\{\top\}\Big) \Big\| P_{TIM}\Big) \\ \mathsf{sc}\,\{sig\mathit{nal}, \mathit{alloc}, \mathit{free}, \mathit{wait}, \mathit{active}, TIM_i, TIM_t\} \quad .$$

Nous pouvons en déduire qu'un programme ne peut se terminer que si toutes les instances de tâches actives ont d'abord terminé (ce qui est aussi le cas en Ada).

Chaque instance de tâche doit stocker localement l'identificateur qui lui est alloué, le PM-net P_{id}, dont la partie réseau est donnée figure 7.7 et dont la relation de priorité est vide, est chargé de ce rôle. La valeur de l'identificateur peut être utilisée sous la forme id dans une action atomique, au niveau de la sémantique, cela correspond à avoir l'action $ID(id)$, de manière similaire à ce qu'on a pour les variables.

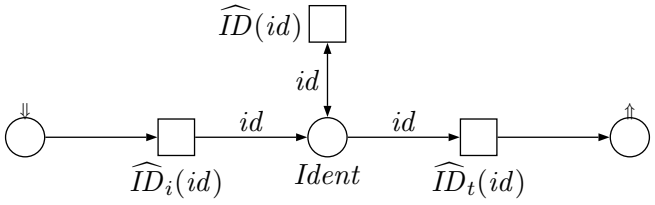


Figure 7.7 — La partie réseau du PM-net P_{id} qui mémorise pour chaque instance de tâche son identificateur.

Afin de pouvoir exécuter plusieurs instances indépendantes d'une tâche $name$, nous utilisons le PM-net P_{name} dont la partie réseau est donnée figure 7.8 et dont la relation de priorité est vide. L'idée est de substituer la transition $t_{\mathcal{X}}$ par le réseau qui définit la tâche proprement dite. De cette façon, une synchronisation sur $NAME$ permet de produire un jeton dans la place i_2 de P_{name} ce qui autorise une nouvelle exécution indépendante du réseau qui a substitué $t_{\mathcal{X}}$. Lorsque cette exécution est terminée, le jeton est renvoyé dans i_1. Nous pouvons contrôler le nombre maximum d'instances pour chaque tâche en limitant la taille de l'ensemble I_{name} associé.

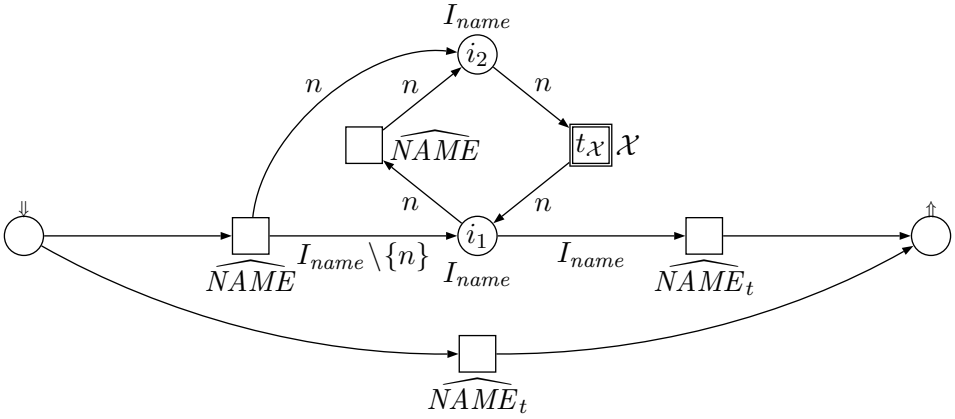


Figure 7.8 — La partie réseau de P_{name} qui gère les différentes instances de la tâche $name$.

La sémantique du démarrage d'une instance est simplement :

$$\mathsf{pm}(\textbf{start}\ name) \stackrel{\text{df}}{=} \{NAME\}.\varnothing.\{\top\} \quad ,$$

qui se synchronise correctement avec les transitions étiquetées $\widehat{NAME}$ dans le réseau P_{name}. Le scoping correspondant est fait au niveau du bloc qui déclare la tâche, comme pour toute autre ressource. De même, un réseau de terminaison $\{NAME_t\}.\varnothing.\{\top\}$ doit aussi être ajouté. Notons que, à l'instar de ce qui se passe pour un programme complet, un bloc ne peut terminer que si toutes ses tâches ont déjà terminé.

La sémantique d'une déclaration de tâche est la suivante :

$$\mathsf{pm}(\textbf{task}\ name\ block\ signals) \stackrel{\text{df}}{=} P_{name}[\mathcal{X} \leftarrow P_{task} \| P_{id}]\, \mathsf{sc}\,\{ID_i, ID_t, ID\} \quad ,$$

avec

$$\begin{aligned} P_{task} \stackrel{\text{df}}{=}\ & \{alloc(id), ID_i(id)\}.\varnothing.\{\top\}\ ; \\ & \pi_a\Big(\Big(\Big(\pi'_s(\mathsf{pm}(block))\ ;\ \{term\}.\varnothing.\{\top\}\Big) \Big\| \mathsf{pm}(signals)\Big) \\ & \qquad \mathsf{sc}\,\{suspend, resume, term\}\Big)\ ; \\ & \{free(id), ID_t(id)\}.\varnothing.\{\top\} \end{aligned}$$

où $\{term\}.\varnothing.\{\top\}$ permet de terminer $\mathsf{pm}(signals)$ qui se charge de capturer les signaux. La sémantique de cette partie, en supposant qu'elle est composée d'une liste de clauses **await** « $aw_1 \dots aw_k$ » est définie ainsi : de façon répétitive, chaque signal est attendu et la réaction correspondante est exécutée ; les signaux KILL, SUSPEND et RESUME sont automatiquement pris en compte, avec les réactions attendues.

La répétition utilisée ici est une itération binaire, ce qui simplifie considérablement l'écriture, nous pouvons l'obtenir à partir de l'itération ternaire et du choix :

$$P_1 * P_2 \stackrel{\text{df}}{=} P_2 \,\square\, [P_1 * P_1 * P_2] \quad .$$

Formellement, nous définissons :

$$\begin{aligned} \mathsf{pm}(aw_1 \dots aw_k) \stackrel{\text{df}}{=} \Big(& \mathsf{pm}(aw_1) \,\square\, \cdots \,\square\, \mathsf{pm}(aw_k) \\ & \square\, \{\widehat{signal}(id, \textsc{suspend}), ID(id), suspend\}.\varnothing.\{\top\} \\ & \square\, \{\widehat{signal}(id, \textsc{resume}), ID(id), resume\}.\varnothing.\{\top\} \\ & \square\, \{\widehat{signal}(id, \textsc{kill}), ID(id), abort\}.\varnothing.\{\top\}\Big) \\ & *(\{term\}.\varnothing.\{\top\}) \end{aligned}$$

$$\begin{aligned} \mathsf{pm}(\textbf{await}\ sig\ \textbf{then}\ com) \stackrel{\text{df}}{=}\ & \{\widehat{signal}(id, sig), ID(id), suspend\}.\varnothing.\{\top\}\ ; \\ & \mathsf{pm}(com)\ ;\ \{resume\}.\varnothing.\{\top\} \end{aligned}$$

Un signal KILL déclenche l'avortement de l'instance de la tâche et de sa partie *signals* qui se trouvent tous les deux sous l'opérateur π_a. Pour les autres signaux, nous pouvons observer qu'ils sont correctement pris en compte, en particulier, la suspension et la reprise de l'instance de tâche sont assurées pendant que le traitant de signal s'exécute.

Finalement, la sémantique des commandes non encore définies est donnée par :

$$\begin{aligned} \mathsf{pm}(\textbf{signal}\ (id, sig)) &\stackrel{\text{df}}{=} \{signal(id, sig)\}.\varnothing.\{\top\} \\ \mathsf{pm}(\textbf{abort}) &\stackrel{\text{df}}{=} \{abort\}.\varnothing.\{\top\} \\ \mathsf{pm}(\textbf{sleep}) &\stackrel{\text{df}}{=} \{sleep\}.\varnothing.\{\top\} \\ \mathsf{pm}(\textbf{wait}\ id) &\stackrel{\text{df}}{=} \{wait(id)\}.\varnothing.\{\top\} \\ \mathsf{pm}(\textbf{active}\ id) &\stackrel{\text{df}}{=} \{active(id)\}.\varnothing.\{\top\} \end{aligned}$$

Nous avons supposé ici que *id* et *sig* sont des valeurs dans *Val* ; on pourrait utiliser à la place des variables B(PN)^2 ; dans ce cas, il faut ajouter à la sémantique les actions permettant de consulter ces variables. Par exemple, si i et s sont deux variables, nous définissons :

$$\mathsf{pm}(\mathbf{signal}(i,s)) \stackrel{\text{df}}{=} \{signal(i^i,s^i), I(i^i,i^o), S(s^i,s^o)\}.\varnothing.\{i^o = i^i, s^o = s^i\} \quad .$$

7.4.3 Procédures dans un contexte préemptible et parallèle

Des difficultés particulières surviennent lorsqu'on introduit la préemption dans un langage de programmation parallèle avec procédures. Si, par exemple, un bloc réalise un appel de procédure et qu'il est avorté par une exception levée dans une branche d'exécution parallèle, il faut avorter l'exécution de l'appel de procédure puisque l'appelant est lui même avorté. De même, si une tâche est suspendue, ses appels de procédures devraient l'être aussi.

Ces aspects ne sont pas pris en compte dans la sémantique que nous avons donnée jusque-là. Dans les langages « classiques » (par exemple Ada et Java), le parallélisme est généralement réalisé par un système de tâches dont chacune est un objet séquentiel qui gère sa propre pile d'appels de procédures. En cas d'avortement ou de suspension, la partie active est alors identifiable sans ambiguïté par un examen de la pile d'appels. Dans le cas de B(PN)^2, on dispose d'une construction de composition parallèle qui rend beaucoup plus difficile la gestion de piles d'appels puisque les parties actives d'un programme peuvent être complètement disséminées, indépendamment d'une structure de tâches.

Un travail conséquent dans cette direction doit donc être réalisé afin d'obtenir une sémantique satisfaisante des appels de procédures. En attendant que ce travail aboutisse et afin de conserver la possibilité d'utiliser des procédures, nous pouvons changer la sémantique des procédures et utiliser du remplacement de code : un appel de procédure correspond alors à un nouveau bloc qui déclare des variables correspondant aux paramètres formels de la procédure ; sa partie commande commence par initialiser ces variables avec les paramètres effectifs de la procédure, puis elle exécute le bloc correspondant au corps de la procédure et finalement, elle réalise les affectations des paramètres passés par retour. De cette façon, un appel de procédure ne dissémine pas le contrôle hors du bloc ou de la tâche qui réalise l'appel et une préemption peut avoir lieu de manière cohérente.

Il est clair que cette sémantique des appels de procédures n'est pas satisfaisante sur le long terme. Nous pensons qu'une gestion précise de piles d'appels dans une version de B(PN)^2 sans composition parallèle, est réalisable en introduisant le parallélisme uniquement à travers des tâches et en ajoutant au niveau de chaque instance un environnement d'exécution chargé de mémoriser la pile d'appels et de prendre en charge la préemption. Une telle approche devrait aussi être développée pour le langage Ada qui nous semble un bon choix pour appliquer nos méthodes.

8 Sémantique opérationnelle

Historiquement, le modèle des M-nets est une extension aux réseaux colorés de *l'algèbre des Petri boxes* (*Petri Box Calculus*, *PBC*) qui a été définie dès le début avec une sémantique réseau (les *Petri boxes*) et une sémantique opérationnelle sur des *expressions* ; ces deux sémantiques étant cohérentes. Les deux modèles ont ensuite évolué assez indépendamment. Du côté des M-nets, on a surtout mis en avant les aspects réseaux, leurs extensions et leurs applications. Du côté Petri boxes, la syntaxe et la sémantique opérationnelle ont été généralisées pour donner *l'algèbre des réseaux de Petri* (*Petri Net Algebra, PNA*).

Les liens asynchrones ont été introduits principalement au niveau des M-nets ; une définition a été donnée au niveau des boxes mais aucune syntaxe n'a été introduite. La motivation de départ des travaux présentés dans ce chapitre est donc double. D'une part, les liens asynchrones ayant prouvé leur utilité, il est apparu important d'en doter les boxes de manière satisfaisante. D'autre part, les liens asynchrones introduisent une mémoire dans les réseaux (entre deux exécutions au sein d'une boucle, un réseau peut garder une trace de ce qu'il a fait, par des jetons dans des places de liens), ce qui a la réputation d'empêcher la compositionnalité des sémantiques. Il fallait donc prouver que ce n'était pas le cas et que les liens asynchrones sont non seulement utiles en pratique mais aussi pertinents du point de vue théorique.

Une motivation secondaire est que nous éprouvons depuis quelques temps le besoin d'une syntaxe des M-nets prenant en compte les récentes extensions. En effet, une telle syntaxe existait sur une version plus simple des M-nets [Kla97, Kla98], mais elle n'a pas évolué avec les M-nets. Il est apparu dans les derniers travaux que l'utilisation d'expressions plutôt que de réseaux permettait généralement de gagner en compacité et en clarté (en particulier, cela nous permet d'être compréhensibles par un plus large public). Ce travail a été amorcé dans [Kla00] où une syntaxe a été définie et appliquée à la sémantique compositionnelle du langage $\mathrm{B(PN)}^2$ (qui pour la première fois est défini entièrement par des expressions). Avant d'entamer la définition d'une sémantique opérationnelle de ces nouvelles expressions, il nous a paru naturel de commencer par celle des boxes avec liens asynchrones. Le résultat est *l'algèbre des boxes asynchrones* (*Asynchronous Box Calculus*, *ABC*) qui propose une algèbre de réseaux places/transitions, une syntaxe pour cette

algèbre avec une sémantique opérationnelle, cohérente avec la sémantique réseau.

Ce chapitre présente donc une étude exhaustive du modèle ABC et de ses propriétés. Il commence par une introduction informelle permettant de fixer l'intuition. Viennent ensuite les définitions de base puis celle de l'algèbre de réseaux. Puis nous donnons les moyens pour relier les comportements des réseaux à leurs structures. La section suivante introduit la syntaxe, sa sémantique réseau et sa sémantique opérationnelle, pour se terminer sur les résultats de cohérence entre les sémantiques. Dans la section suivante, nous donnons plusieurs généralisations sur lesquelles la plupart des preuves sont basées.

8.1 L'algèbre des boxes asynchrones

Le modèle de l'algèbre de réseaux Petri (*Petri Net Algebra*, PNA) qui nous sert de base dans ce chapitre propose les opérateurs suivants qui sont similaires aux opérateurs M-nets que nous avons utilisés jusque là : le choix $E_1 \,\square\, E_2$, la séquence $E_1; E_2$, la composition parallèle $E_1 \| E_2$, l'itération $E_1 \circledast E_2$ (il s'agit d'une itération binaire sans initialisation) et le scoping $E \,\mathsf{sc}\, a$ (dans ce contexte, les actions n'ont pas de paramètre). Considérons par exemple les trois expressions ci-dessous qui modélisent une section critique et deux processus pouvant y faire appel.

$$\begin{array}{lll} \textsc{CritSect} & \stackrel{\text{df}}{=} & ((a_1; r_1) \,\square (a_2; r_2)) \circledast f \quad , \\ \textsc{User}_1 & \stackrel{\text{df}}{=} & \widehat{a_1}; \widehat{r_1} \quad , \\ \textsc{User}_2 & \stackrel{\text{df}}{=} & \widehat{a_2}; \widehat{r_2} \quad . \end{array}$$

Les actions a_1 et a_2 (ainsi que leurs conjuguées $\widehat{a_1}$ et $\widehat{a_2}$) représentent l'accès à une ressource partagée alors que les actions r_1 et r_2 (ainsi que $\widehat{r_1}$ et $\widehat{r_2}$) symbolisent sa libération. L'action f correspond à une action finale. Nous pouvons composer les trois expressions pour qu'elles s'exécutent en parallèle :

$$\textsc{PreMutex} \stackrel{\text{df}}{=} \textsc{User}_1 \| \textsc{CritSect} \| \textsc{User}_2 \quad .$$

Le réseau de Petri correspondant à cette expression est appelé un *box* et il est présenté à gauche de la figure 8.1. Comme les M-nets, les places des boxes ont des statuts qui sont représentés graphiquement de la même façon. Par ailleurs, nous indiquons à l'intérieur des transitions leurs étiquettes.

Ainsi définie, l'expression PreMutex et le box correspondant spécifient correctement les trois composants qui les constituent mais ils ne leur permettent pas de communiquer. Cela peut-être fait simplement au moyen du scoping :

$$\textsc{Mutex} \stackrel{\text{df}}{=} \textsc{PreMutex} \,\mathsf{sc}\, \{a_1, a_2, r_1, r_2\} \quad ,$$

dont le box correspondant est donné à droite de la figure 8.1.

La sémantique opérationnelle des expressions ABC est donnée par des *règles SOS* (*Structured Operational Semantics*) dans le style de Plotkin [Plo81]. Cependant, au lieu d'exprimer les évolutions en modifiant la structure des expressions, comme $a.E \xrightarrow{a} E$

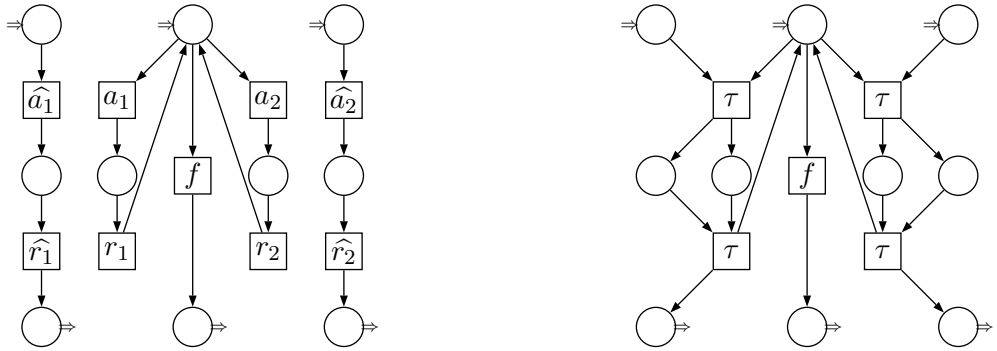


Figure 8.1 — À gauche, le box de PreMutex et à droite celui de Mutex.

en CCS [Mil80, Mil83, Mil89], l'idée ici est de représenter l'état courant de l'évolution au moyen de barres placées au-dessus et au-dessous des sous-expressions. Ces barres correspondent au marquage initial et final, respectivement, des boxes associés. Ceci est illustré figure 8.2 où le réseau de gauche correspond à PreMutex dans lequel les deux processus utilisateurs ont terminé et la section critique est toujours dans son état initial ; le box de droite correspond à l'état initial de Mutex.

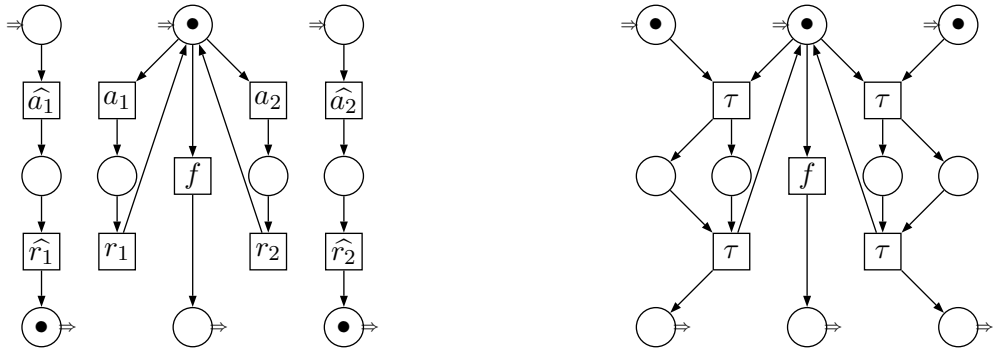


Figure 8.2 — À gauche, le box de $\underline{\text{User}_1}\|\overline{\text{CritSect}}\|\underline{\text{User}_2}$ et à droite celui de $\overline{\text{Mutex}}$.

Nous distinguons deux types de règles SOS. Les premières sont des équivalences permettant d'exprimer que deux expressions représentent le même état. Par exemple :

$$\overline{\text{User}_1\|\text{CritSect}\|\text{User}_2} \equiv \overline{\text{User}_1}\|\overline{\text{CritSect}\|\text{User}_2} \equiv \overline{\text{User}_1}\|\overline{\text{CritSect}}\|\overline{\text{User}_2} \quad ,$$

Les secondes spécifient des évolutions qui correspondent à un changement d'état du système. Au moyen de ces règles, nous pouvons par exemple dériver :

$$((a_1;r_1)\,\square(a_2;r_2)) \circledast \overline{f} \xrightarrow{\{f\}} ((a_1;r_1)\,\square(a_2;r_2)) \circledast \underline{f} \quad .$$

Les deux algèbres composant PNA (celle des expressions et celle des boxes) sont complètement cohérentes : elles génèrent des systèmes de transitions isomorphes [6, Bes$^+$01b].

Communications asynchrones. Nous pouvons étendre PNA au moyens de liens asynchrones en lui ajoutant de nouvelles actions de base, capables d'envoyer, de recevoir ou de tester la présence d'un jeton dans une place de liens. Par exemple, nous pouvons considérer les trois processus suivants, qui modélisent respectivement un producteur, un consommateur et un testeur, chacun étant capable de réaliser exactement une action avant de terminer :

$$\begin{aligned} \textsc{ProdOne} &\stackrel{\text{df}}{=} pb^{+} \quad , \\ \textsc{ConsOne} &\stackrel{\text{df}}{=} cb^{-} \quad , \\ \textsc{TestOne} &\stackrel{\text{df}}{=} tb^{\pm} \quad . \end{aligned}$$

Ici, pb^{+} est une action dont le rôle est de produire un jeton dans une place de liens de statut b tout en générant une action visible p. De manière similaire, cb^{-} consomme une jeton dans une telle place et génère l'action c. Finalement, $tb^{\pm}$ combine ces deux possibilités et teste donc la présence d'un jeton en générant l'action t. Les réseaux de Petri correspondant à ces trois processus sont donnés figure 8.3.

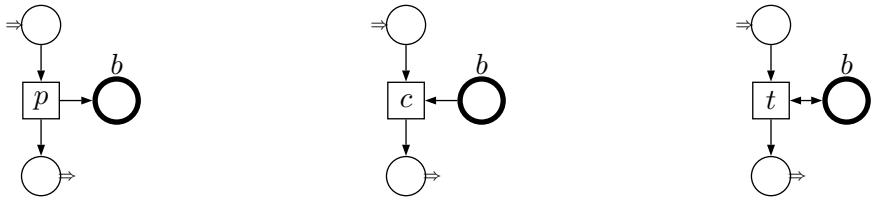


Figure 8.3 — De gauche à droite, les réseaux correspondant à ProdOne, ConsOne et TestOne.

Comme dans le cas des M-nets, l'application d'un des opérateurs de contrôle de flot entraîne la fusion des places de liens ayant le même statut. Nous pouvons par exemple définir un système $\textsc{Syst} \stackrel{\text{df}}{=} \textsc{ProdOne} \| \textsc{ConsOne}$ dont le box associé est donné à gauche de la figure 8.4. Nous pouvons aussi construire $\textsc{TestSyst} \stackrel{\text{df}}{=} \textsc{Syst} \| \textsc{Test}$, son box associé étant donné au milieu de la figure 8.4. Dans ces deux exemples, nous constatons que les communications entre les processus sont possibles grâces à la fusion des places de liens. La restriction de liens existe là encore et fonctionne comme dans les M-nets. Le réseau $\textsc{Syst}\,\mathsf{tie}\,b$ dans lequel la place de statut b a reçu le statut b est donné à droite de la figure 8.4.

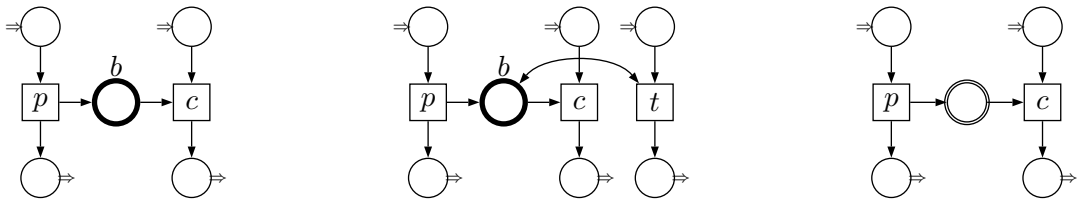


Figure 8.4 — À gauche, le réseau correspondant à Syst et au milieu celui pour TestSyst. Le réseau de droite correspond à $\textsc{Syst}\,\mathsf{tie}\,b$.

Extension des algèbres. Le modèle PNA enrichi des liens asynchrones, avec les trois nouveaux types d'actions de base (comme pb^+, cb^- et $tb^\pm$) et la restriction de liens, forme *l'algèbre des boxes asynchrones* (*Asynchronous Box Calculus*, *ABC*). Ce nouveau modèle est donc composé d'une algèbre de réseaux (les boxes) et d'une algèbre d'expressions dotée d'une sémantique dénotationnelle (en terme de boxes) et d'une sémantique opérationnelle (en terme de règles SOS). Nous montrerons que ces deux sémantiques sont complètement cohérentes puisqu'elles génèrent des systèmes de transitions isomorphes (exactement comme dans le cas de PNA).

En introduisant les places de liens, qui peuvent ne pas être bornées, nous perdons la propriété de PNA qui garantissait que les réseaux étaient toujours 1-bornés. Nous décidons donc de lever cette restriction, non seulement pour les places de liens, mais aussi pour les places de contrôle. Cela permet d'éviter des contraintes introduites dans PNA à cet effet (par exemple, on ne pouvait pas utiliser une composition parallèle comme premier argument d'une itération). Nous montrons que les réseaux dans ABC n'autorisent pas l'auto-concurrence de transitions, qui est une limitation bien moins contraignante.

Exemples. Dans la suite de ce chapitre, nous nous appuyons sur trois exemples basés sur l'itération. Nous définissons d'abord les processus suivants, modélisant respectivement un producteur, un consommateur et un testeur, capables de répéter leurs actions :

$$\begin{aligned}
\textsc{Prod} &\stackrel{\text{df}}{=} \textsc{ProdOne} \circledast f = pb^+ \circledast f \quad , \\
\textsc{Cons} &\stackrel{\text{df}}{=} \textsc{ConsOne} \circledast f = cb^- \circledast f \quad , \\
\textsc{Test} &\stackrel{\text{df}}{=} \textsc{TestOne} \circledast f = tb^\pm \circledast f \quad .
\end{aligned}$$

Le premier exemple illustre simplement le fonctionnement des liens asynchrones. Il est composé de deux producteurs et d'un consommateur :

$$\textsc{Syst}_1 \stackrel{\text{df}}{=} s; ((\textsc{Prod} \| \textsc{Prod} \| \textsc{Cons}) \,\mathsf{tie}\, b) \quad .$$

Le box N_{S1}, correspondant à l'état initial de ce système, est montré en haut à gauche de la figure 8.5 ; nous avons indiqué à côté de chaque transition son nom. Pour ce système, nous considérons le scénario d'évolution suivant :

- le système est démarré avec l'exécution de l'action s ;
- chaque producteur envoie un jeton dans la place de lien ;
- le consommateur en retire un jeton pendant que le premier producteur en crée un nouveau ;
- les trois processus terminent en exécutant leurs actions f.

Ce scénario correspond à la séquence de steps

$$N_{S1}[\{t_1\}\{t_2, t_3\}\{t_2, t_4\}\{t_5, t_6, t_7\}\rangle N'_{S1} \quad ,$$

où N'_{S1} est N_{S1} avec deux jetons dans la place de liens, un dans chaque place de sortie et pas de jetons ailleurs. Ce réseau est donc dans un marquage final. Nous pouvons exprimer la même exécution sur les actions visibles :

$$N_{S1}[\{s\}\{p, p\}\{p, c\}\{f, f, f\}\rangle N'_{S1} \quad .$$

Les steps sont cette fois des multi-ensembles d'actions et ne permettent pas de distinguer exactement l'origine d'une action (le p dans le troisième step pourrait venir de n'importe quel producteur). En général, c'est ce type d'exécution qui est intéressant en dernier lieu. Pourtant, pour les besoins des preuves, la plus grande partie des définitions s'appuiera sur les noms des transitions.

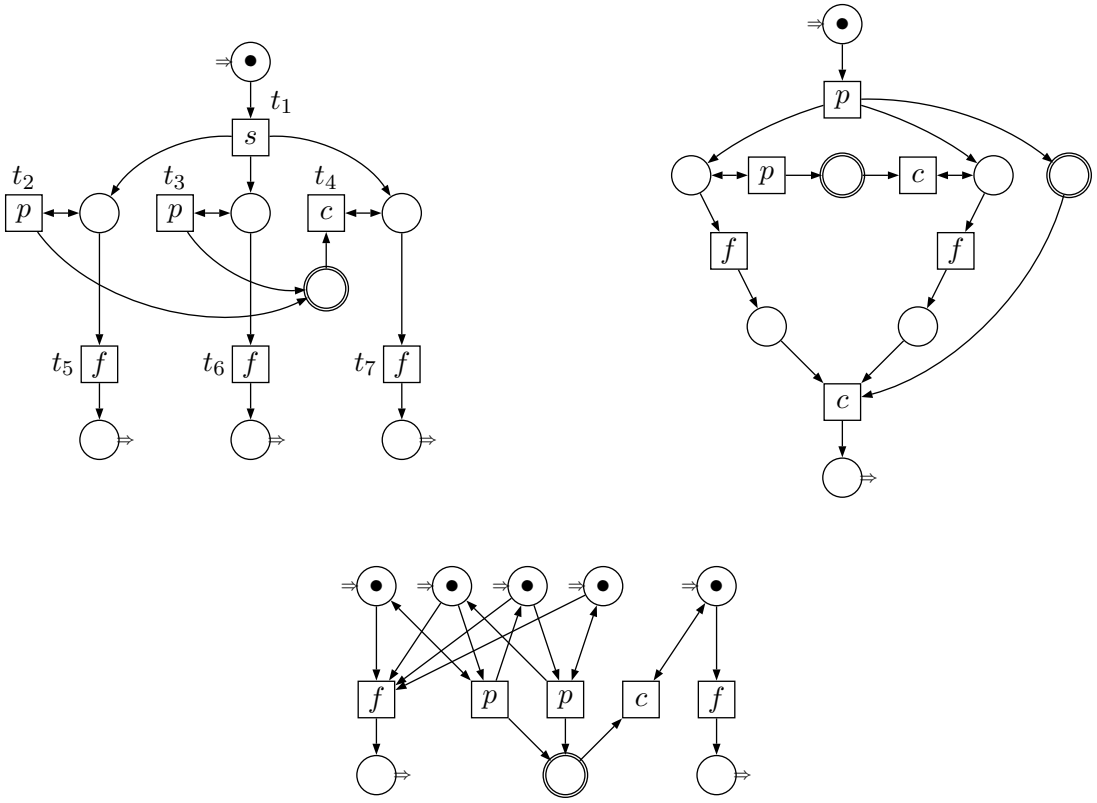


Figure 8.5 — En haut à gauche, N_{S1}, le box de $\overline{\text{SYST}_1}$ avec les noms de ses transitions. À sa droite, N_{S2}, le box de $\overline{\text{SYST}_2}$ et en bas, N_{S3}, celui de $\overline{\text{SYST}_3}$.

Le second exemple permet d'illustrer l'encapsulation par la restriction de liens :

$$\text{SYST}_2 \stackrel{\text{df}}{=} (\text{PRODONE}; ((\text{PROD} \| \text{CONS})\,\mathsf{tie}\, b); \text{CONSONE})\,\mathsf{tie}\, b \quad .$$

Le box N_{S2}, qui correspond à ce système dans son état initial, est donné en haut à droite de la figure 8.5. Pour cet exemple, nous considérons l'évolution suivante :

- le système démarre avec le franchissement de la transition étiqueté p (tout en haut) et produit un jeton sur la place de liens externe (à droite) ;
- le producteur interne envoie deux jetons dans la place de liens interne (au milieu) ;
- le consommateur interne retire un jeton de la place de liens interne ;
- le producteur et le consommateur internes se terminent simultanément avec leur actions f ;

- le système complet termine en franchissant la transition étiquetée c (en bas) qui retire le jeton de la place de liens externe.

Cette exécution correspond à la séquence de steps suivante :

$$N_{S2}[\{p\}\{p\}\{p\}\{c\}\{f,f\}\{c\}\rangle N'_{S2} \quad ,$$

où N'_{S2} est comme N_{S2} sauf qu'il est marqué d'un jeton dans la place de sortie et d'un dans la place de liens interne (sans aucun jeton ailleurs). Nous observons sur cet exemple que les communications asynchrones sont correctement restreintes aux parties du système situées sous chaque application de $\mathsf{tie}\, b$.

Le troisième exemple illustre comment obtenir des places de contrôle non 1-bornées, sans qu'il soit toutefois possible d'avoir l'auto-concurrence. Il s'agit du système :

$$\textsc{Syst}_3 \stackrel{\text{df}}{=} (((\textsc{ProdOne} \| \textsc{ProdOne}) \circledast f) \| \textsc{Cons})\, \mathsf{tie}\, b \quad .$$

Le box N_{S3}, qui correspond à l'état initial de système est donné au bas de la figure 8.5 (la façon dont il est construit dépend de la définition donnée section 8.5.1). Par rapport à $\textsc{Syst}_1$, ce système produit les jetons deux par deux puisque l'itération porte sur la composition parallèle des deux producteurs simples. Nous pouvons constater sur la figure que le franchissement de l'une des transitions étiquetée p amène tout de suite deux jetons dans l'une des places d'entrée. L'autre transition étiquetée p est alors franchissable, mais sans auto-concurrence car l'une de ses places d'entrée n'a qu'un jeton. Elle doit être franchie, en rétablissant le marquage de départ au niveau des places d'entrée, pour que la transition de gauche, étiquetée f, puisse être franchie à son tour.

Nous pouvons aussi constater sur cet exemple que des arcs arrivent sur des places d'entrée, ce qui était interdit avec les M-nets (qui doivent être ex-dirigés). C'est une autre restriction que ABC (et déjà PNA) permet de supprimer par rapport aux M-nets.

Pour ce dernier exemple, nous considérons l'exécution suivante :

- le système démarre en exécutant la transition étiquetée p la plus à gauche ;
- le consommateur supprime de la place de liens le jeton qui vient d'y être produit ; simultanément, l'autre producteur s'exécute ;
- le système termine en franchissant les deux transitions étiquetées f.

Cela correspond à la séquence de steps

$$N_{S3}[\{p\}\{p,c\}\{f,f\}\rangle N'_{S3} \quad ,$$

où N'_{S3} est comme N_{S3} avec un jeton dans chaque place de sortie et dans la place de liens et aucun dans les places d'entrée.

8.2 Définitions et notations

Nous adoptons globalement les mêmes notations que pour les M-nets et nous rappelons ci-dessous celles qui nous seront utiles ; nous donnons aussi les simplifications nous permettant de nous restreindre à des réseaux places/transitions.

L'arité de tout symbole d'action $a \in \mathbb{A}$ est nulle puisque les actions ne peuvent pas avoir de paramètre. Nous définissons aussi l'ensemble $\mathbb{A}_\tau \stackrel{\text{df}}{=} \mathbb{A} \uplus \{\tau\}$ qui contient toutes les actions de $\mathbb{A}$ plus une action silencieuse, τ, qui ne peut pas servir à la synchronisation (puisqu'elle n'a pas de conjuguée).

L'ensemble des symboles de liens $\mathbb{B}$ est conservé à l'identique, de même que l'ensemble $\mathbb{S}$ des statuts de places. L'ensemble des valeurs *Val* est réduit au singleton $\{\bullet\}$.

Afin d'exprimer le scoping de transitions, nous utilisons la *fonction d'interface* $\varphi_{\mathsf{sc}\,a}$, il s'agit d'une fonction partielle de $\mathsf{mult}(\mathbb{A}_\tau) \setminus \{\varnothing\}$ dans $\mathbb{A}_\tau$; son domaine est $\{\{x\} \mid x \in \mathbb{A}_\tau \setminus \{a, \widehat{a}\}\} \uplus \{a, \widehat{a}\}$ et nous avons $\varphi_{\mathsf{sc}\,a}(\{x\}) \stackrel{\text{df}}{=} x$ pour tout $x \notin \{a, \widehat{a}\}$ et $\varphi_{\mathsf{sc}\,a}(\{a, \widehat{a}\}) \stackrel{\text{df}}{=} \tau$. Dans la section 8.6, nous verrons que d'autres fonctions d'interfaces plus générales peuvent être envisagées afin d'exprimer toutes sortes de communications synchrones. Avant cela, nous utiliserons aussi φ_{id} dont le domaine est $\{\{a\} \mid a \in \mathbb{A}_\tau\}$ et telle que $\varphi_{id}(\{a\}) \stackrel{\text{df}}{=} a$ pour tout $a \in \mathbb{A}_\tau$.

Réseaux de Petri étiquetés. La classe de réseaux considérée ici est plus simple que les réseaux colorés présentés au chapitre 3 ; en effet, puisque $\mathit{Val} = \{\bullet\}$, nous n'avons pas besoin des gardes ou des types de places. De plus, le arcs n'ont besoin d'être étiquetés que par un entier, leur *poids*, correspondant au nombre de jetons qu'ils transportent.

Un *réseau étiqueté* (marqué) est donc $N \stackrel{\text{df}}{=} (S, T, W, \lambda, M)$ où S est l'ensemble des places, T celui des transitions, W est la fonction de poids associant à tout arc de $(S \times T) \cup (T \times S)$ un entier positif ou nul (dans ce cas, l'arc n'est pas dessiné), λ est la fonction d'étiquetage associant à chaque place son statut (un élément de $\mathbb{S} \uplus \mathbb{B}$) et à chaque transition une action de $\mathbb{A}_\tau$ ou une fonction d'interface φ, M est le marquage de N qui sera vu comme un élément de $\mathsf{mult}(S)$. Étant donné un réseau N, sauf indication contraire, il sera implicitement considéré que $N = (S_N, T_N, W_N, \lambda_N, M_N)$.

Nous adoptons comme pour les M-nets les notations de marquages restreints, par exemple, M_N^{c} désigne le marquage M du réseau N restreint aux places de contrôle de N.

Un marquage M de N est dit :

- *sain* si $M^{\mathsf{c}} \geq N^{\mathsf{e}}$ implique que $M^{\mathsf{c}} = N^{\mathsf{e}}$ et si $M^{\mathsf{c}} \geq N^{\mathsf{x}}$ implique que $M^{\mathsf{c}} = N^{\mathsf{x}}$;
- *ac-libre* si, pour toute transition t, il existe une place de contrôle $s \in {}^\bullet t \cap N^{\mathsf{c}}$ telle que $M(s) < 2\,W(s,t)$, ce qui implique que ce marquage n'autorise pas l'*auto-concurrence* ;
- *quasi-sûr* si, pour toute transition t, il existe une place de contrôle $s \in {}^\bullet t \cap N^{\mathsf{c}}$ telle que $M(s) \leq 1$. Remarquons qu'alors M est ac-libre.

Un réseau étiqueté est *simple* si sa fonction de poids est bornée par la fonction constante égale à 1.

Lorsqu'une transition est franchissable, elle n'a qu'une seule valuation, nous pouvons donc l'omettre des steps et séquences de steps. Pour un réseau N, nous notons $[N\rangle$ l'ensemble des réseaux N' tels qu'il existe une séquence de steps $U_1 \cdots U_k$ telle que $N[U_1 \cdots U_k\rangle N'$. Nous notons aussi $\mathsf{enabled}(N)$ l'ensemble des steps franchissables de N.

Dans la suite, nous considérons aussi des steps Γ composés d'actions qui correspondent aux étiquettes des transitions impliquées dans les steps ; nous notons alors $N[\Gamma\rangle_\lambda N'$ le

franchissement d'un tel step (qui est un véritable multi-ensemble d'actions, et non un simple ensemble comme dans le cas des steps de transitions).

Systèmes de transitions. Le comportement d'un réseau étiqueté peut être entièrement représenté par son *système de transitions*. Comme nous avons deux sortes de steps, nous aurons deux types de systèmes de transitions.

Le *système de transitions complet* d'un réseau $N = (S, T, W, \lambda, M)$ est $\mathsf{fts}(N) \stackrel{\text{df}}{=} (V, L, A, \mathit{init})$ où $V \stackrel{\text{df}}{=} [N\rangle$ est l'ensemble des états ; $L \stackrel{\text{df}}{=} 2^T$ est l'ensemble des étiquettes d'arcs ; $A \stackrel{\text{df}}{=} \{(N, U, N') \in V \times L \times V \mid N[U\rangle N'\}$ est l'ensemble des arcs ; $\mathit{init} \stackrel{\text{df}}{=} N$ est l'état initial.

Le *système de transitions étiqueté* d'un réseau N, noté $\mathsf{lts}(N)$, est obtenu de $\mathsf{fts}(N)$ en changeant l'étiquette U de chaque arc en $\lambda_N(U)$, qui est l'ensemble d'actions $\{\lambda_N(t) \in \mathbb{A} \mid t \in U\}$. (Cela peut conduire à fusionner des arcs.)

Remarquons que nous définissons des graphes dont les nœuds dont des *boxes* et non simplement des *marquages*. Cela permet de simplifier la formulation des théorèmes 26 et 28 qui établissent la cohérence des systèmes de transitions générés par les expressions ABC et les boxes qui leur correspondent.

Un *isomorphisme* entre deux systèmes de transitions (V, L, A, init) et $(V', L', A', \mathit{init}')$ est une bijection $\mathsf{iso} : V \to V'$ telle que $\mathsf{iso}(\mathit{init}) = \mathit{init}'$ et, pour tout $(v, l, v') \in V \times (L \cup L') \times V$, $(v, l, v') \in A$ si et seulement si $(\mathsf{iso}(v), l, \mathsf{iso}(v')) \in A'$.

8.3 Une algèbre de boxes

Un *box asynchrone* (ou simplement un *box*) est un réseau de Petri étiqueté N tel que :

- chaque transition est étiquetée par une action de $\mathbb{A}_\tau$;
- N est ex-restreint : il possède au moins une place d'entrée et une place de sortie ;
- N est B-restreint : pour chaque symbole de liens $b \in \mathbb{B}$, N possède une et une seule place de statut b ;
- N est T-restreint : pour chaque transition t, ${}^\bullet t \cap N^{\mathsf{c}} \neq \varnothing \neq t^\bullet \cap N^{\mathsf{c}}$.

Comme nous l'avons précisé dans l'introduction de ce chapitre, l'ex-orientation n'est pas requise pour les boxes.

Un box N est dit *statique* si $M_N^{\mathsf{c}} = \varnothing$ et si tous les marquages accessibles à partir de N^{e} ou N^{x} dans N privé de ses places de liens (et des arcs adjacents) sont sains et ac-libres. Il est dit *dynamique* si $M_N^{\mathsf{c}} \neq \varnothing$ et si tous les marquages accessibles à partir de M_N^{c} dans N privé de ses places de liens (et des arcs adjacents) sont sains et ac-libres.

La figure 8.6 donne quatre types de boxes servant de cas de base à la construction de tous les autres.

Le résultat suivant exprime en particulier qu'un box dynamique le reste après évolution.

Proposition 9.
Soit N un box et $N[U\rangle N'$.

1. *Si N est statique, alors $U = \varnothing$ et $N = N'$.*

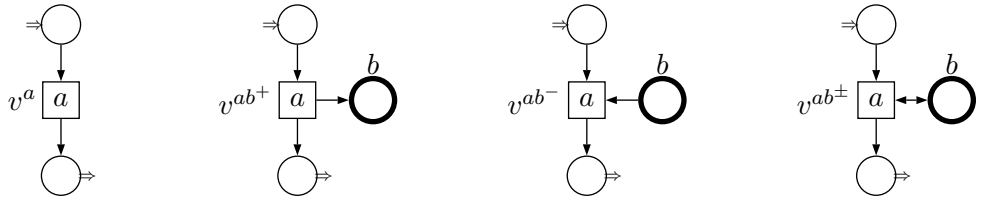


Figure 8.6 — De gauche à droite, les boxes N_a, N_{ab^+}, N_{ab^-} et $N_{ab^\pm}$, pour $a \in \mathbb{A}_\tau$ et $b \in \mathbb{B}$.

2. *Si N est dynamique alors U est un ensemble de transitions et N' est un box dynamique.*

Preuve. Par définition. □

Nous définissons maintenant des *opérateurs de marquages* qui modifient le marquage d'un box. Étant donnés $b \in \mathbb{B}$ et $B \in \mathsf{mult}(\mathbb{B})$:

- $N.B$ ajoute $B(b)$ jetons à la place de N de statut b, pour chaque $b \in \mathbb{B}$. Nous notons aussi $N.b \stackrel{\text{df}}{=} N.\{b\}$.
- $\overline{N}$ ajoute un jeton à chaque place d'entrée de N, autrement dit, $M_{\overline{N}} = M_N + N^{\mathsf{e}}$.
- $\underline{N}$ ajoute un jeton à chaque place de sortie de N, autrement dit, $M_{\underline{N}} = M_N + N^{\mathsf{x}}$.
- $\lfloor N \rfloor$ est N privé des jetons dans ses places de contrôle : $M_{\lfloor N \rfloor} = M_N - M_N^{\mathsf{c}}$.
- $\llbracket N \rrbracket$ est N privé de tous ses jetons : $M_{\llbracket N \rrbracket} = M_\varnothing(N)$.

Les deux derniers opérateurs s'étendent naturellement à des n-uplets de boxes. Remarquons que, par sa définition, l'opération $.B$ n'affecte que les places de liens ouvertes.

Proposition 10.
Soient N un box et B et B' des éléments de $\mathsf{mult}(\mathbb{B})$.

1. *N est statique ssi $N.B$ est statique.*
2. *N est dynamique ssi $N.B$ est dynamique.*
3. *$\overline{N}$ est dynamique ssi N est statique ssi $\underline{N}$ est dynamique.*
4. *$N.\varnothing = N$, $N.B.B' = N.(B + B')$, $\overline{N}.B = \overline{N.B}$ et $\underline{N}.B = \underline{N.B}$.*
5. *Si N est statique ou dynamique alors $\lfloor N \rfloor$ et $\llbracket N \rrbracket$ sont statiques.*
6. *Si N est statique alors $\lfloor N \rfloor = N$.*
7. *$\llbracket \llbracket N \rrbracket \rrbracket = \lfloor \llbracket N \rrbracket \rfloor = \llbracket \lfloor N \rfloor \rrbracket = \llbracket N \rrbracket$.*
8. *$\lfloor N \rfloor.B = \lfloor N.B \rfloor$, $\lfloor \overline{N} \rfloor = \lfloor \underline{N} \rfloor = \lfloor \lfloor N \rfloor \rfloor = \lfloor N \rfloor$.*
9. *$\llbracket N.B \rrbracket = \llbracket \overline{N} \rrbracket = \llbracket \underline{N} \rrbracket = \llbracket N \rrbracket$.*

Preuve. Par définition. □

8.3.1 Boxes opérateurs

Les *boxes opérateurs* permettent de synthétiser les différents opérateurs par substitution de transitions (décrite à la section suivante). Un box opérateur Ω est un réseau étiqueté fini, simple, ex-restreint et T-restreint dont chacune des transitions est étiquetée par une fonction d'interface et qui ne possède que des places de contrôle (et n'est donc pas B-restreint). Nous supposons que les transitions v_1, …, v_n d'un box opérateur Ω sont implicitement ordonnées et alors, tout n-uplet de boxes $\mathbf{N} = (N_1, \ldots, N_n)$ est appelé un Ω-uplet ; nous notons aussi N_{v_i} le réseau N_i pour $i \leq n$. La substitution des transitions d'un box opérateur Ω par les boxes d'un Ω-uplet $\mathbf{N}$ est notée $\Omega(\mathbf{N})$, il s'agit ici de substituer chaque transition v_i de Ω par le N_{v_i} correspondant dans le Ω-uplet.

Nous distinguons quatre groupes de boxes opérateurs qui sont décrits ci-dessous et donnés sur la figure 8.7. Pour cette description, nous notons abox l'ensemble des boxes asynchrones, abox_s l'ensemble des boxes asynchrones statiques et abox_d l'ensemble des boxes asynchrones dynamiques.

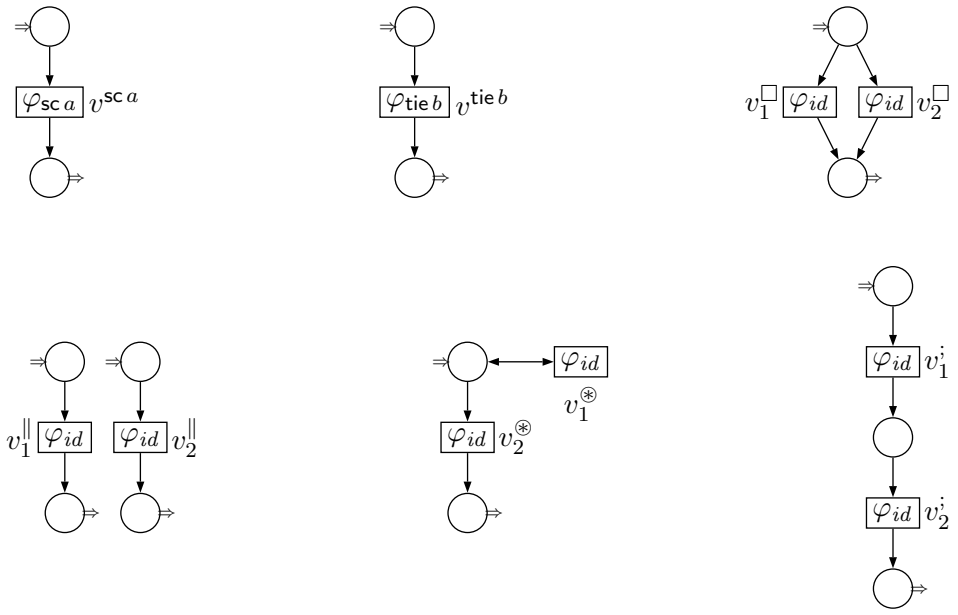


Figure 8.7 — Les boxes opérateurs de ABC : de gauche à droite et de bas en haut, $\Omega_{\mathsf{sc}\,a}$, $\Omega_{\mathsf{tie}\,b}$, $\Omega_{\square}$, $\Omega_{\|}$, $\Omega_{\circledast}$ et $\Omega_{;}$, avec $a \in \mathbb{A}_\tau$ et $b \in \mathbb{B}$.

Choix $\Omega_{\square}$, itération $\Omega_{\circledast}$ et séquence $\Omega_{;}$. Les trois premiers opérateurs permettent différentes composition de contrôle de flot. Ils sont binaires et ont tous le domaine $\mathsf{dom}(\Omega_{\square}) = \mathsf{dom}(\Omega_{\circledast}) = \mathsf{dom}(\Omega_{;}) \stackrel{\text{df}}{=} (\mathsf{abox}_s \times \mathsf{abox}_s) \cup (\mathsf{abox}_s \times \mathsf{abox}_d) \cup (\mathsf{abox}_d \times \mathsf{abox}_s)$, et donc, au plus l'un des arguments peut être dynamique.

Pour une paire de boxes $\mathbf{N} = (N_1, N_2)$, nous notons :

$$N_1 \,\square\, N_2 \stackrel{\text{df}}{=} \Omega_{\square}(\mathbf{N}) \quad , \qquad N_1 \circledast N_2 \stackrel{\text{df}}{=} \Omega_{\circledast}(\mathbf{N}) \quad , \qquad N_1 ; N_2 \stackrel{\text{df}}{=} \Omega_{;}(\mathbf{N}) \quad .$$

Composition parallèle $\Omega_{\|}$. Il s'agit aussi d'un opérateur binaire mais son domaine d'application est $\mathsf{dom}(\Omega_{\|}) \stackrel{\text{df}}{=} (\mathsf{abox}_s \times \mathsf{abox}_s) \cup (\mathsf{abox}_d \times \mathsf{abox}_d)$, de cette façon, ses deux arguments peuvent évoluer en concurrence. (Remarquons que $\mathsf{abox}_s \times \mathsf{abox}_d$ et $\mathsf{abox}_d \times \mathsf{abox}_s$ sont exclus du domaine de manière à conserver la cohérence : le résultat d'une composition parallèle sera dynamique si et seulement si *les deux* boxes composés le sont.)

Pour un $\Omega_{\|}$-uplet $\mathbf{N} = (N_1, N_2)$, nous notons :

$$N_1 \| N_2 \stackrel{\text{df}}{=} \Omega_{\|}(\mathbf{N}) \quad .$$

Scoping $\Omega_{\mathsf{sc}\,a}$. Cet opérateur est paramétré par une action $a \in \mathbb{A}$, il est unaire et son domaine d'application est $\mathsf{dom}(\Omega_{\mathsf{sc}\,a}) \stackrel{\text{df}}{=} \mathsf{abox}$. Le changement des interfaces de communication est capturé par la fonction d'interface $\varphi_{\mathsf{sc}\,a}$, déjà définie, qui force la synchronisation des paires d'actions $(a, \widehat{a})$.

Pour un box N, et une action $a \in \mathbb{A}$, nous notons :

$$N\,\mathsf{sc}\,a \stackrel{\text{df}}{=} \Omega_{\mathsf{sc}\,a}(N) \quad .$$

Restriction de liens $\Omega_{\mathsf{tie}\,b}$. Ce dernier opérateur est paramétré par un symbole de liens $b \in \mathbb{B}$, il est unaire et son domaine d'application est $\mathsf{dom}(\Omega_{\mathsf{tie}\,b}) \stackrel{\text{df}}{=} \mathsf{abox}$. Son rôle est de « cacher » la place de statut b, pour cela, il lui donne le statut b et ajoute une nouvelle place d'étiquette b.

Pour un box N, et un symbole de liens $b \in \mathbb{B}$, nous notons :

$$N\,\mathsf{tie}\,b \stackrel{\text{df}}{=} \Omega_{\mathsf{tie}\,b}(N) \quad .$$

8.3.2 Substitution de transitions

À partir de maintenant et dans la suite de ce chapitre, l'identité des transitions (leurs noms) dans les boxes asynchrones va jouer un rôle essentiel. Pour assurer l'identification des transitions, leurs noms seront des arbres finis étiquetés, permettant de retracer les opérateurs impliqués dans la construction d'un box. Ces arbres sont commutatifs : l'ordre des fils d'un nœud n'a pas d'importance.

Considérons un ensemble η contenant des identités de transitions de base. Il contient en particulier les noms indiqués sur les figures 8.6 et 8.7. Un nom $v \in \eta$ est considéré comme un arbre ayant v pour seul nœud (qui est alors sa racine et sa seule feuille). Afin d'exprimer des noms plus complexes sous la forme d'arbres commutatifs, nous utilisons la notation suivante (voir aussi la figure 8.8 pour une illustration) :

- $v \triangleleft \mathsf{T}$ représente l'arbre composé de la racine v sur laquelle sont greffés les arbres de T, où $v \in \eta$ est un nom de transition de base et T est un ensemble fini d'arbres d'identités de transitions.
- $v \triangleleft \mathsf{t}$ représente $x \triangleleft \{t\}$.
- $v \lll \mathsf{T}$ est l'ensemble d'arbres $\{v \triangleleft \mathsf{t} \mid \mathsf{t} \in \mathsf{T}\}$.

Une politique de nommage similaire pourrait être utilisée pour les identités des places, cependant elle n'est utile que lorsqu'on considère l'opérateur de récursion, ce qui n'est pas le cas ici. Nous pouvons donc utiliser un schéma plus simple en supposant que les identités des places peuvent être changées autant que nécessaire pour éviter des conflits de noms. En particulier, pour appliquer la substitution de transitions, nous supposons que les ensembles de places des opérandes sont disjoints. Lorsque ce n'est pas le cas, nous renommons les places de manière cohérente. Sous cette hypothèse, nous pouvons construire de nouveaux noms de places en groupant des noms de places existantes. Par exemple, si s_1 et s_2 sont deux places alors (s_1, s_2) peut être utilisé comme identité d'une nouvelle place.

La substitution de transitions est illustrée figure 8.8 où nous donnons les différentes étapes de la construction de $((N_{ab^+} \| N_{\widehat{a}b^-}) \,\mathsf{sc}\, a) \,\square\, N_f$, pour deux actions a et f et un symbole de liens b. Nous y indiquons aussi les noms des transitions construites avec les règles donnée ci-dessous.

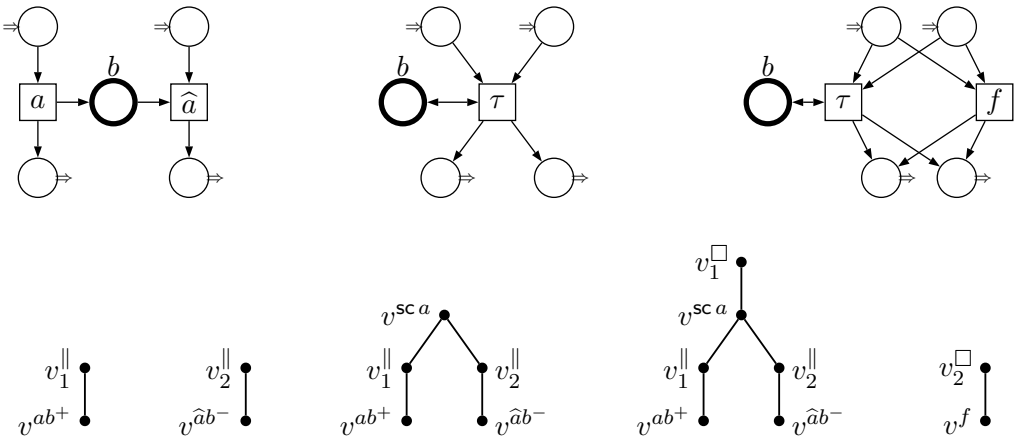


Figure 8.8 — La ligne du haut présente de gauche à droite les boxes $N \stackrel{\text{df}}{=} N_{ab^+} \| N_{\widehat{a}b^-}$, $N \,\mathsf{sc}\, a$ et $(N \,\mathsf{sc}\, a) \,\square\, N_f$, où a et f sont des actions de $\mathbb{A}$ et b est un symbole de liens dans $\mathbb{B}$. La ligne du bas donne les identités des transitions de ces boxes, dans l'ordre où elles sont dessinées. Le quatrième arbre peut s'écrire $v_1^{\square} \triangleleft v^{\mathsf{sc}\, a} \triangleleft \{v_1^{\|} \triangleleft v^{ab^+}, v_2^{\|} \triangleleft v^{\widehat{a}b^-}\}$.

Opérateurs binaires. Soit $\Omega_{\mathsf{bin}} \in \{\Omega_{\square}, \Omega_{\circledast}, \Omega_{;}, \Omega_{\|}\}$ un opérateur binaire de ABC et $\mathbf{N} = (N_1, N_2) = (N_{v_1^{\mathsf{bin}}}, N_{v_2^{\mathsf{bin}}})$ une paire de boxes. Le résultat de la substitution des transitions v_1^{bin} et v_2^{bin} de Ω_{bin} par les boxes N_1 et N_2 est le réseau étiqueté $\Omega_{\mathsf{bin}}(\mathbf{N}) = N$ dont les composants sont définis de la façon suivante.

L'ensemble des transitions de N est l'ensemble des arbres $v_i^{\mathsf{bin}} \triangleleft t$ pour $t \in T_{N_i}$ et $i \in \{1, 2\}$. L'étiquette de chaque transition $v_i^{\mathsf{bin}} \triangleleft t$ est celle de t.

Chaque place $s \in S_{N_1} \cup S_{N_2}$ de statut i ou b est aussi présente dans S_N. Son étiquette et son marquage sont inchangés, et, pour toute transitions $w \triangleleft t$, la fonction de poids est définie par :

$$W_N(s, w \triangleleft t) \stackrel{\text{df}}{=} \begin{cases} W_{N_i}(s,t) & \text{si } w = v_i^{\mathsf{bin}} \text{ et } s \in S_{N_i}, \\ 0 & \text{sinon}\,; \end{cases}$$

et de manière similaire pour $W_N(w \triangleleft t, s)$.

Pour chaque place $s \in S_{\Omega_{\mathsf{bin}}}$, avec ${}^\bullet s = \{u_1, \ldots, u_k\}$ et $s^\bullet = \{w_1, \ldots, w_m\}$, pour k et m dans $\{0, 1, 2\}$, nous construisons dans S_N les places de la forme $p \stackrel{\text{df}}{=} (x_1, \ldots, x_k, e_1, \ldots, e_m)$, où chaque x_i (s'il y en a) est une place de sortie de N_{u_i} et chaque e_j (s'il y en a) est une place d'entrée de N_{w_j}. (Remarquons que chaque p est construite une et une seule fois car, dans les boxes opérateurs de ABC, chaque transition a exactement une place en entrée et une en sortie.) L'étiquette de p est celle de s, son marquage est la somme des marquages des x_1, …, x_k, e_1, …, e_m et pour chaque transition $w \triangleleft t$, la fonction de poids est donnée par :

$$W_N(p, w \triangleleft t) \stackrel{\text{df}}{=} \begin{cases} W_{N_w}(x_i,t) + W_{N_w}(e_j,t) & \text{si } w \in {}^\bullet s \cap s^\bullet \text{ et } w = u_i = w_j, \\ W_{N_w}(x_i,t) & \text{si } w \in {}^\bullet s \setminus s^\bullet \text{ et } w = u_i, \\ W_{N_w}(e_j,t) & \text{si } w \in s^\bullet \setminus {}^\bullet s \text{ et } w = w_j, \\ 0 & \text{sinon}\,; \end{cases}$$

et de manière similaire pour $W_N(w \triangleleft t, p)$.

Pour chaque symbole de liens $b \in \mathbb{B}$, nous créons une unique place $s^b \stackrel{\text{df}}{=} (s^b_{v_1^{\mathsf{bin}}}, s^b_{v_2^{\mathsf{bin}}})$ dans S_N où $s^b_{v_1^{\mathsf{bin}}}$ et $s^b_{v_2^{\mathsf{bin}}}$ sont les uniques places de statut b dans N_1 et N_2 respectivement. Le marquage de s^b est défini comme la somme des marquages de $s^b_{v_1^{\mathsf{bin}}}$ et $s^b_{v_2^{\mathsf{bin}}}$ et pour chaque transition $w \triangleleft t$, la fonction de poids est donnée par :

$$W_N(s^b, w \triangleleft t) \stackrel{\text{df}}{=} W_{N_w}(s^b_w, t) \quad ,$$

et de manière similaire pour $W_N(w \triangleleft t, s^b)$.

Scoping. L'application de l'opérateur de scoping $\Omega_{\mathsf{sc}\,a}$ à un box N produit un box possédant les places de N et dont l'ensemble de transitions comprend tous les arbres $t \stackrel{\text{df}}{=} v^{\mathsf{sc}\,a} \triangleleft \{t_1, t_2\}$ où t_1 et t_2 sont des transitions de T_N telles que $\{\lambda_N(t_1), \lambda_N(t_2)\} = \{a, \widehat{a}\}$, ainsi que tous les arbres $t' \stackrel{\text{df}}{=} v^{\mathsf{sc}\,a} \triangleleft t_3$ où $t_3 \in T_N$ est telle que $\lambda_N(t_3) \notin \{a, \widehat{a}\}$. L'étiquette des t est τ et chaque t' a l'étiquette du t_3 lui correspondant. La fonction de poids, pour chaque place p, est définie par :

$$\begin{array}{l} W_{\Omega_{\mathsf{sc}\,a}(N)}(p,t) \stackrel{\text{df}}{=} W_N(p,t_1) + W_N(p,t_2) \quad \text{et} \\ W_{\Omega_{\mathsf{sc}\,a}(N)}(p,t') \stackrel{\text{df}}{=} W_N(s,t_3) \quad ; \end{array}$$

et de manière similaire pour $W_{\Omega_{\mathsf{sc}\,a}(N)}(t,p)$ et $W_{\Omega_{\mathsf{sc}\,a}(N)}(t',p)$.

Restriction de liens. L'application de l'opérateur $\Omega_{\mathsf{tie}\,b}$ à un box N produit un réseau similaire à N mais dans lequel l'identité de chaque transition $t \in T_N$ est changée en $v^{\mathsf{tie}\,b} \triangleleft t$; de plus, le statut de l'unique place de statut b est changé en b et nous ajoutons à $S_{\Omega_{\mathsf{tie}\,b}(N)}$ une nouvelle place de statut b, isolée (sans arc adjacent) et non marquée.

8.3.3 Cohérence dans le domaine des boxes

Pour un box composite, la propriété d'avoir un marquage de contrôle vide est directement liée à la même propriété sur les boxes qui le composent.

Proposition 11.
Soient $\mathbf{N} = (N_1, N_2)$ *une paire de boxes et* N *un box.*

1. *Si* Ω *est un opérateur binaire de ABC, alors* $M^{\mathsf{c}}_{\Omega(\mathbf{N})} = \varnothing$ *ssi* $M^{\mathsf{c}}_{N_1} = \varnothing = M^{\mathsf{c}}_{N_2}$.

2. *Si* N' *est construit comme* $N\,\mathsf{sc}\,a$, $N\,\mathsf{tie}\,b$ *ou* $N.B$, *pour* $a \in \mathbb{A}$, $b \in \mathbb{B}$ *et* $B \in \mathsf{mult}(\mathbb{B})$, *alors* $M^{\mathsf{c}}_{N'} = \varnothing$ *ssi* $M^{\mathsf{c}}_{N} = \varnothing$.

Preuve. Conséquence de la proposition 29 page 155. □

Nous montrons maintenant que la substitution de transitions renvoie toujours un objet syntaxiquement correct si elle est appliquée avec des domaines corrects.

Théorème 12.
Soit Ω *un opérateur binaire de ABC et soit* $\mathbf{N} \in \mathsf{dom}(\Omega)$. *Alors,* $\Omega(\mathbf{N})$ *est un box dont le marquage est sain et ac-libre. De plus, si tous les boxes dynamiques (s'il en existe) de* $\mathbf{N}$ *ont des marquages quasi-sûrs, alors il en est de même pour* $\Omega(\mathbf{N})$.

Preuve. Conséquence du théorème 32 page 156 et des définitions. □

Théorème 13.
Soient N *un box statique ou dynamique,* $a \in \mathbb{A}$, $b \in \mathbb{B}$ *et* $B \in \mathsf{mult}(\mathbb{B})$. *Alors,* $N\,\mathsf{sc}\,a$, $N\,\mathsf{tie}\,b$ *et* $N.B$ *sont des boxes au marquage sain et ac-libre. De plus, si le marquage de* N *est quasi-sûr, celui de ces boxes l'est aussi.*

Preuve. Conséquence du théorème 32 page 156 et des définitions. □

Terminons cette section en remarquant que si nous n'utilisons pas l'opérateur de marquage $.B$ ni l'un des boxes de base N_{ab^+}, N_{ab^-} ou $N_{ab^\pm}$, alors les opération que nous obtenons sont similaires à celles existant dans l'algèbre des boxes standard (PNA) [Bes$^+$92, Bes$^+$01a, Bes$^+$01b] ; la seule différence réside dans les places de liens ouvertes qui sont toutes isolées et non marquées et l'abandon de certaines contraintes.

8.4 Relations entre structure et comportement des boxes composites

Dans cette section, nous mettons en évidence comment le comportement d'un box composite dépend de celui des boxes qui le composent.

8.4.1 Propriétés statiques de boxes

Nous donnons pour commencer un résultat important du point de vue du développement d'une algèbre de réseaux et d'expressions.

Proposition 14.
Soient Ω un opérateur binaire de ABC, N_1 et N_2 deux boxes statiques, N un box statique ou dynamique, $a \in \mathbb{A}$, $b \in \mathbb{B}$ et $B, B' \in \mathsf{mult}(\mathbb{B})$.

1. $\overline{N_1 \,\square\, N_2} = \overline{N_1} \,\square\, N_2 = N_1 \,\square\, \overline{N_2}$ *et* $\underline{N_1 \,\square\, N_2} = \underline{N_1} \,\square\, N_2 = N_1 \,\square\, \underline{N_2}$.
2. $\overline{N_1 \circledast N_2} = \overline{N_1} \circledast N_2 = \underline{N_1} \circledast N_2 = N_1 \circledast \overline{N_2}$ *et* $N_1 \circledast \underline{N_2} = \underline{N_1 \circledast N_2}$.
3. $\overline{N_1; N_2} = \overline{N_1}; N_2$, $\underline{N_1}; N_2 = N_1; \overline{N_2}$ *et* $N_1; \underline{N_2} = \underline{N_1; N_2}$.
4. $\overline{N_1 \| N_2} = \overline{N_1} \| \overline{N_2}$ *et* $\underline{N_1} \| \underline{N_2} = \underline{N_1 \| N_2}$.
5. $\overline{N \,\mathsf{sc}\, a} = \overline{N} \,\mathsf{sc}\, a$, $\underline{N} \,\mathsf{sc}\, a = \underline{N \,\mathsf{sc}\, a}$, $\overline{N \,\mathsf{tie}\, b} = \overline{N} \,\mathsf{tie}\, b$ *et* $\underline{N} \,\mathsf{tie}\, b = \underline{N \,\mathsf{tie}\, b}$.
6. $(N \,\mathsf{sc}\, a).B = (N.B) \,\mathsf{sc}\, a$ *et* $(N \,\mathsf{tie}\, b).B = (N.B) \,\mathsf{tie}\, b$ *à condition que* $b \notin B$.
7. *Si* $(N_1, N_2) \in \mathsf{dom}(\Omega)$ *alors* $\Omega((N_1.B, N_2.B)) = \Omega((N_1, N_2)).(B + B')$.

Preuve. Conséquence des propositions 33 page 157 et 34(2) page 158 et des définitions. □

8.4.2 Équivalences structurelles

Notre intention ici est de capturer les situations où différentes applications du même opérateur conduisent à des résultats identiques. Nous commençons par définir plusieurs relations auxiliaires. Pour $\mathbf{N} \stackrel{\text{df}}{=} (N_1, N_2)$ et $\mathbf{N}' \stackrel{\text{df}}{=} (N_1', N_2')$ deux paires de boxes, nous définissons les relations suivantes.

- $\mathbf{N} \equiv_0 \mathbf{N}'$ s'il existe un box N tel que $\{N_1, N_1'\} = \{\overline{N}, \underline{N}\}$ et $N_2 = N_2'$. Cette relation est destinée à l'itération lorsque son premier argument est dans un marquage initial, ce qui est équivalent à l'avoir dans un marquage final.
- $\mathbf{N} \equiv_1 \mathbf{N}'$ s'il existe des boxes N_1'' et N_2'' tels que $\{\mathbf{N}, \mathbf{N}'\} = \{(\overline{N_1''}, N_2''), (N_1'', \overline{N_2''})\}$. Cette relation est destinée au choix et à l'itération en marquage initial, ce qui peut être obtenu par le marquage initial de l'un ou l'autre des ses arguments.
- $\mathbf{N} \equiv_2 \mathbf{N}'$ s'il existe des boxes N_1'' et N_2'' tels que $\{\mathbf{N}, \mathbf{N}'\} = \{(\underline{N_1''}, N_2''), (N_1'', \underline{N_2''})\}$. Cette relation met en jeu le marquage final et ressemble à la précédente mais pour le choix seulement.
- $\mathbf{N} \equiv_3 \mathbf{N}'$ s'il existe des boxes N_1'' et N_2'' tels que $\{\mathbf{N}, \mathbf{N}'\} = \{(\underline{N_1''}, N_2''), (N_1'', \overline{N_2''})\}$. Cette relation est destinée à la séquence et à l'itération lorsque le premier argument a atteint un marquage final, cela correspond à avoir le second argument dans un marquage initial.
- $\mathbf{N} \equiv_4 \mathbf{N}'$ s'il existe des boxes N_1'' et N_2'', et des éléments de $\mathsf{mult}(\mathbb{B})$ B_1, B_2, B_1' et B_2' tels que $B_1 + B_2 = B_1' + B_2'$, $\mathbf{N} = (N_1''.B_1, N_2''.B_2)$ et $\mathbf{N}' = (N_1''.B_1', N_2''.B_2')$. Autrement dit, les paires $\mathbf{N}$ et $\mathbf{N}'$ ne diffèrent que par la distribution des jetons dans les places de liens ouvertes de même statuts. Cette relation est destinée aux opérateurs binaires, qui assurent la fusion des places de liens ouvertes et ajoutent leurs marquages.

Rappelons que, étant données deux relations binaires R et R' sur un ensemble E, nous définissons $R \circ R' \stackrel{\text{df}}{=} \{(x,y) \mid \exists z \in E : (x,z) \in R \wedge (z,y) \in R'\}$.

Nous définissons maintenant une relation binaire pour chacun des opérateurs de ABC :

$$\begin{array}{llll} \equiv_{\Omega_\parallel} \stackrel{\text{df}}{=} \equiv_4 & & \equiv_{\Omega_\Box} \stackrel{\text{df}}{=} \equiv_4 \circ (id_{\mathsf{abox}} \cup \equiv_1 \cup \equiv_2) \\ \equiv_{\Omega_{\mathsf{sc}\,a}} \stackrel{\text{df}}{=} id_{\mathsf{abox}} & & \equiv_{\Omega_\circledast} \stackrel{\text{df}}{=} \equiv_4 \circ (id_{\mathsf{abox}} \cup \equiv_0 \cup \equiv_1 \cup \equiv_3) \\ \equiv_{\Omega_{\mathsf{tie}\,b}} \stackrel{\text{df}}{=} id_{\mathsf{abox}} & & \equiv_{\Omega_;} \stackrel{\text{df}}{=} \equiv_4 \circ (id_{\mathsf{abox}} \cup \equiv_3) \end{array}$$

où id_{abox} est l'identité sur abox.

Les relations $\equiv_\Omega$ sont réflexives et symétriques, mais elles ne sont pas transitives en général. En effet, pour deux boxes N_1 et N_2, nous avons :

$$\begin{array}{ll} (\overline{\overline{N_1}}, N_2) \equiv_{\Omega_\Box} (\overline{N_1}, \overline{N_2}) & \text{et} \\ (\overline{N_1}, \overline{N_2}) \equiv_{\Omega_\Box} (N_1, \overline{\overline{N_2}}) & \text{mais} \\ (\overline{\overline{N_1}}, N_2) \not\equiv_{\Omega_\Box} (N_1, \overline{\overline{N_2}}) & . \end{array}$$

Cependant, comme le montre le résultat suivant, si nous les restreignons aux domaines des opérateurs correspondants, elles deviennent transitives et identifient les Ω-uplets de boxes qui produisent les mêmes réseaux composites.

Proposition 15.
Soit Ω un box opérateur de ABC, et soient $\mathbf{N} \in \mathsf{dom}(\Omega)$ et $\mathbf{N}'$ deux Ω-uplets de boxes.

1. *Si $\mathbf{N} \equiv_\Omega \mathbf{N}'$ alors $\mathbf{N}' \in \mathsf{dom}(\Omega)$ et $\llbracket \mathbf{N} \rrbracket = \llbracket \mathbf{N}' \rrbracket$.*
2. *$\equiv_\Omega$ est une relation d'équivalence sur $\mathsf{dom}(\Omega)$.*
3. *Si $\llbracket \mathbf{N} \rrbracket = \llbracket \mathbf{N}' \rrbracket$ alors $\Omega(\mathbf{N}) = \Omega(\mathbf{N}')$ ssi $\mathbf{N} \equiv_\Omega \mathbf{N}'$.*

Preuve. Conséquence de la proposition 34 page 158. □

La condition $\llbracket \mathbf{N} \rrbracket = \llbracket \mathbf{N}' \rrbracket$ dans la troisième propriété ne peut pas être supprimée. En effet, nous avons $N_{ab^+} \not\equiv_{\Omega_{\mathsf{sc}\,a}} N_{ab^-}$ mais pourtant $N_{ab^+}\,\mathsf{sc}\,a = N_{ab^-}\,\mathsf{sc}\,a$ (les scopings ne laissent aucune transition).

8.4.3 Propriétés dynamiques des boxes composites

Dans les résultats qui suivent, nous capturons la compositionnalité comportementale de notre modèle, c'est-à-dire la façon dont le comportement des boxes composites (en termes de steps franchissables) est relié au comportement de leurs constituants. Dans sa plus simple expression, cela revient à établir quels steps de $\Omega(\mathbf{N})$ sont franchissables sachant ceux qui le sont dans $\mathbf{N}$.

Proposition 16.
Soient $\Omega_{\mathsf{bin}} \in \{\Omega_\parallel, \Omega_\Box, \Omega_\circledast, \Omega_;\}$ et $\mathbf{N} \in \mathsf{dom}(\Omega_{\mathsf{bin}})$. Alors $\mathsf{enabled}(\Omega_{\mathsf{bin}}(\mathbf{N}))$ contient exactement tous les ensembles de transitions $U = (v_1^{\mathsf{bin}} \lhd\!\lhd U_1) \cup (v_2^{\mathsf{bin}} \lhd\!\lhd U_2)$ tels qu'il existe une paire de boxes $\mathbf{N}' = (N_1', N_2')$ satisfaisant $\mathbf{N}' \equiv_{\Omega_{\mathsf{bin}}} \mathbf{N}$ et $U_i \in \mathsf{enabled}(N_i')$ pour $i \in \{1,2\}$. De plus, $\Omega_{\mathsf{bin}}(\mathbf{N})[U\rangle\Omega_{\mathsf{bin}}(\mathbf{N}'')$ où $N_i'[U_i\rangle N_i''$ pour $i \in \{1,2\}$.

Preuve. Pour $\Omega_{\|}$, le résultat est une conséquence des définitions ; $\equiv_{\Omega_{\|}}$ est nécessaire pour réarranger les jetons dans les places de liens ouvertes de N_1 et N_2, de façon à ce que les steps U_1 et U_2 soient franchissables séparément.

Pour les autres opérateurs, le résultat découle de la proposition 35 page 161. □

Proposition 17.
Soient N un box, $a \in \mathbb{A}$, $b \in \mathbb{B}$, $B \in \mathsf{mult}(\{b\})$ et $B' \in \mathsf{mult}(\mathbb{B})$.

1. $\mathsf{enabled}(N\,\mathsf{sc}\,a)$ *comprend exactement tous les ensembles de transitions*

$$U = (v^{\mathsf{sc}\,a} \ll Z) \cup \{v^{\mathsf{sc}\,a} \lhd \{v_1, w_1\}, \ldots, v^{\mathsf{sc}\,a} \lhd \{v_k, w_k\}\}$$

tels que $Z \cup V \cup W \in \mathsf{enabled}(N)$, où les steps Z, $V \stackrel{\text{df}}{=} \{v_1, \ldots, v_k\}$ et $W \stackrel{\text{df}}{=} \{w_1, \ldots, w_k\}$ satisfont à $\{a, \widehat{a}\} \cap \lambda_N(Z) = \varnothing$, $\lambda_N(V) = \{a\}$ et $\lambda_N(W) = \{\widehat{a}\}$. De plus, $N\,\mathsf{sc}\,a[U\rangle N'\,\mathsf{sc}\,a$, où $N[Z \cup V \cup W\rangle N'$.

2. $\mathsf{enabled}(N\,\mathsf{tie}\,b)$ *comprend exactement tous les ensembles $U = v^{\mathsf{tie}\,b} \ll V$ tels que $V \in \mathsf{enabled}(N)$. De plus, $N\,\mathsf{tie}\,b[U\rangle N'\,\mathsf{tie}\,b$, où $N[V\rangle N'$.*

3. $\mathsf{enabled}((N\,\mathsf{tie}\,b).B)$ *comprend exactement tous les ensembles $U \in \mathsf{enabled}(N\,\mathsf{tie}\,b)$. De plus, $(N\,\mathsf{tie}\,b).B[U\rangle N'.B$, où $N\,\mathsf{tie}\,b[U\rangle N'$.*

4. $\mathsf{enabled}(N)$ *est un sous-ensemble de $\mathsf{enabled}(N.B')$. De plus, si $N[U\rangle N'$, alors $N.B'[U\rangle N'.B'$.*

Preuve. Par définition. □

La réciproque de la dernière propriété n'est pas vraie, en effet, le box dynamique $\overline{N_{ab^-}}$ n'autorise pas le franchissement de step non vide alors que $\overline{N_{ab^-}}.b[\{v^{ab^-}\}\rangle \underline{N_{ab^-}}$.

Le comportement des boxes asynchrones de base de ABC est capturé ci-dessous.

Proposition 18.
Soit $B \in \mathsf{mult}(\mathbb{B})$ et soit $N = \overline{N_\beta}.B$, où N_β est l'un des boxes de base donnés figure 8.6.

1. *Pour $\beta \in \{a, ab^+\}$, les steps non vides de N sont, respectivement, $N[\{v^a\}\rangle \underline{N_a}.B$ et $N[\{v^{ab^+}\}\rangle \underline{N_{ab^+}}.b.B$.*

2. *Pour $\beta \in \{ab^-, ab^\pm\}$, si $b \notin B$ alors N n'a que des steps vides, sinon, les steps non vides de N sont, respectivement, $N[\{v^{ab^-}\}\rangle \underline{N_{ab^-}}.(B - \{b\})$ et $N[\{v^{ab^\pm}\}\rangle \underline{N_{ab^\pm}}.B$.*

Preuve. Par définition. □

Nous pouvons tirer plusieurs conséquences importantes des résultats présentés ci-dessus. En particulier, que la façon dont les boxes dynamiques et statiques sont composés garantit que le résultat est un box statique ou dynamique si le domaine des opérateurs est respecté.

Proposition 19.
Soit Ω un opérateur de ABC et soit $\mathbf{N} \in \mathsf{dom}(\Omega)$. Alors, tout réseau dérivable de $\Omega(\mathbf{N})$ est de la forme $\Omega(\mathbf{N}')$ où $\mathbf{N}' \in \mathsf{dom}(\Omega)$ et $[\![\mathbf{N}]\!] = [\![\mathbf{N}']\!]$. De plus, si aucun box dans $\mathbf{N}$

n'est dynamique, alors tout réseau dérivable de $\overline{\Omega(\mathbf{N})}$ *ou de* $\underline{\Omega(\mathbf{N})}$ *est de la forme* $\Omega(\mathbf{N}')$*, où* $\mathbf{N}' \in \mathsf{dom}(\Omega)$ *et* $[\![\mathbf{N}]\!] = [\![\mathbf{N}']\!]$*.*

Preuve. La première partie résulte des propositions 16 et 17(1,2). La seconde partie est conséquence de la première et de la proposition 14. □

Théorème 20.
Tout réseau composite de ABC est un box dynamique ou statique dont le marquage est quasi-sûr. De plus, il est statique ssi les opérateurs de marquages $\overline{(\cdot)}$ *et* $\underline{(\cdot)}$ *ne sont pas utilisés, à moins qu'ils ne soient placés sous un opérateur* $\lfloor\cdot\rfloor$ *ou* $[\![\cdot]\!]$*.*

Preuve. Résulte du théorème 12, des propositions 10 et 19 et du fait que les boxes asynchrones des figures 8.6 et 8.7 sont tous statiques. Les réseaux composites sont quasi-sûrs en conséquence du théorème 12 et du fait que tous les marquages sains des réseaux des figures 8.6 et 8.7 sont quasi-sûrs (il s'agit soit des marquages d'entrée soit des marquages de sortie). □

8.5 Une algèbre d'expressions

Nous considérons une algèbre d'expressions sur la signature :

$$\{a, ab^+, ab^-, ab^\pm \mid a \in \mathbb{A}_\tau, b \in \mathbb{B}\} \cup \{\overline{(\cdot)}, \underline{(\cdot)}\} \cup \{\|, ;, \square, \circledast\} \cup \{\mathsf{sc}\, a \mid a \in \mathbb{A}\} \cup \{\mathsf{tie}\, b \mid b \in \mathbb{B}\} \cup \{.b \mid b \in \mathbb{B}\} \quad ,$$

où les constantes sont les éléments de $\{a, ab^+, ab^-, ab^\pm \mid a \in \mathbb{A}_\tau, b \in \mathbb{B}\}$, les opérateurs binaires dans $\{\|, ;\square, \circledast\}$ sont utilisés en notation infixe, les opérateurs unaires de $\{\mathsf{sc}\, a \mid a \in \mathbb{A}\} \cup \{\mathsf{tie}\, b \mid b \in \mathbb{B}\} \cup \{.b \mid b \in \mathbb{B}\}$ sont utilisés en notation postfixe et pour les opérateurs de $\{\overline{(\cdot)}, \underline{(\cdot)}\}$, la position de l'argument est indiquée par $(\cdot)$.

Nous distinguons deux classes d'expressions ABC qui correspondent aux boxes statiques et dynamiques et qui sont les *expressions statiques* et *dynamiques*, notées respectivement aexpr_s et aexpr_d. Nous notons aussi aexpr la réunion des deux classes. Leur syntaxe est :

$$\begin{array}{lllllllllll}
\mathsf{aexpr}_s & E & ::= & \beta & | & E\,\mathsf{sc}\,a & | & E\,\mathsf{tie}\,b & | & E.b & | & E\|E & | & E \square E \\
 & & & E;E & | & E \circledast E & & & & & & & & \\
\mathsf{aexpr}_d & D & ::= & \overline{E} & | & \underline{E} & | & D\,\mathsf{sc}\,a & | & D\,\mathsf{tie}\,b & | & D.b & | & D\|D \\
 & & & D \square E & | & E \square D & | & D;E & | & E;D & | & D \circledast E & | & E \circledast D
\end{array}$$

où $\beta \in \{a, ab^+, ab^-, ab^\pm \mid a \in \mathbb{A}_\tau, b \in \mathbb{B}\}$, $a \in \mathbb{A}$ et $b \in \mathbb{B}$. De plus, F est utilisé pour représenter n'importe quelle expression statique ou dynamique.

Nous utilisons les notations $\lfloor F \rfloor$ et $[\![F]\!]$ permettant de produire des expression statiques. $\lfloor F \rfloor$ est F dont toutes les occurrences de $\overline{(\cdot)}$ et de $\underline{(\cdot)}$ on été supprimées ; $[\![F]\!]$ est $\lfloor F \rfloor$ dont toutes les occurrences de $.b$ ont été supprimées. Remarquons qu'il n'y a pas de terme de la forme $F.\{b, \ldots, b'\}$ car ils pourront s'obtenir par $F.b.\cdots.b'$, mais cette notation pourra servir comme raccourci.

Essentiellement, une expression encode la structure d'un box, ainsi que son marquage courant au niveau des places de contrôle (avec les barres) et des places de liens (avec les $.b$). Donc, par exemple, une expression $\overline{E}$ représente E dans son état initial, ce qui correspond en termes de réseaux à un marquage initial du box de E.

Remarquons que la notation $.b$ est utilisée pour les expression statiques comme dynamiques. En effet, une partie dormante d'une expression peut avoir des $.b$ qui seront utilisés par la suite dans une partie active. Par exemple, $ab^+.b; \overline{fb^-}$ a une partie statique dormante avec un $.b$ mais peut être transformée en $ab^+; \overline{fb^-.b}$ qui peut s'exécuter (voir section 8.5.2).

8.5.1 Sémantique dénotationnelle

La sémantique dénotationnelle des expressions est donnée par une fonction $\mathsf{box} : \mathsf{aexpr} \to \mathsf{abox}$, définie comme un homomorphisme, par induction sur la structure des expressions, en suivant leur syntaxe. Soient $\beta \in \{a, ab^+, ab^-, ab^\pm \mid a \in \mathbb{A}_\tau, b \in \mathbb{B}\}$, $a \in \mathbb{A}$ et $b \in \mathbb{B}$, soient de plus $\mathsf{una} \in \{\mathsf{sc}\, a, \mathsf{tie}\, b, .b\}$ et $\mathsf{bin} \in \{\|, \Box, ;, \circledast\}$. Alors :

$$\begin{aligned}
\mathsf{box}(\beta) &\stackrel{\text{df}}{=} N_\beta \quad , \\
\mathsf{box}(\overline{E}) &\stackrel{\text{df}}{=} \overline{\mathsf{box}(E)} \quad , \\
\mathsf{box}(\underline{E}) &\stackrel{\text{df}}{=} \underline{\mathsf{box}(E)} \quad , \\
\mathsf{box}(F\,\mathsf{una}) &\stackrel{\text{df}}{=} \mathsf{box}(F)\,\mathsf{una} \quad , \\
\mathsf{box}(F_1\,\mathsf{bin}\,F_2) &\stackrel{\text{df}}{=} \mathsf{box}(F_1)\,\mathsf{bin}\,\mathsf{box}(F_2) \quad .
\end{aligned}$$

Cette fonction sémantique renvoie toujours une réseau syntaxiquement correct et la propriété d'être un box statique ou dynamique a été correctement capturée par la syntaxe.

Théorème 21.
Soit F une expression ABC.

1. $\mathsf{box}(F)$ *est un box statique ou dynamique.*
2. $\mathsf{box}(F)$ *est un box statique ssi F est une expression statique.*

Preuve. Provient du théorème 20, par induction sur la structure de F. □

8.5.2 Relation de similarité structurelle

La relation de *similarité structurelle* sur les expressions ABC, notée $\equiv$, est définie comme la plus petite relation d'équivalence telle que les équations de la figure 8.9 sont satisfaites.

En utilisant ces règles d'équivalence, nous pouvons dériver que $\overline{(a \Box c) \| (d; e)} \equiv (\overline{a} \Box c) \| (\overline{d}; e)$, comme montré sur la figure 8.10.

Les règles de la figure 8.9 découlent directement de celles de PNA ou bien sont ajoutées pour capturer le fait que les messages asynchrones, produits par les ab^+ et représentés par les $.b$, peuvent se déplacer librement dans une expression de façon à être consommés

CON1	$\dfrac{F \equiv F'}{F\,\mathsf{una} \equiv F'\,\mathsf{una}}$	CON2	$\dfrac{F_1 \equiv F_1', F_2 \equiv F_2'}{F_1\,\mathsf{bin}\,F_2 \equiv F_1'\,\mathsf{bin}\,F_2'}$
ENT	$\dfrac{E \equiv E'}{\overline{E} \equiv \overline{E'}}$	EX	$\dfrac{E \equiv E'}{\underline{E} \equiv \underline{E'}}$
OPL	$(F.b)\,\mathsf{bin}\,F' \equiv (F\,\mathsf{bin}\,F').b$	OPR	$F\,\mathsf{bin}(F'.b) \equiv (F\,\mathsf{bin}\,F').b$
E1	$\overline{E}\,\mathsf{una} \equiv \overline{E\,\mathsf{una}}$	X1	$\underline{E}\,\mathsf{una} \equiv \underline{E\,\mathsf{una}}$
B1	$(F.b)\,\mathsf{una} \equiv (F\,\mathsf{una}).b \quad \text{si } \mathsf{una} \neq \mathsf{tie}\,b$	IS1	$\overline{E_1; E_2} \equiv \overline{E_1}; E_2$
IS2	$\underline{E_1}; E_2 \equiv E_1; \overline{E_2}$	IS3	$E_1; \underline{E_2} \equiv \underline{E_1; E_2}$
IPAR1	$\overline{E_1 \| E_2} \equiv \overline{E_1} \| \overline{E_2}$	IPAR2	$\underline{E_1} \| \underline{E_2} \equiv \underline{E_1 \| E_2}$
IC1L	$\overline{E_1 \,\square\, E_2} \equiv \overline{E_1} \,\square\, E_2$	IC1R	$\overline{E_1 \,\square\, E_2} \equiv E_1 \,\square\, \overline{E_2}$
IC2L	$\underline{E_1} \,\square\, E_2 \equiv \underline{E_1 \,\square\, E_2}$	IC2R	$E_1 \,\square\, \underline{E_2} \equiv \underline{E_1 \,\square\, E_2}$
IIT1	$\overline{E_1 \circledast E_2} \equiv \overline{E_1} \circledast E_2$	IIT2	$\underline{E_1} \circledast E_2 \equiv \overline{E_1} \circledast E_2$
IIT3	$\underline{E_1} \circledast E_2 \equiv E_1 \circledast \overline{E_2}$	IIT4	$E_1 \circledast \underline{E_2} \equiv \underline{E_1 \circledast E_2}$

Figure 8.9 — Relation de similarité structurelle pour les expressions ABC. Avec $b \in \mathbb{B}$, una un opérateur unaire et bin un opérateur binaire.

par une action de la forme ab^- (voir les règles B1, OPL et OPR). Cependant, un $.b$ ne peut jamais traverser la limite imposée par l'application de l'opérateur $\mathsf{tie}\,b$: la règle B1 empêche una d'être $\mathsf{tie}\,b$, mais pas d'être $\mathsf{tie}\,b'$ si $b' \neq b$ et la même règle avec $\mathsf{una} = .b'$ donne $F.b.b' \equiv F.b'.b$, c'est-à-dire la commutativité de l'opérateur $.b$.

Nous pouvons remarquer que, grâce aux règles CON1, CON2, ENT et EX, l'équivalence ainsi définie est en fait une congruence sur tous les opérateurs de l'algèbre. Il est de plus facile d'observer que la relation de similarité structurelle est stable sur le domaine des expressions, c'est-à-dire que si une expression correspond à un membre d'une des équations, l'autre membre définit alors une expression correcte. De plus, la relation préserve le type des expressions (statiques ou dynamiques) et identifie les expressions ayant la même sémantique dénotationnelle comme nous le montrons maintenant.

Théorème 22.
Soient F_1 et F_2 deux expressions ABC.

1. *Si $F_1 \equiv F_2$ alors $\lfloor F_1 \rfloor = \lfloor F_2 \rfloor$, $[\![F_1]\!] = [\![F_2]\!]$ et $\mathsf{box}(F_1) = \mathsf{box}(F_2)$.*
2. *Si $[\![F_1]\!] = [\![F_2]\!]$ alors, $\mathsf{box}(F_1) = \mathsf{box}(F_2)$ ssi $F_1 \equiv F_2$.*

Preuve. Découle des définitions (en particulier, les règles figure 8.9 et celle de box) et des propositions 10, 14 et 15, par induction sur la structure de F. □

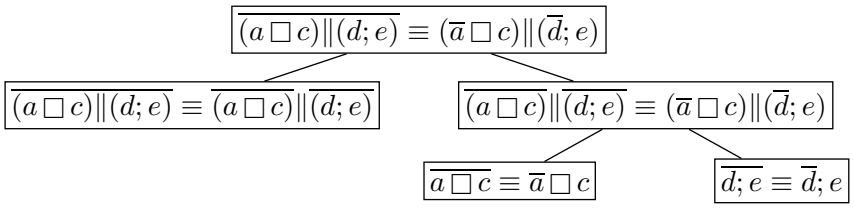


Figure 8.10 — L'arbre de dérivation de $\overline{(a \Box c) \| (d;e)} \equiv (\overline{a} \Box c) \| (\overline{d};e)$.

La condition $[\![F_1]\!] = [\![F_2]\!]$ dans la deuxième propriété est nécessaire. En effet, en prenant $F_1 \stackrel{\text{df}}{=} a \,\mathsf{sc}\, a$ et $F_2 \stackrel{\text{df}}{=} \widehat{a} \,\mathsf{sc}\, a$, nous avons $F_1 \not\equiv F_2$ mais $\mathsf{box}(F_1) = \mathsf{box}(F_2)$ (le scoping ne laisse aucune transition).

Théorème 23.
Soit F une expression ABC.

1. $\mathsf{box}(F) = \overline{\lfloor \mathsf{box}(F) \rfloor}$ *ssi* $F \equiv \overline{\lfloor F \rfloor}$.
2. $\mathsf{box}(F) = \underline{\lfloor \mathsf{box}(F) \rfloor}$ *ssi* $F \equiv \underline{\lfloor F \rfloor}$.

Preuve. Comme pour le théorème 22. □

Dans le premier cas, F est une *expression initiale*, dans le second cas c'est une *expression finale*.

8.5.3 Sémantique opérationnelle basée sur les transitions

Dans le développement d'une sémantique opérationnelle de ABC, nous commençons par introduire des règles opérationnelles. Elles sont basées sur les transitions des réseaux qui donnent la sémantique dénotationnelle des expressions ABC. Grâce à cela, nous pourrons formuler notre principal résultat de cohérence. Par la suite, nous introduirons les règles basées sur les étiquettes des transitions, avec les résultats associés.

Soit $\mathbb{T}$ l'ensemble des transitions (il s'agit d'arbres comme défini en section 8.3.2) qui peuvent être obtenus à partir de la fonction box. Nous pouvons montrer facilement que chaque transition $t \in \mathbb{T}$ a toujours la même étiquette quelle que soit la façon dont elle est obtenue par box. Nous notons donc $\lambda(t)$ cette étiquette. Notons de plus $\mathbb{U} \stackrel{\text{df}}{=} \mathsf{mult}(\mathbb{T})$.

Dans la première sémantique opérationnelle, nous considérons des évolutions de la forme $F \xrightarrow{U} F'$, telles que F et F' sont des expressions et $U \in \mathbb{U}$. L'idée est que U est un step valide entre les boxes associés à F et F', c'est-à-dire que $\mathsf{box}(F)[U\rangle\mathsf{box}(F')$, ce que nous montrons au théorème 25.

Formellement, nous définissons la relation ternaire $\rightarrow$ comme la plus petite relation comprenant tous les triplets $(F, U, F') \in \mathsf{aexpr} \times \mathbb{U} \times \mathsf{aexpr}$ tels que les relations de la figure 8.11 sont vérifiées. Remarquons que nous notons $F \xrightarrow{U} F'$ à la place de $(F, U, F') \in \rightarrow$.

Dans la définition de EOP, nous n'imposons aucune restriction sur U_1 et U_2, mais le domaine d'application de bin assure que cette règle est toujours utilisée avec une

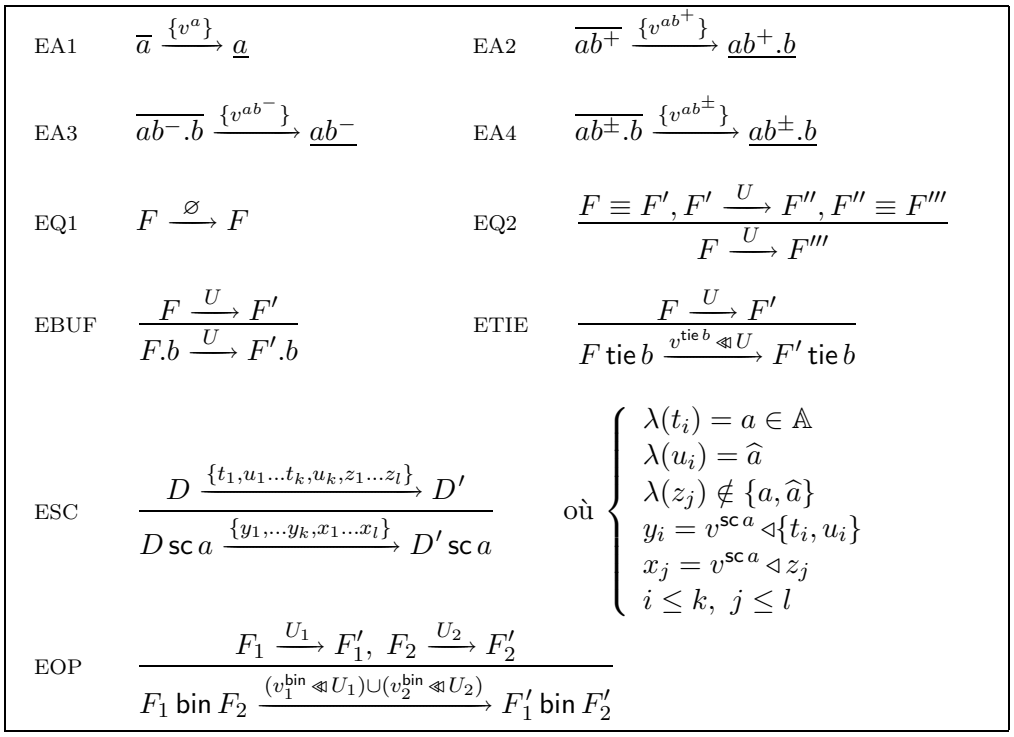


Figure 8.11 — Règles de la sémantique opérationnelle basées sur les transitions, avec $a \in \mathbb{A}_\tau$ (sauf dans la règle ESC où $a \neq \tau$), $b \in \mathbb{B}$ et où bin est un opérateur binaire de ABC.

combinaison correcte d'expressions dynamiques et statiques. Par exemple, dans le cas du choix, U_1 ou U_2 est nécessairement vide.

Nous donnons maintenant des propriétés sur les règles de dérivation. En premier lieu, une évolution vide correspond toujours à deux expressions structurellement similaires.

Proposition 24.
Soient F et F' deux expressions ABC. Alors, $F \xrightarrow{\varnothing} F'$ ssi $F \equiv F'$.

Preuve. ($\Rightarrow$) Considérons l'arbre de dérivation A permettant d'obtenir $F \xrightarrow{\varnothing} F'$. Comme aucune règle ne permet d'obtenir un step vide à partir d'un step non vide, les feuilles de A doivent faire référence à des instances de la règle EQ1. Nous obtenons alors le résultat en raisonnant par induction sur la structure de A et en prenant en compte que la similarité structurelle est une congruence et que toutes les autres règles d'inférence préservent l'équivalence des expressions.

($\Leftarrow$) Conséquence des règles EQ1 et EQ2 en prenant $F = F' = F'' = F'''$. □

Le résultat suivant montre qu'une évolution avec la sémantique opérationnelle transforme une expression en une autre qui lui est statiquement équivalente. De plus, cela correspond à un step valide du box donnant la sémantique dénotationnelle. Nous in-

terprétons cela comme la correction de la sémantique opérationnelle des expressions. L'implication est ensuite renversée pour obtenir la complétude.

Théorème 25.
Soit F une expression ABC.

1. *Si $F \xrightarrow{U} F'$, alors F' est une expression ABC telle que $\mathsf{box}(F)[U\rangle\mathsf{box}(F')$ et $[\![F]\!] = [\![F']\!]$.*
2. *Si $\mathsf{box}(F)[U\rangle N'$, alors il existe une expression ABC F' telle que $\mathsf{box}(F') = N'$ et $F \xrightarrow{U} F'$.*

Preuve. Nous prouvons le résultat par induction sur la structure de F, à partir des règles des figures 8.9 et 8.11, et en s'appuyant sur la définition de box, les propositions 14, 16, 17, 18 et 24 et le théorème 22(2). □

À titre d'exemple, la figure 8.12 donne la séquence d'évolutions correspondant au scénario 1 (voir section 8.1) et la figure 8.13 donne l'arbre de dérivation pour l'étape $\xrightarrow{\{t_2,t_4\}}$ (dixième ligne dans la figure 8.12).

8.5.4 Cohérence des sémantiques dénotationnelles et opérationnelles

Cette section présente le résultat principal de ce chapitre : les sémantiques dénotationnelles et opérationnelles des expressions ABC sont montrées cohérentes : elles génèrent des systèmes de transitions isomorphes.

Le système de transitions (complet ou étiqueté) d'un box dynamique N_d est $\mathsf{fts}(N_d)$, défini comme auparavant, celui d'un box statique N_s est $\mathsf{fts}(N_s) \stackrel{\text{df}}{=} \mathsf{fts}(\overline{N_s})$.

Soit D une expression dynamique. Nous notons $[D\rangle$ l'ensemble des expression dérivables à partir de D, il s'agit du plus petit ensemble contenant D tel que si $D' \in [D\rangle$ et $D' \xrightarrow{U} D''$ pour un $U \in \mathbb{U}$, alors $D'' \in [D\rangle$. Par ailleurs, Nous notons $[D]_{\equiv}$ la classe d'équivalence de $\equiv$ contenant D. Le *système de transitions complet* de D est $\mathsf{fts}(D) \stackrel{\text{df}}{=} (V, L, A, \mathit{init})$ où $V \stackrel{\text{df}}{=} \{[D']_{\equiv} \mid D' \in [D\rangle\}$ est l'ensemble des états, $L \stackrel{\text{df}}{=} \mathbb{U}$ est l'ensemble des étiquettes d'arcs, $A \stackrel{\text{df}}{=} \{([D']_{\equiv}, U, [D'']_{\equiv}) \in V \times \mathbb{U} \times V \mid D' \xrightarrow{U} D''\}$ est l'ensemble des arcs et $\mathit{init} \stackrel{\text{df}}{=} [D]_{\equiv}$ est l'état initial. Pour une expression statique E, nous posons $\mathsf{fts}(E) \stackrel{\text{df}}{=} \mathsf{fts}(\overline{E})$.

Remarquons que nous basons les systèmes de transitions des expressions sur les classes d'équivalences de $\equiv$ plutôt que sur les expressions elles mêmes. La raison est que nous pouvons avoir $D \xrightarrow{\varnothing} D'$ pour deux expressions D et D' distinctes alors que, pour les boxes, $N[\varnothing\rangle N'$ implique toujours $N = N'$.

Théorème 26.
Pour toute expression ABC F, $\mathsf{iso}_f(F) \stackrel{\text{df}}{=} \{([F']_{\equiv}, \mathsf{box}(F')) \mid [F']_{\equiv}$ est un nœud de $\mathsf{fts}(F)\}$ est un isomorphisme entre $\mathsf{fts}(F)$ et $\mathsf{fts}(\mathsf{box}(F))$. De plus, $\mathsf{iso}_f(F)$ préserve la propriété d'être en état initial ou final (en termes de boxes et d'expressions, et non en termes de système de transitions).

Preuve. Conséquence du théorème 25, de la proposition 24 et de la définition de la fonction sémantique box. □

$$
\begin{array}{lll}
\overline{s;(((pb^+ \circledast f)\|(pb^+ \circledast f)\|(cb^- \circledast f))\,\mathsf{tie}\,b)} & & \\
\quad\equiv & \overline{s};(((pb^+ \circledast f)\|(pb^+ \circledast f)\|(cb^- \circledast f))\,\mathsf{tie}\,b) & \text{IS1} \\
\xrightarrow{\{t_1\}} & \underline{s};(((pb^+ \circledast f)\|(pb^+ \circledast f)\|(cb^- \circledast f))\,\mathsf{tie}\,b) & \text{EA1, EOP} \\
\quad\equiv & s;\overline{(((pb^+ \circledast f)\|(pb^+ \circledast f)\|(cb^- \circledast f))\,\mathsf{tie}\,b)} & \text{IS2, E1} \\
\quad\equiv & s;(((\overline{pb^+} \circledast f)\|(\overline{pb^+} \circledast f)\|(\overline{cb^-} \circledast f))\,\mathsf{tie}\,b) & \text{IPAR1, IIT1} \\
\xrightarrow{\{t_2,t_3\}} & s;(((\underline{pb^+.b} \circledast f)\|(\underline{pb^+.b} \circledast f)\|(\overline{cb^-} \circledast f))\,\mathsf{tie}\,b) & \text{EA2, EOP, ETIE} \\
\quad\equiv & s;(((\underline{pb^+.b} \circledast f)\|(\underline{pb^+} \circledast f)\|(\overline{cb^-} \circledast f).b)\,\mathsf{tie}\,b) & \text{X1, OPL-R} \\
\quad\equiv & s;(((\underline{pb^+.b} \circledast f)\|(\underline{pb^+} \circledast f)\|(\overline{cb^-}.b \circledast f))\,\mathsf{tie}\,b) & \text{OPL} \\
\quad\equiv & s;(((\overline{pb^+}.b \circledast f)\|(\underline{pb^+} \circledast f)\|(\overline{cb^-.b} \circledast f))\,\mathsf{tie}\,b) & \text{IIT2, E1} \\
\xrightarrow{\{t_2,t_4\}} & s;(((\underline{pb^+.b}.b \circledast f)\|(\underline{pb^+} \circledast f)\|(\underline{cb^-} \circledast f))\,\mathsf{tie}\,b) & \text{EA2-3, EOP, ETIE} \\
\quad\equiv & s;(((pb^+.b.b \circledast \overline{f})\|(pb^+ \circledast \overline{f})\|(cb^- \circledast \overline{f}))\,\mathsf{tie}\,b) & \text{X1, IIT3} \\
\xrightarrow{\{t_5,t_6,t_7\}} & s;(((pb^+.b.b \circledast \underline{f})\|(pb^+ \circledast \underline{f})\|(cb^- \circledast \underline{f}))\,\mathsf{tie}\,b) & \text{EA1, EOP, ETIE} \\
\quad\equiv & \underline{s;(((pb^+.b.b \circledast f)\|(pb^+ \circledast f)\|(cb^- \circledast f))\,\mathsf{tie}\,b)} & \text{IIT4, IPAR2, X1}
\end{array}
$$

Figure 8.12 — L'exécution correspondant au scénario 1. Les identités des différentes transitions sont : $t_1 = v_1^: \triangleleft v^s$, $t_2 = v_2^: \triangleleft v_1^{\mathsf{tie}\,b} \triangleleft v_1^{\|} \triangleleft v_1^{\|} \triangleleft v_1^{\circledast} \triangleleft v^{pb^+}$, $t_3 = v_2^: \triangleleft v_1^{\mathsf{tie}\,b} \triangleleft v_1^{\|} \triangleleft v_2^{\|} \triangleleft v_1^{\circledast} \triangleleft v^{pb^+}$, $t_4 = v_2^: \triangleleft v_1^{\mathsf{tie}\,b} \triangleleft v_2^{\|} \triangleleft v_1^{\circledast} \triangleleft v^{cb^-}$, $t_5 = v_2^: \triangleleft v_1^{\mathsf{tie}\,b} \triangleleft v_1^{\|} \triangleleft v_1^{\|} \triangleleft v_2^{\circledast} \triangleleft v^f$, $t_6 = v_2^: \triangleleft v_1^{\mathsf{tie}\,b} \triangleleft v_1^{\|} \triangleleft v_2^{\|} \triangleleft v_2^{\circledast} \triangleleft v^f$ et $t_7 = v_2^: \triangleleft v_1^{\mathsf{tie}\,b} \triangleleft v_2^{\|} \triangleleft v_2^{\circledast} \triangleleft v^f$.

8.5.5 Sémantique opérationnelle basée sur les étiquettes

Nous adaptons maintenant la sémantique opérationnelle présentée ci-dessus pour qu'elle fasse apparaître les actions plutôt que les transitions. Les principaux résultats sont étendus à cette nouvelle sémantique.

Pour commencer, nous conservons l'équivalence structurelle $\equiv$ sans aucun changement. Ensuite, nous définissons des évolutions de la forme $F \xrightarrow{\Gamma} F'$ où F et F' sont des expressions comme avant et Γ est un multi-ensemble d'actions dans $\mathbb{L} \stackrel{\text{df}}{=} \mathsf{mult}(\mathbb{A}_\tau)$. Ces évolutions sont définies à partir des règles données sur la figure 8.14.

Les deux types de sémantique opérationnelle sont clairement en relation. En fait, chaque évolution basée sur des étiquettes est une évolution basée sur des transitions dont seules les étiquettes ont été retenues.

Proposition 27.
Soient F une expression ABC et $\Gamma \in \mathbb{L}$. Alors $F \xrightarrow{\Gamma} F'$ ssi il existe $U \in \mathbb{U}$ tel que $F \xrightarrow{U} F'$ et $\lambda(U) = \Gamma$.

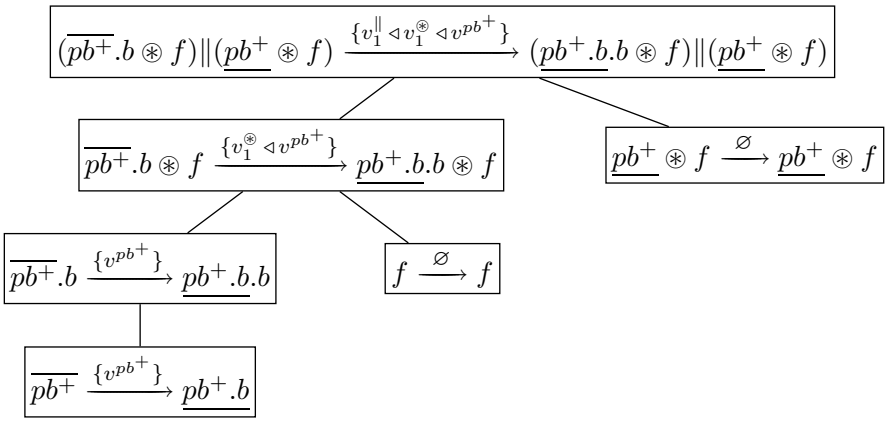


Figure 8.13 — L'arbre de dérivation permettant de déduire $\xrightarrow{\{t_2,t_4\}}$ dans l'exécution présentée figure 8.12.

Preuve. Par définition, en particulier celle de la relation $\longrightarrow$ (voir figure 8.11). □

Les résultats sur la sémantique opérationnelle basée sur les transitions s'étendent donc directement à celle basée sur les étiquettes.

La proposition précédente montre que la sémantique opérationnelle basée sur les étiquettes d'une expression F est fidèlement représentée par le *système de transition étiqueté* de F, noté $\mathsf{lts}(F)$, qui est défini comme $\mathsf{fts}(F)$ dans lequel chaque étiquette d'arc U est changée en $\lambda(U)$. Nous pouvons donc formuler le résultat de cohérence pour la sémantique opérationnelle basée sur les étiquettes.

Théorème 28.
Pour toute expression ABC F, $\mathsf{iso}_l(F) \stackrel{\text{df}}{=} \{([F']_{\equiv}, \mathsf{box}(F')) \mid [F']_{\equiv}$ est un nœud de $\mathsf{lts}(F)\}$ est un isomorphisme entre $\mathsf{lts}(F)$ et $\mathsf{lts}(\mathsf{box}(F))$. De plus, $\mathsf{iso}_l(F)$ préserve la propriété d'être en état initial ou final (en termes de boxes et d'expressions, et non en termes de système de transitions).

Preuve. Conséquences des définitions, du théorème 26 et de la proposition 27. □

Les règles de la sémantique opérationnelle basée sur les étiquettes sont mises en pratique sur les figures 8.15 et 8.16, pour le deuxième et le troisième scénario présentés à la section 8.1.

Pour illustrer comment la restriction de liens contrôle la façon dont les jetons dans les places de liens peuvent être utilisés, considérons la sixième ligne de l'exécution du scénario 2 présenté figure 8.15 :

$$(pb^+; (((\overline{pb^+} \circledast f) \| (\overline{cb^-} \circledast f)) \,\mathsf{tie}\, b); cb^-.b) \,\mathsf{tie}\, b \quad .$$

Remarquons que si nous pouvions déplacer le $.b$ depuis la fin de l'expression jusque sous la barre de $\overline{cb^-}$, nous pourrions exécuter cb^- (par la règle LA3). Cependant, la règle B1 interdit cela, ce qui est complètement cohérent avec le box correspondant (donné

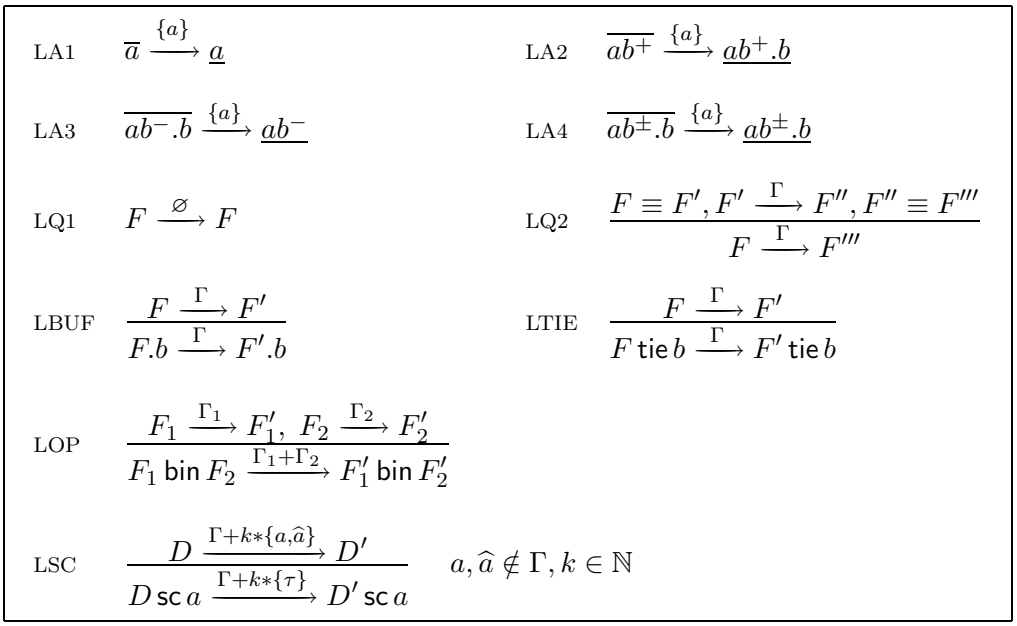


Figure 8.14 — Les règles pour la sémantique opérationnelle basée sur les étiquettes. Avec $a \in \mathbb{A}_\tau$ (sauf pour la règle LSC où $a \neq \tau$), $b \in \mathbb{B}$ et bin un opérateur binaire de ABC.

figure 8.5) qui, après avoir franchi sa transition étiquetée p tout en haut, ne peut pas immédiatement exécuter celle étiquetée c car elle a en entrée la place de liens interne qui est encore vide.

8.6 Généralisations

Nous présentons dans cette section une généralisation du modèle ABC qui autorise une plus grande variété d'opérateurs. Nous avons choisi de présenter cette extension à part pour permettre à l'intuition de se fixer sur ABC (qui est un modèle assez simple mais déjà complètement utilisable) ; une bonne compréhension des principes présentés jusque-là facilite grandement celle du modèle plus général que nous introduisons maintenant. Cette généralisation permet de plus d'éviter de prouver séparément les propriétés attachées à chaque opérateur de ABC, et aussi de préparer de futures extensions de ce modèle, en particulier l'ajout de l'opérateur de récursion.

8.6.1 Boxes opérateurs de séquencement et d'interface

Pour commencer, nous allons considérer que les opérateurs $\Omega_\Box$, $\Omega_\circledast$ et $\Omega_;$ sont des cas particuliers d'une classe plus générale de boxes opérateurs. Un *box opérateur de séquencement* Ω_{sq} est un box opérateur n'ayant aucune place isolée, possédant exactement

$$
\begin{array}{lll}
\overline{(pb^+;(((pb^+ \circledast f)\|(cb^- \circledast f))\,\mathsf{tie}\,b);cb^-)\,\mathsf{tie}\,b} & & \\
\quad\equiv & (\overline{pb^+};(((pb^+ \circledast f)\|(cb^- \circledast f))\,\mathsf{tie}\,b);cb^-)\,\mathsf{tie}\,b & \text{E1, IS1} \\
\xrightarrow{\{p\}} & (\underline{pb^+.b};(((pb^+ \circledast f)\|(cb^- \circledast f))\,\mathsf{tie}\,b);cb^-)\,\mathsf{tie}\,b & \text{LA1, LOP, LTIE} \\
\quad\equiv & (\underline{pb^+};(((pb^+ \circledast f)\|(cb^- \circledast f))\,\mathsf{tie}\,b);cb^-.b)\,\mathsf{tie}\,b & \text{X1, OPL-R} \\
\quad\equiv & (pb^+;\overline{(((pb^+ \circledast f)\|(cb^- \circledast f))\,\mathsf{tie}\,b);cb^-.b)\,\mathsf{tie}\,b} & \text{IS1-2, E1} \\
\quad\equiv & (pb^+;(((\overline{pb^+} \circledast f)\|(\overline{cb^-} \circledast f))\,\mathsf{tie}\,b);cb^-.b)\,\mathsf{tie}\,b & \text{IPAR1, IIT1} \\
\xrightarrow{\{p\}} & (pb^+;(((\underline{pb^+.b} \circledast f)\|(\overline{cb^-} \circledast f))\,\mathsf{tie}\,b);cb^-.b)\,\mathsf{tie}\,b & \text{LA1, LOP, LTIE} \\
\quad\equiv & (pb^+;(((\overline{pb^+}.b \circledast f)\|(\overline{cb^-} \circledast f))\,\mathsf{tie}\,b);cb^-.b)\,\mathsf{tie}\,b & \text{IIT2, E1} \\
\xrightarrow{\{p\}} & (pb^+;(((\underline{pb^+.b}.b \circledast f)\|(\overline{cb^-} \circledast f))\,\mathsf{tie}\,b);cb^-.b)\,\mathsf{tie}\,b & \text{LA1, LOP, LTIE} \\
\quad\equiv & (pb^+;(((\underline{pb^+.b} \circledast f)\|(\overline{cb^-.b} \circledast f))\,\mathsf{tie}\,b);cb^-)\,\mathsf{tie}\,b & \text{OPL-R, E1} \\
\xrightarrow{\{c\}} & (pb^+;(((\underline{pb^+.b} \circledast f)\|(\underline{cb^-} \circledast f))\,\mathsf{tie}\,b);cb^-.b)\,\mathsf{tie}\,b & \text{LA3, LOP, LTIE} \\
\quad\equiv & (pb^+;(((pb^+.b \circledast \overline{f})\|(cb^- \circledast \overline{f}))\,\mathsf{tie}\,b);cb^-.b)\,\mathsf{tie}\,b & \text{IIT3} \\
\xrightarrow{\{f,f\}} & (pb^+;(((pb^+.b \circledast \underline{f})\|(cb^- \circledast \underline{f}))\,\mathsf{tie}\,b);cb^-.b)\,\mathsf{tie}\,b & \text{LA1, LOP, LTIE} \\
\quad\equiv & (pb^+;(((pb^+.b \circledast f)\|(cb^- \circledast f))\,\mathsf{tie}\,b);\overline{cb^-.b})\,\mathsf{tie}\,b & \text{IIT4, IPAR2, X1, IS2} \\
\xrightarrow{\{c\}} & (pb^+;(((pb^+.b \circledast f)\|(cb^- \circledast f))\,\mathsf{tie}\,b);\underline{cb^-})\,\mathsf{tie}\,b & \text{LA3, LOP, LTIE} \\
\quad\equiv & \underline{(pb^+;(((pb^+.b \circledast f)\|(cb^- \circledast f))\,\mathsf{tie}\,b);cb^-)\,\mathsf{tie}\,b} & \text{IS3, X1}
\end{array}
$$

Figure 8.15 — L'exécution du scénario 2.

une place d'entrée et une place de sortie et tel que, pour chaque transition $v \in T_{\Omega_{sq}}$, $|{}^\bullet v| = |v^\bullet| = 1$ et $\lambda_{\Omega_{sq}}(v) = \varphi_{id}$. Cela revient à dire que Ω_{sq} peut être vu comme un *automate fini* dont chacune des transitions sera remplacée par un box potentiellement complexe, grâce à la substitution de transitions. Nous supposons que toutes les transitions de Ω_{sq} ont des identités distinctes prises dans η, et que deux boxes opérateurs distincts ont des ensembles de nœuds disjoints. Le domaine d'application de Ω_{sq} est $\mathsf{dom}(\Omega_{sq})$ qui comprend tous les Ω_{sq}-uplets de boxes dynamiques et statiques dont au plus un est dynamique.

Comme exemple d'opérateur de séquencement, nous pouvons définir une itération ternaire similaire à celle présentée pour les M-nets (voir la section 3.3.2, en particulier, le réseau $N_{[**]}$ de la figure 3.8 page 58).

Nous pouvons aussi considérer le scoping comme une instance d'une classe plus générale d'opérateurs agissant sur les actions. Un *box opérateur d'interface de communication* Ω_φ (voir figure 8.17) est un opérateur unaire paramétré par une fonction d'interface

$$
\begin{array}{lll}
\overline{(((pb^+ \| pb^+) \circledast f) \| (cb^- \circledast f))\,\mathsf{tie}\,b} & & \\
\quad \equiv & (((\overline{pb^+} \| \overline{pb^+}) \circledast f) \| (\overline{cb^- \circledast f}))\,\mathsf{tie}\,b & \text{E1, IPAR1, IIT1} \\
\xrightarrow{\{p\}} & (((\underline{pb^+.b} \| \overline{pb^+}) \circledast f) \| (\overline{cb^- \circledast f}))\,\mathsf{tie}\,b & \text{LA2} \\
\quad \equiv & (((\underline{pb^+} \| \overline{pb^+}) \circledast f) \| (\overline{cb^-.b} \circledast f))\,\mathsf{tie}\,b & \text{X1, OPL, OPR, E1} \\
\xrightarrow{\{p,c\}} & (((\underline{pb^+} \| \underline{pb^+.b}) \circledast f) \| (\underline{cb^-} \circledast f))\,\mathsf{tie}\,b & \text{LA2, LA3} \\
\quad \equiv & (((pb^+ \| pb^+.b) \circledast \overline{f}) \| (cb^- \circledast \overline{f}))\,\mathsf{tie}\,b & \text{IPAR2, IIT3} \\
\xrightarrow{\{f,f\}} & (((pb^+ \| pb^+.b) \circledast \underline{f}) \| (cb^- \circledast \underline{f}))\,\mathsf{tie}\,b & \text{LA1} \\
\quad \equiv & \underline{(((pb^+ \| pb^+.b) \circledast f) \| (cb^- \circledast f))\,\mathsf{tie}\,b} & \text{IIT4, IPAR2, X1}
\end{array}
$$

Figure 8.16 — L'exécution du scénario 3.

$\varphi : \mathsf{mult}(\mathbb{A}_\tau) \setminus \{\varnothing\} \to \mathbb{A}_\tau$, qui a le domaine d'application $\mathsf{dom}(\Omega_\varphi) \stackrel{\text{df}}{=} \mathsf{abox}$. Afin d'assurer que nous créons toujours des réseaux finis, nous supposons qu'il n'existe pas d'ensemble fini $A \subseteq \mathbb{A}_\tau$ tel que $\mathsf{mult}(A) \cap \mathsf{dom}(\Omega_\varphi)$ est infini. Nous pouvons montrer que $\Omega_{\mathsf{sc}\,a}$ vérifie cette contrainte. Le rôle de Ω_φ sera d'effectuer les changements au niveau des interfaces de communication de ses opérandes conformément à ce que φ spécifie.

Nous pouvons par exemple considérer un scoping regroupant toutes les transitions portant une même action (au lieu des paires de transitions portant des actions conjuguées), il faudrait pour cela prendre la fonction d'interface φ_a sur le domaine $(\mathsf{mult}(\{a\}) \cup \{\{a'\} \mid a' \in \mathbb{A}_\tau\}) \setminus \{\varnothing\}$ qui associe x à tout singleton $\{x\} \subset \mathbb{A}_\tau \setminus \{a\}$ et renvoie τ sinon.

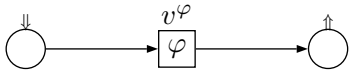


Figure 8.17 — Le box opérateur Ω_φ.

Remarquons qu'il n'est pas nécessaire d'étendre l'opérateur de composition parallèle pour le rendre n-aire. En effet, nous pouvons le définir plus simplement comme : $\|_n(N_1, \ldots, N_n) \stackrel{\text{df}}{=} (\cdots (N_1 \| \cdots) \| N_n)$.

Dans la suite nous faisons référence aux opérateurs de séquencement ou d'interface de communication ainsi qu'à la composition parallèle ou à la restriction de liens en tant qu'*opérateurs généraux*.

8.6.2 Substitution de transitions

La substitution de transitions définies à la section 8.3.2 pour les opérateurs de ABC peut être facilement étendue à des opérateurs d'arité arbitraire.

Opérateurs de séquencement. Soit Ω un opérateur de séquencement dont les transitions sont v_1, …, v_n, et soit $\mathbf{N} = (N_1, \ldots, N_n) = (N_{v_1}, \ldots, N_{v_n})$ un Ω-uplet de boxes dans $\mathsf{dom}(\Omega)$. Alors $\Omega(\mathbf{N}) = N'$ dont les éléments sont définis ci-dessous.

L'ensemble des transitions de N' est $\{v^i \lhd\!\lhd T_{N_i} \mid i \in \{1, \ldots, n\}\}$. L'étiquette de chaque $v_i \lhd t$ est celle de t. Chaque place $p \in S_{N_i}$ de statut i ou b appartient aussi à $S_{N'}$, son statut et son marquage sont conservés et pour chaque transition $w \lhd t$, la fonction de poids est définie par :

$$W_{N'}(p, w \lhd t) \stackrel{\text{df}}{=} \begin{cases} W_{N_i}(p,t) & \text{si } w = v_i, \\ 0 & \text{sinon}\,; \end{cases}$$

et de manière similaire pour $W_{N'}(w \lhd t, p)$.

Pour chaque place $s \in S_\Omega$, avec ${}^\bullet s = \{u_1, \ldots, u_k\}$ et $s^\bullet = \{w_1, \ldots, w_m\}$, nous construisons dans $S_{N'}$ toutes les places de la forme $p \stackrel{\text{df}}{=} (x_1, \ldots, x_k, e_1, \ldots, e_m)$, où chaque x_i est une place de sortie de N_{u_i} et chaque e_j est une place d'entrée de N_{w_j}. Dans les preuves, nous notons $\langle\!\langle s \rangle\!\rangle$ l'ensemble des places p ainsi construites, le sous-ensemble de $\langle\!\langle s \rangle\!\rangle$ faisant intervenir une place q est noté $\langle\!\langle s, q \rangle\!\rangle$; de même, le sous-ensemble de $\langle\!\langle s, q \rangle\!\rangle$ faisant intervenir une place q' est noté $\langle\!\langle s, q, q' \rangle\!\rangle$; et ainsi de suite pour un nombre quelconque de places. L'étiquette de chaque p est celle de s, son marquage est la somme des marquages de x_1, …, x_k, e_1, …, e_m, et, pour toute transition $w \lhd t$, la fonction de poids est donnée par :

$$W_{N'}(p, w \lhd t) \stackrel{\text{df}}{=} \begin{cases} W_{N_w}(x_i,t) + W_{N_w}(e_j,t) & \text{si } w \in {}^\bullet s \cap s^\bullet \text{ et } w = u_i = w_j, \\ W_{N_w}(x_i,t) & \text{si } w \in {}^\bullet s \setminus s^\bullet \text{ et } w = u_i, \\ W_{N_w}(e_j,t) & \text{si } w \in s^\bullet \setminus {}^\bullet s \text{ et } w = w_j, \\ 0 & \text{sinon}\,; \end{cases}$$

et de manière similaire pour $W_{N'}(w \lhd t, p)$.

Pour chaque $b \in \mathbb{B}$, nous ajoutons à $S_{N'}$ une place $p^b \stackrel{\text{df}}{=} (p^b_{v_1}, \ldots, p^b_{v_n})$ de statut b, où chaque $p^b_{v_i}$ est l'unique place de statut b dans N_i. Le marquage de p^b est la somme des marquages des $p^b_{v_i}$ et, pour chaque transition $w \lhd t$, la fonction de poids est donnée par :

$$W_{N'}(p^b, w \lhd t) \stackrel{\text{df}}{=} W_{N_w}(p^b_w, t) \quad ,$$

et de manière similaire pour $W_{N'}(w \lhd t, p^b)$.

Opérateurs d'interface. Pour un opérateur d'interface Ω_φ, lorsque le domaine de φ contient un multi-ensemble Γ, nous appliquons un changement d'interface à tout ensemble de transitions dont les étiquettes correspondent à Γ : ces transitions vont être synchronisées pour produire une nouvelle transition d'étiquette $\varphi(\Gamma)$. (Remarquons que puisque les opérateurs de séquencement et la composition parallèle utilisent la fonction d'interface φ_{id}, aucune transition n'est changée pour eux.)

L'application de l'opérateur Ω_φ à un box N donne un box similaire à N mais dont l'ensemble de transitions comprend tous les arbres $t \stackrel{\text{df}}{=} v^\varphi \lhd \{t_1, \ldots, t_l\}$ tels que $\{t_1, \ldots, t_l\} \in$

$\mathsf{mult}(T_N)$ et $\Lambda \stackrel{\text{df}}{=} \{\lambda_N(t_1), \ldots, \lambda_N(t_l)\}$ est dans le domaine de φ. L'étiquette de chaque t est $\varphi(\Lambda)$ et, pour chaque place $p \in S_{\Omega_\varphi(N)}$, la fonction de poids est donnée par :

$$W_{\Omega_\varphi(N)}(p,t) \stackrel{\text{df}}{=} \sum_{i=1}^{l} W_N(p,t_i) \quad ,$$

et de manière similaire pour $W_{\Omega_\varphi(N)}(t,p)$.

Propriétés. Nous donnons maintenant des résultats généraux sur la substitution de transitions.

Proposition 29.
Soit Ω l'opérateur de composition parallèle $\Omega_\|$ ou un opérateur de séquencement. Soient encore $\mathbf{N}$ un Ω-uplet de boxes, N un box arbitraire, φ une fonction d'interface, $b \in \mathbb{B}$ et $B \in \mathsf{mult}(\mathbb{B})$.

1. *$M^{\mathsf{c}}_{\Omega(\mathbf{N})} = \varnothing$ ssi $M^{\mathsf{c}}_{N_{v_i}} = \varnothing$ pour chaque $v_i \in T_\Omega$.*
2. *S'il existe N' égal à $\Omega_\varphi(N)$, $N \,\mathsf{tie}\, b$ ou $N.B$, alors $M^{\mathsf{c}}_{N'} = \varnothing$ ssi $M^{\mathsf{c}}_N = \varnothing$.*

Preuve. (1) Découle des définitions, en particulier de la T-restriction de Ω et de l'ex-restriction de chaque box N_{v_i}. En effet, grâce à ces contraintes, pour chaque $e \in N^{\mathsf{e}}_{v_i}$, si $s \in {}^\bullet v_i$ alors $\langle\!\langle s,e \rangle\!\rangle \neq \varnothing$ et chaque $p \in \langle\!\langle s,e \rangle\!\rangle$ est une place de contrôle dans $\Omega(\mathbf{N})$ et $M_{N_{v_i}}(e) \leq M_{\Omega(\mathbf{N})}(p)$. La situation est similaire pour $x \in N^{\mathsf{x}}_{v_i}$. Finalement, les places internes de N_{v_i} ne sont pas changées.
(2) Conséquence directe du fait que les places de contrôle ne sont pas changées. □

Lemme 30.
Si M est un marquage sain d'un box N tel que $M(e) + M(x) \geq 1$ pour toutes les places $e \in N^{\mathsf{e}}$ et $x \in N^{\mathsf{x}}$, alors $M^{\mathsf{c}} \in \{N^{\mathsf{e}}, N^{\mathsf{x}}\}$.

Preuve. Supposons par l'absurde que $M^{\mathsf{c}} \notin \{N^{\mathsf{e}}, N^{\mathsf{x}}\}$. Alors, puisque M est sain, il existe $e \in N^{\mathsf{e}}$ et $x \in N^{\mathsf{x}}$ tels que $M(e) = M(x) = 0$, ce qui contredit l'hypothèse $M(e) + M(x) \geq 1$. □

Lemme 31.
Soit $\Omega(\mathbf{N})$ une application correcte d'un opérateur de séquencement Ω, et soit une transition $z \in T_\Omega$ telle que $N = N_z$ est un box dynamique. De plus, soient s'' une place de S_Ω, $M \stackrel{\text{df}}{=} M_{\Omega(\mathbf{N})}$, $\{s\} \stackrel{\text{df}}{=} {}^\bullet z$ et $\{s'\} \stackrel{\text{df}}{=} z^\bullet$

1. *S'il existe une place marquée dans $\langle\!\langle s'' \rangle\!\rangle$ alors $s'' \in \{s, s'\}$.*
2. *Si $M^{\mathsf{c}} = N^{\mathsf{e}}$ alors $M^{\mathsf{c}} = \langle\!\langle s \rangle\!\rangle$. Si $M^{\mathsf{c}} = N^{\mathsf{x}}$ alors $M^{\mathsf{c}} = \langle\!\langle s' \rangle\!\rangle$.*
3. *Si $M^{\mathsf{c}} \geq \langle\!\langle s'' \rangle\!\rangle$ alors $M^{\mathsf{c}} = \langle\!\langle s'' \rangle\!\rangle$ et l'une des propriétés suivantes est vérifiée :*
   - *$s'' = s \neq s'$ et $M^{\mathsf{c}}_N = N^{\mathsf{e}}$.*
   - *$s'' = s' \neq s$ et $M^{\mathsf{c}}_N = N^{\mathsf{x}}$.*

- $s'' = s' = s$ *et* $M_N^{\mathsf{c}} \in \{N^{\mathsf{e}}, N^{\mathsf{x}}\}$.

Preuve. (1) Conséquence de la définition de $\Omega(\mathbf{N})$ et du fait que toutes les places de contrôle dans les réseaux N_v, pour $v \neq z$, sont vides.

(2) Supposons $M^{\mathsf{c}} = N^{\mathsf{e}}$ (le cas $M^{\mathsf{c}} = N^{\mathsf{x}}$ est symétrique). Alors, les places de contrôle marquées de $\Omega(\mathbf{N})$ sont celles dans la construction desquelles les places de N^{e} ont été utilisées. De plus, $M(q) = M_N(e)$ pour toutes places $e \in N^{\mathsf{e}}$ et $q \in \langle\!\langle s, e \rangle\!\rangle$. D'où le résultat.

(3) Le point (1) nous donne que $s'' \in \{s, s'\}$. Supposons d'abord que $s'' = s' \neq s$ (le cas $s'' = s \neq s'$ est symétrique). Nous avons alors $M(q) = M_N(e)$ pour toutes places $e \in N^{\mathsf{e}}$ et $q \in \langle\!\langle s'', e \rangle\!\rangle$, et donc, $M_N \geq N^{\mathsf{e}}$. Comme M est sain, nous avons alors $M_N = N^{\mathsf{e}}$. De plus, par le point (2), nous avons $M^{\mathsf{c}} = \langle\!\langle s \rangle\!\rangle$.
Supposons maintenant que $s'' = s' = s$, Alors nous avons $1 \leq M(q) = M_N(e) + M_N(x)$ pour toutes places $e \in N^{\mathsf{e}}$, $x \in N^{\mathsf{x}}$ et $q \in \langle\!\langle s'', e, x \rangle\!\rangle$. D'où, par le lemme 30, $M_N^{\mathsf{c}} \in \{N^{\mathsf{e}}, N^{\mathsf{x}}\}$. De plus, par le point (2), nous obtenons $M^{\mathsf{c}} = \langle\!\langle s'' \rangle\!\rangle$. □

Théorème 32.
Soit Ω un box opérateur général et soit $\mathbf{N} \in \mathsf{dom}(\Omega)$. Alors, $\Omega(\mathbf{N})$ est un box donc le marquage est sain et ac-libre. De plus, si tous les boxes dynamiques de $\mathbf{N}$ (s'il y en a) ont un marquage quasi-sûr, alors le marquage de $\Omega(\mathbf{N})$ est quasi-sûr.

Preuve. La preuve est immédiate pour les opérateurs qui ne sont pas des opérateurs de séquencement. Supposons donc que Ω est un box opérateur de séquencement.

Alors, $\Omega(\mathbf{N})$ est :

- ex-restreint puisque $\langle\!\langle s \rangle\!\rangle = \varnothing$ pour toute place $s \in S_\Omega$;
- B-restreint par définition ;
- T-restreint puisque tous les boxes dans $\mathbf{N}$ le sont et que $\langle\!\langle s, p \rangle\!\rangle = \varnothing$ dès que $s \in S_\Omega$ et $p \in N_v^{\mathsf{e}}$ pour $v \in s^\bullet$ (ou $p \in N_v^{\mathsf{x}}$ pour $v \in {}^\bullet s$).

Le fait que $M_{\Omega(\mathbf{N})}$ est sain vient du lemme 31(3), en prenant $\{s\} = \Omega^{\mathsf{e}}$ ou $\{s\} = \Omega^{\mathsf{x}}$.

Nous prouvons maintenant que $M_{\Omega(\mathbf{N})}$ est ac-libre, et, le cas échéant, quasi-sûr. Si les boxes de $\mathbf{N}$ sont tous statiques, alors, par définition, toutes les places de contrôle de $\Omega(\mathbf{N})$ sont vides ; la propriété découle alors immédiatement de la T-restriction de $\Omega(\mathbf{N})$. Supposons donc que N_w est l'unique box dynamique de $\mathbf{N}$ et prenons une transition $v \triangleleft t$ dans $T_{\Omega(\mathbf{N})}$. Nous pouvons alors considérer deux cas.

Cas 1 : $v \neq w$. Si ${}^\bullet t$ contient une place interne p, alors p est aussi une place interne de $\Omega(\mathbf{N})$ avec $p \in {}^\bullet(v \triangleleft t)$ et $M_{\Omega(\mathbf{N})}(p) = M_{N_w}(p) = 0$. Sinon, ${}^\bullet t$ contient une place $p \in N_v^{\mathsf{e}} \cup N_v^{\mathsf{x}}$. Supposons que $p \in N_v^{\mathsf{e}}$ (le cas $p \in N_v^{\mathsf{x}}$ étant symétrique) et que ${}^\bullet v = \{s\}$, alors, $\langle\!\langle s, p \rangle\!\rangle \subseteq {}^\bullet(v \triangleleft t)$. Nous pouvons alors considérer les sous-cas suivants :

- Si $w \notin {}^\bullet s \cup s^\bullet$, autrement dit, si $s \notin {}^\bullet w \cup w^\bullet$ alors, par définition de $\Omega(\mathbf{N})$, pour tout $q \in \langle\!\langle s, p \rangle\!\rangle$, nous avons $M_{\Omega(\mathbf{N})}(q) = 0$. D'où le résultat.
- Si $w \in {}^\bullet s \setminus s^\bullet$ alors, puisque N_w est sain, ou bien $M_{N_w}^{\mathsf{c}} = N_w^{\mathsf{x}}$, ou bien il existe $x \in N_w^{\mathsf{x}}$ tel que $M_{N_w}(x) = 0$. Dans le premier cas, pour tout $q \in \langle\!\langle s, p \rangle\!\rangle$, nous avons $M_{\Omega(\mathbf{N})}(q) = 1$. En revanche, dans le second cas, pour tout $q \in \langle\!\langle s, p, x \rangle\!\rangle$, nous avons $M_{\Omega(\mathbf{N})}(q) = 0$. D'où le résultat.

- Si $w \in s^\bullet \setminus {}^\bullet s$ alors, puisque N_w est sain, ou bien $M^{\mathsf{c}}_{N_w} = N^{\mathsf{e}}_w$, ou bien il existe $e \in N^{\mathsf{e}}_w$ tel que $M_{N_w}(e) = 0$. Dans le premier cas, pour tout $q \in \langle\!\langle s, p \rangle\!\rangle$, nous avons $M_{\Omega(\mathbf{N})}(q) = 1$. En revanche, dans le second cas, pour tout $q \in \langle\!\langle s, p, e \rangle\!\rangle$, nous avons $M_{\Omega(\mathbf{N})}(q) = 0$. D'où le résultat.
- Si $w \in {}^\bullet s \cap s^\bullet$ alors, puisque N_w est sain, ou bien $M^{\mathsf{c}}_{N_w} \in \{N^{\mathsf{e}}_w, N^{\mathsf{x}}_w\}$, ou bien il existe $e \in N^{\mathsf{e}}_w$ et $x \in N^{\mathsf{x}}_w$ tels que $M_{N_w}(e) = M_{N_w}(x) = 0$. Dans le premier cas, pour tout $q \in \langle\!\langle s, p \rangle\!\rangle$, nous avons $M_{\Omega(\mathbf{N})}(q) = 1$. En revanche, dans le second cas, pour tout $q \in \langle\!\langle s, p, e, x \rangle\!\rangle$, nous avons $M_{\Omega(\mathbf{N})}(q) = 0$. D'où le résultat.

Cas 2 : $v = w$. Comme N_w est un box ac-libre, il possède une place de contrôle p telle que $M_{N_w}(p) < 2\, W_{N_w}(p, t)$. Cela signifie en particulier que $W_{N_w}(p, t) \geq 1$ et donc que $p \in {}^\bullet t$. Si de plus N_w est quasi-sûr, nous pouvons choisir p de telle façon que $M_{N_w}(p) \leq 1$.

Si p est une place interne de N_w, alors c'est aussi une place interne de $\Omega(\mathbf{N})$ avec $p \in {}^\bullet(w \triangleleft t)$ et :

$$M_{\Omega(\mathbf{N})}(p) = M_{N_w} < 2\, W_{N_w}(p, t) = 2\, W_{\Omega(\mathbf{N})}(p, w \triangleleft t) \quad .$$

De plus, $M_{\Omega(\mathbf{N})}(p) = M_{N_v}(p) \leq 1$ dans le marquage quasi-sûr. D'où le résultat.

Si maintenant $p \in N^{\mathsf{e}}_w$ (le cas $p \in N^{\mathsf{x}}_w$ est symétrique), soit s une place de Ω telle que $\{s\} = {}^\bullet w$. Nous pouvons alors considérer deux sous-cas :

- Si ${}^\bullet w \neq \{s\}$ alors, pour tout $q \in \langle\!\langle s, p \rangle\!\rangle$,

  $$M_{\Omega(\mathbf{N})}(q) = M_{N_w}(p) < 2\, W_{N_w}(p, t) = 2\, W_{\Omega(\mathbf{N})}(q, w \triangleleft t) \quad .$$

  De plus, $M_{\Omega(\mathbf{N})}(q) = M_{N_w}(p) \leq 1$ dans le marquage quasi-sûr. D'où le résultat.
- Si ${}^\bullet w = \{s\} = w^\bullet$ alors, puisque N_w est sain, ou bien $M^{\mathsf{c}}_{N_w} = N^{\mathsf{x}}_w$, ou bien il existe $x \in N^{\mathsf{x}}_w$ tel que $M_{N_w}(x) = 0$. Dans le premier cas, pour tout $q \in \langle\!\langle s, p \rangle\!\rangle$, nous avons $M_{\Omega(\mathbf{N})}(q) = 1$. En revanche, dans le second cas, pour tout $q \in \langle\!\langle s, p, x \rangle\!\rangle$, nous avons

  $$M_{\Omega(\mathbf{N})}(q) = M_{N_w}(p) < 2\, W_{N_w}(p, t) = 2\, W_{\Omega(\mathbf{N})}(q, w \triangleleft t) \quad .$$

  De plus, $M_{\Omega(\mathbf{N})}(q) = M_{N_w}(p) \leq 1$ dans le marquage quasi-sûr. D'où le résultat.

□

Proposition 33.
Soit Ω un box opérateur de séquencement, et soit $\mathbf{N}$ un Ω-uplet de boxes statiques.

1. *Si la transitions $v \in T_\Omega$ est telle que $\Omega^{\mathsf{e}} = {}^\bullet v$ (respectivement, telle que $\Omega^{\mathsf{e}} = v^\bullet$), alors $\overline{\Omega(\mathbf{N})} = \Omega(\mathbf{N}')$ où $\mathbf{N}'$ est $\mathbf{N}$ dans lequel N_v a été remplacé par $\overline{N_v}$ (respectivement, par $\underline{N_v}$).*

2. *Si la transition $v \in T_\Omega$ est telle que $\Omega^{\mathsf{x}} = {}^\bullet v$ (respectivement, telle que $\Omega^{\mathsf{x}} = v^\bullet$), alors $\underline{\Omega(\mathbf{N})} = \Omega(\mathbf{N}')$ où $\mathbf{N}'$ est $\mathbf{N}$ dans lequel N_v a été remplacé par $\overline{N_v}$ (respectivement, par $\underline{N_v}$).*

Preuve. Par définition. □

8.6.3 Équivalences structurelles générales

Soit Ω un box opérateur général et soient $\mathbf{N}$ et $\mathbf{N}'$ deux Ω-uplet de boxes. Nous définissons six relations auxiliaires de la façon suivante :

- $\mathbf{N} \equiv^0_\Omega \mathbf{N}'$ s'il existe une transition $v \in T_\Omega$ et un box N'' tels que ${}^\bullet v = v^\bullet$, $\{N_v, N'_v\} = \{\overline{N''}, \underline{N''}\}$ et $N_u = N'_u$ pour tout $u \in T_\Omega \setminus \{v\}$.
- $\mathbf{N} \equiv^1_\Omega \mathbf{N}'$ s'il existe deux transitions $v \neq w \in T_\Omega$ et deux boxes N''_v et N''_w tels que ${}^\bullet v = {}^\bullet w$, $\{(N_v, N_w), (N'_v, N'_w)\} = \{(\overline{N''_v}, N''_w), (N''_v, \underline{N''_w})\}$ et $N_u = N'_u$ pour tout $u \in T_\Omega \setminus \{v, w\}$.
- $\mathbf{N} \equiv^2_\Omega \mathbf{N}'$ s'il existe deux transitions $v \neq w \in T_\Omega$ et deux boxes N''_v et N''_w tels que $v^\bullet = w^\bullet$, $\{(N_v, N_w), (N'_v, N'_w)\} = \{(\underline{N''_v}, N''_w), (N''_v, \overline{N''_w})\}$ et $N_u = N'_u$ pour tout $u \in T_\Omega \setminus \{v, w\}$.
- $\mathbf{N} \equiv^3_\Omega \mathbf{N}'$ s'il existe deux transitions $v \neq w \in T_\Omega$ et deux boxes N''_v et N''_w tels que $v^\bullet = {}^\bullet w$, $\{(N_v, N_w), (N'_v, N'_w)\} = \{(\underline{N''_v}, N''_w), (N''_v, \underline{N''_w})\}$ et $N_u = N'_u$ pour tout $u \in T_\Omega \setminus \{v, w\}$.
- $\mathbf{N} \equiv^4_\Omega \mathbf{N}'$ si, pour tout $v \in T_\Omega$, il existe un box N''_v et deux multi-ensembles B_v et B'_v sur $\mathbb{B}$ tels que :
$$\sum_{v \in T_\Omega} B_v = \sum_{v \in T_\Omega} B'_v \quad ,$$
et, pour tout $v \in T_\Omega$, $N_v = N''_v.B$ et $N'_v = N''_v.B'$.
- $\equiv^5_\Omega \stackrel{\text{df}}{=} id_{\mathsf{abox}}$, où id_{abox} est l'identité sur les boxes.

Nous définissons ensuite

$$\equiv_\Omega \stackrel{\text{df}}{=} \equiv^4_\Omega \circ \bigcup_{i \in \{0,1,2,3,5\}} \equiv^i_\Omega \quad .$$

Remarquons que $\equiv_{\Omega_\parallel} = \equiv^4_\Omega$ et que, pour tout opérateur général unaire, $\equiv_\Omega = id_{\mathsf{abox}}$.

Proposition 34.
Soit Ω un box opérateur général, et soit $\mathbf{N} \in \mathsf{dom}(\Omega)$.

1. *Si $\mathbf{N} \equiv_\Omega \mathbf{N}'$ pour un uplet $\mathbf{N}'$, alors $\mathbf{N}' \in \mathsf{dom}(\Omega)$ et $[\![\mathbf{N}]\!] = [\![\mathbf{N}']\!]$.*
2. *$\equiv_\Omega$ est une équivalence sur $\mathsf{dom}(\Omega)$.*
3. *S'il existe $\mathbf{N}' \in \mathsf{dom}(\Omega)$ tel que $[\![\mathbf{N}]\!] = [\![\mathbf{N}']\!]$, alors $\Omega(\mathbf{N}) = \Omega(\mathbf{N})$ ssi $\mathbf{N} \equiv_\Omega \mathbf{N}'$.*

Preuve. (1) Le résultat est évident pour les boxes opérateurs unaires. Pour la composition parallèle, il provient de l'observation que $\equiv_{\Omega_\parallel} = \equiv^4_\Omega$ qui ne modifie que le marquage des places de liens ouvertes, et donc, ne change pas la propriété d'être un box statique ou dynamique (voir aussi la proposition 10(1)). Pour un opérateur de séquencement Ω, il nous suffit d'observer que les relations $\equiv^i_\Omega$ agissent uniquement sur les marquages des composants et qu'elles préservent le nombre relatif de boxes statiques et dynamiques dans $\mathbf{N}$. Dans le cas de $\equiv^4_\Omega$, le résultat vient de la proposition 10(1). Les autres cas se montrent séparément. Par exemple, considérons la relation

$\equiv^1_\Omega$, avec $(N_v, N_w) = (\overline{N''_v}, N''_w)$ et $(N'_v, N'_w) = (N''_v, \overline{N''_w})$. Alors, par définition de $\mathsf{dom}(\Omega)$,

$(N_v, N_w) \in \mathsf{abox}_d \times \mathsf{abox}_s$. Et donc, par la proposition 10(2), nous obtenons $N''_v \in \mathsf{abox}_s$ et ainsi $(N'_v, N'_w) \in \mathsf{abox}_s \times \mathsf{abox}_d$. D'où $\mathbf{N}' \in \mathsf{dom}(\Omega)$.

(2) Le résultat est à nouveau évident pour les boxes opérateurs unaires. Pour le box opérateur parallèle, nous vérifions facilement que $\equiv_{\Omega_\parallel} = \equiv^{\Omega}_4$ est une relation réflexive, symétrique et transitive. Considérons maintenant un opérateur de séquencement Ω. En prenant $B_v = B'_v$ pour tout $v \in T_\Omega$, nous avons $id_{\mathsf{abox}} \subseteq \equiv^4_\Omega$ et donc, la relation $\equiv_\Omega$ est réflexive puisque $id_{\mathsf{abox}} \subseteq (\equiv^4_\Omega \circ \equiv^5_\Omega)$. De plus, il est facile de voir que chaque $\equiv^i_\Omega$ est symétrique et que $(\equiv^4_\Omega \circ \equiv^i_\Omega) = (\equiv^i_\Omega \circ \equiv^4_\Omega)$, pour tout $i \in \{0,1,2,3,5\}$. Et donc, $\equiv_\Omega$ est aussi symétrique. Elle est enfin transitive sur $\mathsf{dom}(\Omega)$, en effet, pour toute paire $(i,j) \in \{0,1,2,3,5\}^2$, en se restreignant à $\mathsf{dom}(\Omega)$ pour la seconde propriété, nous avons :

$$(\equiv^4_\Omega \circ \equiv^i_\Omega \circ \equiv^4_\Omega \circ \equiv^j_\Omega) = (\equiv^4_\Omega \circ \equiv^4_\Omega \circ \equiv^i_\Omega \circ \equiv^j_\Omega) = (\equiv^4_\Omega \circ \equiv^i_\Omega \circ \equiv^j_\Omega) \quad ,$$
$$(\equiv^i_\Omega \circ \equiv^j_\Omega) \subseteq \bigcup_{k \in \{0,1,2,3,5\}} \equiv^k_\Omega \quad .$$

La première propriété est directe, la seconde se montre en analysant les différents cas, et en particulier en utilisant le fait que chaque transition de Ω a seulement une place en entrée et une en sortie et que, pour tout $\mathbf{N} \in \mathsf{dom}(\Omega)$, au plus un composant de $\mathbf{N}$ est dynamique. Par exemple, nous pouvons montrer que :

$$(\equiv^1_\Omega \circ \equiv^0_\Omega) \subseteq \equiv^3_\Omega \quad , \qquad (\equiv^1_\Omega \circ \equiv^1_\Omega \subseteq \equiv^1_\Omega \cup \equiv^5_\Omega) \quad , \qquad (\equiv^1_\Omega \circ \equiv^2_\Omega) = \varnothing \quad ,$$
$$(\equiv^1_\Omega \circ \equiv^3_\Omega) \subseteq \equiv^3_\Omega \quad , \qquad (\equiv^1_\Omega \circ \equiv^5_\Omega) = \equiv^1_\Omega \quad .$$

(3) Comme $[\![\mathbf{N}]\!] = [\![\mathbf{N}']\!]$, $\mathbf{N}$ et $\mathbf{N}'$ ne peuvent différer que par leurs marquages. Pour un box opérateur d'interface, la propriété est évidente puisque qu'il ne modifie que les transitions (et les arcs attachés) d'une façon qui ne dépend que de leurs étiquettes, mais pas des marquages. Pour l'opérateur de restriction de liens, la propriété est encore évidente puisque qu'il ne modifie que le statut de la même place dans $\mathbf{N} = N$ et $\mathbf{N}' = N'$, et ajoute à chacun d'eux une nouvelle place avec le même marquage vide. Pour l'opérateur de composition parallèle, la propriété résulte du fait que les marquages de places de liens fermées sont inchangés, alors que celui des places de liens ouvertes, qui sont fusionnées, est modifié d'une façon qui est exactement préservée par $\equiv_{\Omega_\parallel} = \equiv^4_\Omega$. Considérons pour finir un box opérateur de séquencement Ω. Alors, $[\![\Omega(\mathbf{N})]\!] = [\![\Omega(\mathbf{N}')]\!]$ puisque $[\![\mathbf{N}]\!] = [\![\mathbf{N}']\!]$. De plus, par définition de la substitution de transitions, nous avons

$$M^{\mathsf{f}}_{\Omega(\mathbf{N})} = M^{\mathsf{f}}_{\Omega(\mathbf{N}')} \iff \sum_{v \in T_\Omega} M^{\mathsf{f}}_{N_v} = \sum_{v \in T_\Omega} M^{\mathsf{f}}_{N'_v} \quad ,$$

où nous identifions le marquage des places de liens ouvertes avec le multi-ensemble de symboles de liens qu'il représente, ce qui correspond exactement à la propriété préservée par $\equiv^4_\Omega$. Ajoutons que les places de liens fermées (et leurs marquages) sont exactement le mêmes dans $\mathbf{N}$ et $\Omega(\mathbf{N}')$ (par définition), de même que dans $\mathbf{N}'$ et $\Omega(\mathbf{N}')$, et qu'elles ne sont pas concernées par $\equiv_\Omega$. Ainsi, nous pouvons nous limiter aux places de contrôle et considérer chaque sens de l'équivalence.

($\Leftarrow$) Si tous les boxes de $\mathbf{N}$ sont statiques, alors $\mathbf{N} \equiv^4_\Omega \mathbf{N}'$ et nous avons clairement $\Omega(\mathbf{N}) = \Omega(\mathbf{N}')$. Il reste donc à montrer que, pour tout $i \in \{0,1,2,3,5\}$, $\mathbf{N} \equiv^i_\Omega \mathbf{N}'$ implique que $M^{\mathsf{c}}_{\Omega(\mathbf{N})} = M^{\mathsf{c}}_{\Omega(\mathbf{N}')}$. Pour cela, considérons chaque valeur de i séparément :

- $\mathbf{N} \equiv^0_\Omega \mathbf{N}'$. Supposons que v et N'' sont tels que dans la définition de $\equiv^0_\Omega$. Sans perte de généralité, supposons que $N_v = \overline{N''}$ et $N'_v = \underline{N''}$. Alors, puisque N_v et N'_v sont sains, nous avons $M_{N_v} = N_v^{\mathsf{e}}$ et $M_{N'_v} = N_v'^{\mathsf{x}}$. D'où, par le lemme 30(2), $M^{\mathsf{c}}_{\Omega(\mathbf{N})} = \langle\!\langle s \rangle\!\rangle = M_{\Omega(\mathbf{N}')}$, où $\{s\} = {}^\bullet v = v^\bullet$, et donc le résultat.
- $\mathbf{N} \equiv^1_\Omega \mathbf{N}'$. Supposons que v, w, N''_v et N''_w sont tels que dans la définition de $\equiv^1_\Omega$. Sans perte de généralité, nous pouvons supposer que $N_v = \overline{N''_v}$ et $N'_w = \underline{N''_w}$. Alors, puisque N_v et N'_v sont sains, nous avons $M_{N_v} = N_v^{\mathsf{e}}$ et $M_{N'_v} = N_v'^{\mathsf{e}}$. D'où, par le lemme 30(2), $M^{\mathsf{c}}_{\Omega(\mathbf{N})} = \langle\!\langle s \rangle\!\rangle M_{\Omega(\mathbf{N}')}$, où $\{s\} = {}^\bullet v = {}^\bullet w$, et donc le résultat.
- Les cas $\mathbf{N} \equiv^2_\Omega \mathbf{N}'$ et $\mathbf{N} \equiv^3_\Omega \mathbf{N}'$ sont similaires au précédent.
- $\mathbf{N} \equiv^5_\Omega \mathbf{N}'$. Nous avons alors $\mathbf{N} = \mathbf{N}'$ et donc le résultat.

($\Rightarrow$) Si $\mathbf{N}$ est composé de boxes statiques uniquement, alors $M^{\mathsf{c}}_{\Omega(\mathbf{N})} = \varnothing = M^{\mathsf{c}}_{\Omega(\mathbf{N}')}$ et donc $\mathbf{N}'$ est aussi un Ω-uplet de boxes statiques. En effet, s'il existe une place interne marquée dans un N'_v, alors elle est aussi marquée dans $\Omega(\mathbf{N}') = \Omega(\mathbf{N})$ ce qui contredit l'hypothèse que $\mathbf{N}$ n'a que des boxes statiques. De même, s'il existe une place d'entrée e marquée dans un N'_v, en posant ${}^\bullet v = \{s\}$, alors toutes les places de $\langle\!\langle s, e \rangle\!\rangle$ sont aussi marquées dans $\Omega(\mathbf{N}') = \Omega(\mathbf{N})$, ce qui est encore impossible. Nous montrons de même qu'il ne peut y avoir de place de sortie marquée dans aucun N'_v. Donc, dans ce cas, nous avons $\mathbf{N} \equiv^4_\Omega \mathbf{N}'$ et donc $\mathbf{N} \equiv_\Omega \mathbf{N}'$ ce qui donne le résultat.

Supposons maintenant que $\mathbf{N}$ comporte un box dynamique N_v, avec ${}^\bullet v = \{s\}$ et $v^\bullet = \{s'\}$. Pour toute place interne p de chaque N_z, par définition, p est aussi une place de $\Omega(\mathbf{N}) = \Omega(\mathbf{N}')$ et $M_{N_z}(p) = M_{\Omega(\mathbf{N})}(p) = M_{\Omega(\mathbf{N}')}(p) = M_{N'_z}(p)$. Ensuite, pour les places de n'importe quel $\langle\!\langle s'' \rangle\!\rangle$, où $s'' \in S_\Omega$, c'est-à-dire celles construites à partir des places d'interface des boxes de $\mathbf{N}$ et $\mathbf{N}'$, nous considérons trois cas.

Cas 1 : $M^{\mathsf{c}}_{N_v} \notin \{N_v^{\mathsf{e}}, N_v^{\mathsf{x}}\}$. Puisque N_v est sain, il existe $e \in N_v^{\mathsf{e}}$ et $x \in N_v^{\mathsf{x}}$ tels que $M_{N_v}(e) = 0 = M_{N_v}(x)$. Donc, pour chaque place s'' de Ω, et pour chaque place q appartenant à $\langle\!\langle s'' \rangle\!\rangle$ si $s'' \notin \{s, s'\}$, $\langle\!\langle s'', e \rangle\!\rangle$ si $s'' = s \neq s'$, $\langle\!\langle s'', x \rangle\!\rangle$ si $s'' = s' \neq s$ et $\langle\!\langle s'', e, x \rangle\!\rangle$ si $s'' = s' = s$, nous obtenons que $M_{\Omega(\mathbf{N})}(q) = 0 = M_{\Omega(\mathbf{N}')}(q)$ et donc que toutes les places de $\mathbf{N}'$ utilisées dans la construction de q ont le marquage vide. En particulier, c'est le cas de e, x et de toutes les places d'interface de N'_w, pour $w \neq v$. De plus, cela implique que pour chaque place d'entrée $e' \neq e$ de N'_v, remplacer e par e' dans une place $q \in \langle\!\langle s, e \rangle\!\rangle$ ou $q \in \langle\!\langle s, e, x \rangle\!\rangle$ donne que $M_{\Omega(\mathbf{N})}(q) = M_{N_v}(e') = M_{\Omega(\mathbf{N}')}(q) = M_{N'_v}(e')$; et de même pour une place de sortie $x' \neq x$ de N'_v. D'où $\mathbf{N} \equiv^4_\Omega \mathbf{N}'$ et la propriété.

Cas 2 : $M^{\mathsf{c}}_{N_v} = N_v^{\mathsf{e}}$. Par le lemme 30(2), pour tout $q \in \langle\!\langle s \rangle\!\rangle$, nous avons $M_{\Omega(\mathbf{N})}(q) = 1 = M_{\Omega(\mathbf{N}')}(q)$. Choisissons l'une de ces places q : il doit exister dans N'_v exactement une place d'interface utilisée dans la construction de q marquée d'un jeton, toutes les autres doivent être vides. Supposons que cette place est une place x de $N_w'^{\mathsf{x}}$ et qu'alors $M_{N'_w}(x) = 1$ et $w^\bullet = \{s\}$ (le cas d'une place e de $N_w'^{\mathsf{e}}$ avec $w^\bullet = \{s\}$ est similaire). En
160 remplaçant l'occurrence de x dans q par une autre place $x' \in N_w'^{\mathsf{x}}$, nous obtenons une

autre place de $\langle\!\langle s \rangle\!\rangle$ avec le marquage 1 qui provient nécessairement de x'. Puisque N'_v est sain, nous avons nécessairement $M_{N'_w} = {N'_w}^{\mathsf{x}}$. De plus, N'_z doit être statique pour tout $z \in T_\Omega \setminus \{w\}$. Nous observons alors que si $v = w$, on a $\mathbf{N}(\equiv^4_\Omega \circ \equiv^0_\Omega)\mathbf{N}'$; et si $v \neq w$, alors $\mathbf{N}(\equiv^4_\Omega \circ \equiv^3_\Omega)\mathbf{N}'$. D'où la propriété.

Cas 3 : $M^{\mathsf{c}}_{N_v} = N^{\mathsf{x}}_v$. Ce cas est similaire au précédent. □

Proposition 35.
Soit Ω un box opérateur de séquencement d'arité n et soit $\mathbf{N} \in \mathsf{dom}(\Omega)$.

1. *Si $N_i[U_i\rangle N'_i$ (pour $i \leq n$) alors $\mathbf{N}' \in \mathsf{dom}(\Omega)$ et*
$$\Omega(\mathbf{N})[(v_1 \ll U_1) \cup \cdots \cup (v_n \ll U_n)\rangle \Omega(\mathbf{N}') \quad .$$
2. *Si $\Omega(\mathbf{N})[U\rangle\mathbf{N}'$, alors, il existe des uplets $\mathbf{N}''$ et $\mathbf{N}'''$ dans $\mathsf{dom}(\Omega)$ et des steps U_1, …, U_n tels que $\mathbf{N} \equiv_\Omega \mathbf{N}''$, $N''_i[U_i\rangle N'''_i$ (pour $i \leq n$), $\mathbf{N}' = \Omega(\mathbf{N}''')$ et*
$$U = (v_1 \ll U_1) \cup \cdots \cup (v_n \ll U_n) \quad .$$

Remarquons qu'en conséquence, $\mathsf{enabled}(\Omega(\mathbf{N}))$ comprend exactement tous les ensembles de transitions $(v_1 \ll U_1) \cup \cdots \cup (v_n \ll U_n)$ tels que $\mathbf{N}'' \equiv_\Omega \mathbf{N}$ et $U_i \in \mathsf{enabled}(N''_i)$ (pour $i \leq n$). Notons aussi qu'au plus un des U_i est non vide à cause du domaine d'application de Ω.

Preuve. Dans cette preuve, nous notons s_b l'unique place de statut $b \in \mathbb{B}$ d'un réseau.

(1) Comme $N_i[U_i\rangle N'_i$ (pour $i \leq n$), chaque N'_i est statique, respectivement dynamique, si N_i l'est aussi. Nous avons donc $\mathbf{N}' \in \mathsf{dom}(\Omega)$ comme $\mathbf{N} \in \mathsf{dom}(\Omega)$. Si les boxes de $\mathbf{N}$ sont tous statiques, alors chaque U_j est vide, $\mathbf{N} = \mathbf{N}'$ et $\Omega(\mathbf{N})[\varnothing\rangle\Omega(\mathbf{N}')$. Sinon, $\mathbf{N}$ a exactement un composant dynamique N_i. Nous observons alors que, pour tout $j \neq i$, $U_j = \varnothing$ et $N'_j = N_j$. De plus, les propriétés suivantes sont vérifiées :

- pour chaque place p de N_i de statut i ou b :
$$\begin{array}{rcccccl} M_{\Omega(\mathbf{N})}(p) = M_{N_i}(p) & \geq & \sum_{t\in U_i} W_{N_i}(p,t) & = & \sum_{t\in U_i} W_{\Omega(\mathbf{N})}(p, v_i \lhd t) & \text{et} \\ M_{N_i}(p) & \geq & \sum_{t\in U_i} W_{N_i}(t,p) & = & \sum_{t\in U_i} W_{\Omega(\mathbf{N})}(v_i \lhd t, p) & . \end{array}$$
- Pour tout $b \in \mathbb{B}$, nous avons :
$$\begin{array}{rcccccl} M_{\Omega(\mathbf{N})}(s_b) \geq M_{N_i}(s_b) & \geq & \sum_{t\in U_i} W_{N_i}(s_b,t) & = & \sum_{t\in U_i} W_{\Omega(\mathbf{N})}(s_b, v_i \lhd t) & \text{et} \\ M_{N_i}(s_b) & \geq & \sum_{t\in U_i} W_{N_i}(t,s_b) & = & \sum_{t\in U_i} W_{\Omega(\mathbf{N})}(v_i \lhd t, s_b) & . \end{array}$$
- Pour toute place $s \in S_\Omega$ et toute place $q = (e_1, \ldots, e_k, x_1, \ldots, x_m) \in \langle\!\langle s \rangle\!\rangle$, nous avons (avec $P \stackrel{\text{df}}{=} \{e_1, \ldots, e_k, x_1, \ldots, x_m\} \cap S_{N_i}$) :
$$\begin{array}{rcccccl} M_{\Omega(\mathbf{N})}(q) = \sum_{p\in P} M_{N_i}(p) & \geq & \sum_{p\in P}\sum_{t\in U_i} W_{N_i}(p,t) & = & \sum_{t\in U_i} W_{\Omega(\mathbf{N})}(q, v_i \lhd t) & \text{et} \\ M_{\Omega(\mathbf{N})}(q) & \geq & \sum_{p\in P}\sum_{t\in U_i} W_{N_i}(t,p) & = & \sum_{t\in U_i} W_{\Omega(\mathbf{N})}(v_i \lhd t, q) & . \end{array}$$

D'où $\Omega(\mathbf{N})[(v_1 \lll U_1) \cup \cdots \cup (v_k \lll U_k)\rangle \Omega(\mathbf{N}')$.

(2) Si $\Omega(\mathbf{N})[U\rangle \mathbf{N}'$, alors il existe des steps U_1, ..., U_n tels que $U = (v_1 \lll U_1) \cup \cdots \cup (v_n \lll U_n)$. Si $U = \varnothing$, chaque U_i est vide aussi et la propriété est évidente en prenant $\mathbf{N}'' = \mathbf{N}''' = \mathbf{N}$. C'est le cas, en particulier, lorsque $\mathbf{N}$ n'a que des boxes statiques. Sinon, supposons que N_i est le box dynamique de $\mathbf{N}$ et considérons deux cas.

Cas 1 : $M^{\mathsf{c}}_{n_i} \notin \{N_i^{\mathsf{e}}, N_i^{\mathsf{x}}\}$. Puisque N_i est sain, il existe $e \in N_i^{\mathsf{e}}$ et $x \in N_i^{\mathsf{x}}$ tels que $M_{N_i}(e) = 0 = M_{N_i}(x)$. Pour tout $j \neq i$ et $t \in T_{N_j}$, les conditions suivantes sont alors vérifiées :

- si q est une place interne dans ${}^\bullet t$, alors $q \in {}^\bullet(v_j \lhd t)$ et $M_{\Omega(\mathbf{N})}(q) = M_{N_i}(q) = 0$;
- si q est une place d'entrée dans ${}^\bullet t$, avec ${}^\bullet v_j = \{s\}$ (ou symétriquement si q est une place de sortie dans ${}^\bullet t$ avec $v_j{}^\bullet = \{s\}$), alors, pour tout $q \in \langle\!\langle s, p \rangle\!\rangle$ si ${}^\bullet v_i \neq \{s\} \neq v_i{}^\bullet$, $q \in \langle\!\langle s, p, e \rangle\!\rangle$ si ${}^\bullet v_i = \{s\} \neq v_i{}^\bullet$, $q \in \langle\!\langle s, p, x \rangle\!\rangle$ si ${}^\bullet v_i \neq \{s\} = v_i{}^\bullet$ ou $q \in \langle\!\langle s, p, e, x \rangle\!\rangle$ si ${}^\bullet v_i = \{s\} = v_i{}^\bullet$, nous avons $q \in {}^\bullet(v_j \lhd t)$ et $M_{\Omega(\mathbf{N})}(q) = 0$.

D'où $U_j = \varnothing$ pour tout $j \neq i$.

Soit maintenant N_i'' égal à N_i avec le changement suivant : pour chaque $b \in \mathbb{B}$, $M_{N_i''}(s_b) = M_{\Omega(\mathbf{N})}(s_b)$. De plus, pour tout $j \neq i$, soit N_j'' égal à N_j dont le marquage des places de liens ouvertes a été supprimé. Nous avons alors $\mathbf{N} \equiv_\Omega \mathbf{N}''$ ainsi que les propriétés suivantes :

- pour toute place p de $S_{N_i''}$ de statut i ou b :

$$M_{N_i''}(p) = M_{N_i}(p) = M_{\Omega(\mathbf{N})}(p) \geq \sum_{t \in U_i} W_{\Omega(\mathbf{N})}(p, v_i \lhd t) = \sum_{t \in U_i} W_{N_i''}(p, t) \quad ;$$

- pour $b \in \mathbb{B}$:

$$M_{N_i''}(s_b) = M_{\Omega(\mathbf{N})}(s_b) \geq \sum_{t \in U_i} W_{\Omega(\mathbf{N})}(s_b, v_i \lhd t) = \sum_{t \in U_i} W_{N_i''}(s_b, t) \quad ;$$

- pour toute place d'entrée p de $S_{N_i''}$ (la situation est symétrique si p est une place de sortie) et pour tout $q \in \langle\!\langle s, p \rangle\!\rangle$, si ${}^\bullet v_i = \{s\} \neq v_i{}^\bullet$:

$$M_{N_i''}(p) = M_{N_i}(p) = M_{\Omega(\mathbf{N})}(q) \geq \sum_{t \in U_i} W_{\Omega(\mathbf{N})}(q, v_i \lhd t) = \sum_{t \in U_i} W_{N_i''}(p, t) \quad ;$$

et pour tout $q \in \langle\!\langle s, p, x \rangle\!\rangle$, si ${}^\bullet v_i = \{s\} = v_i{}^\bullet$:

$$\begin{aligned} M_{N_i''}(p) = M_{N_i}(p) = M_{\Omega(\mathbf{N})}(q) &\geq \sum_{t \in U_i} W_{\Omega(\mathbf{N})}(q, v_i \lhd t) \\ &= \sum_{t \in U_i} (W_{N_i''}(p, t) + W_{N_i''}(x, t)) \\ &\geq \sum_{t \in U_i} W_{N_i''}(p, t) \quad . \end{aligned}$$

D'où $N_i''[U_i\rangle N_i'''$ et $N_j''[U_j\rangle N_j'''$ pour tout $j \neq i$. Ainsi, par la première partie de la proposition, nous obtenons $\mathbf{N}' = \Omega(\mathbf{N}''')$.

Cas 2 : $M^{\mathsf{c}}_{N_i} = N_i^{\mathsf{e}}$ (le cas $M^{\mathsf{c}}_{N_i} = N_i^{\mathsf{x}}$ est similaire). Supposons que ${}^\bullet v_i = \{s\}$. Alors, par le lemme 31(2), nous avons $M_{\Omega(\mathbf{N})} = \langle\!\langle s \rangle\!\rangle$. D'où, pour tout $(v \triangleleft t) \in U$: ${}^\bullet t$ ne contient aucune place interne et, si ${}^\bullet t \cap N_v^{\mathsf{e}} \neq \varnothing$ alors ${}^\bullet v = \{s\}$, de plus, ${}^\bullet t \cap N_v^{\mathsf{x}} \neq \varnothing$ alors $v^\bullet = \{s\}$.

Supposons par l'absurde que $\{v \triangleleft t, v' \triangleleft t'\} \in U$, avec $v \neq v'$. Supposons aussi qu'il existe $e \in {}^\bullet t \cap N_v^{\mathsf{e}}$ et $e' \in {}^\bullet t' \cap N_{v'}^{\mathsf{e}}$. Alors, pour tout $q \in \langle\!\langle s, e, e' \rangle\!\rangle$:

$$1 = M_{\Omega(\mathbf{N})}(q) \geq W_{\Omega(\mathbf{N})}(q, v \triangleleft t) + W_{\Omega(\mathbf{N})}(q, v' \triangleleft t') = W_{N_v}(e, t) + W_{N'_v}(e', t') \geq 2 \quad .$$

Ce qui est contradictoire (la situation est similaire si e est remplacé par un $x \in {}^\bullet t \cap N_v^{\mathsf{x}}$, ou si e' est remplacé par $x' \in {}^\bullet t' \cap N_{v'}^{\mathsf{x}}$). En conséquence, il existe un unique step U_j non vide. Considérons alors deux sous-cas.

Cas 2.a : $j = i$. Remarquons d'abord qu'il n'est pas possible d'avoir t et t' dans U_i (y compris si $t = t'$) avec $e \in {}^\bullet t \cap N_i^{\mathsf{e}}$ et $x \in {}^\bullet t' \cap N_i^{\mathsf{x}}$ car ou bien $v_i{}^\bullet = \{s'\} \neq \{s\}$ et alors, pour $q \in \langle\!\langle s', x \rangle\!\rangle$,

$$M_{\Omega(\mathbf{N})}(q) = 0 < 1 \leq W_{N_i}(x, t') = W_{\Omega(\mathbf{N})}(q, v_i \triangleleft t') \quad ,$$

ou bien $v_i{}^\bullet = \{s\}$ et alors, pour $q \in \langle\!\langle s, e, x \rangle\!\rangle$,

$$M_{\Omega(\mathbf{N})}(q) = 1 < 2 \leq W_{N_i}(e, t) + W_{N_i}(x, t') = W_{\Omega(\mathbf{N})}(q, v_i \triangleleft t) + W_{\Omega(\mathbf{N})}(q, v_i \triangleleft t') \quad .$$

Donc, les places de contrôle en entrée des transitions de U_i sont soit toutes dans N_i^{e} soit toutes dans N_i^{x}.

Dans le premier cas, nous ne pouvons pas avoir $t \neq t' \in U_i$ pour un $e \in {}^\bullet t \cap {}^\bullet t' \cap N_i^{\mathsf{e}}$ car sinon, pour tout $q \in \langle\!\langle s, e \rangle\!\rangle$, nous aurions :

$$M_{\Omega(\mathbf{N})}(q) = 1 < 2 \leq W_{N_i}(e, t) + W_{N_i}(e, t') = W_{\Omega(\mathbf{N})}(q, v_i \triangleleft t) + W_{\Omega(\mathbf{N})}(q, v_i \triangleleft t') \quad .$$

Maintenant, pour tout $e \in N_i^{\mathsf{e}}$ et $q \in \langle\!\langle s, e \rangle\!\rangle$:

$$M_{N_i}(e) = M_{\Omega(\mathbf{N})}(q) \geq \sum_{t \in U_i} W_{\Omega(\mathbf{N})}(q, v_i \triangleleft t) = \sum_{t \in U_i} W_{N_i}(e, t) \quad .$$

De plus, pour toute place de liens fermée $p \in S_{N_i}$:

$$M_{N_i}(p) = M_{\Omega(\mathbf{N})}(p) \geq \sum_{t \in U_i} W_{\Omega(\mathbf{N})}(p, v_i \triangleleft t) = \sum_{t \in U_i} W_{N_i}(p, t) \quad .$$

Nous définissons N_i'' comme N_i avec un seul changement : pour chaque $b \in \mathbb{B}$, $M_{N_i''}(s_b) = M_{\Omega(\mathbf{N})}(s_b)$. De plus, pour tout $m \neq i$, soit N_m'' égal à N_m dans lequel nous supprimons tout marquage des places de liens ouvertes. Alors, pour tout $b \in \mathbb{B}$:

$$M_{N_i''}(s_b) = M_{\Omega(\mathbf{N})}(s_b) \geq \sum_{t \in U_i} W_{\Omega(\mathbf{N})}(s_b, v_i \triangleleft t) = \sum_{t \in U_i} W_{N_i''}(s_b, t) \quad .$$

D'où $N_i''[U_i\rangle N_i'''$ et $N_m''[U_m\rangle N_m'''$, pour tout $m \neq i$, avec $\mathbf{N}'' \equiv_\Omega \mathbf{N}$. Et donc, par la première partie de la proposition : $\mathbf{N}' = \Omega(\mathbf{N}''')$.

Dans le second cas (si ${}^\bullet U_i \subseteq N_i^\times$), nous devons avoir ${}^\bullet v_i = \{s\} = v_i{}^\bullet$ et nous pouvons remplacer dans $\mathbf{N}$ les jetons des places d'entrée de N_i par un jeton dans chaque place de sortie de N_i. Cela donne un Ω-uplet d'arguments équivalent (par définition de $\equiv_\Omega^0$) auquel nous pouvons appliquer le raisonnement ci-dessus, avec $N_{N_i}^{\mathsf{c}} = N_i^\times$.

Cas 2.b : $j \neq i$. Ou bien ${}^\bullet v_j = \{s\}$ auquel cas nous remplaçons $\mathbf{N}$ par $\mathbf{N''}$ avec pour seule modification le remplacement de $M_{N_i''}^{\mathsf{c}}$ et $M_{N_j''}^{\mathsf{c}}$ par le marquage vide et $N_j''^{\mathsf{e}}$ respectivement (voir la définition de $\equiv_\Omega^1$). Ou bien nous avons $v_j{}^\bullet = \{s\}$ et alors nous remplaçons $\mathbf{N}$ par $\mathbf{N''}$ avec pour seule modification le remplacement de $M_{N_j''}^{\mathsf{c}}$ par le marquage vide et celui de $M_{N_j''}^{\mathsf{c}}$ par $N_j''^{\times}$ (voir la définition de $\equiv_\Omega^3$). Dans les deux cas, nous retrouvons la situation précédente avec $\mathbf{N} \equiv_\Omega \mathbf{N''}$. □

Proposition 36.
Soient N un box statique ou dynamique et Ω_φ un box opérateur d'interface de communication.

1. *Si $N[U_1 \uplus \cdots \uplus U_k\rangle N'$ et que chaque $\lambda_N(U_i)$ est dans le domaine de φ, alors*

$$\Omega_\varphi(N)[v^\varphi \lll \{U_1, \ldots, U_k\}\rangle \Omega_\varphi(N') \quad .$$

2. *Si $\Omega_\varphi(N)[U\rangle N''$, alors, il existe un box N' et des steps U_1, …, U_k tels que $N'' = \Omega_\varphi(N')$, $U = v^\varphi \lll \{U_1, \ldots, U_k\}$ et $N[U_1 \uplus \cdots \uplus U_k\rangle N'$.*

Preuve. Par définition. □

Remarquons que, en conséquence, $\mathsf{enabled}(\Omega_\varphi(N))$ comprend exactement tous les $U = v^\varphi \lll \{U_1, \ldots, U_k\}$ tels que $(U_1 \uplus \cdots \uplus U_k) \in \mathsf{enabled}(N)$ et chaque $\lambda_N(U_i)$ est dans le domaine de φ.

9 Conclusion et perspectives

En introduction nous posions la question de savoir dans quelle mesure un modèle purement causal, comme les réseaux de Petri, peut permettre de traiter des problèmes temps-réel.

Tout d'abord, nous avons adapté la notion de *temps causal* à l'algèbre des M-nets : nous avons modélisé dans cette algèbre plusieurs horloges et montré comment les utiliser pour temporiser des systèmes. Nous avons aussi montré comment représenter des systèmes à plusieurs horloges (c'est souvent le cas des systèmes répartis) et comment nous pouvions synchroniser ces horloges. Ensuite, nous avons montré que, sous réserve d'éliminer les comportements infinis utilisant des ressources finies (en temps, en espace ou en puissance de calcul), nous pouvions exécuter un système temporisé avec une horloge causale sous des contraintes temps-réel, c'est-à-dire en faisant en sorte que les tics de l'horloge causale surviennent de façon régulière. Comme application concrète, nous avons étudié un système de passage à niveau et constaté les forces et les limites de notre approche : la temporisation par horloge causale permet de traiter des problèmes de façon simple et efficace, mais les performances en pratique sont encore dépendantes de la taille des constantes temporelles utilisées. Finalement, nous avons conjecturé qu'une classe raisonnablement large d'automates temporisés devraient être représentables par des M-nets avec horloge causale, en assurant une bisimilarité des comportements.

Pour certains problèmes temps-réel, la temporisation seule peut être insuffisante et la préemption doit être utilisée. Nous avons donc défini une extension de l'algèbre des M-nets qui propose des opérateurs permettant de modéliser la préemption. Cette extension a été faite en introduisant une notion de priorités entre les événements, ce qui nous a fait sortir du cadre purement causal fixé jusque-là. Nous avons montré que le modèle ainsi obtenu avait le même pouvoir d'expression que le modèle de départ (qui est celui des réseaux de Petri places/transitions dans les cas finis et des machines de Turing dans les cas infinis). Ce résultat indique que les priorités ne sont pas strictement nécessaires. Cependant, nous avons aussi montré que nous obtenions en les utilisant une abréviation (de rapport exponentiel) de la taille des réseaux de Petri à utiliser. Cette même réduction a été obtenue dans une autre approche de la préemption dans le cadre des réseaux de Petri [Kis+97] (il s'agit à notre connaissance du seul modèle de ce genre à part celui que

nous avons proposé). À la lumière de ces constats, nous pensons que si un mécanisme supplémentaire à la causalité n'est pas nécessaire à la représentation de la préemption, il est par contre nécessaire à sa modélisation structurée. En effet, dans le modèle de [Kis+97] comme dans notre approche, les réseaux de Petri obtenus sont structurés et la préemption agit sur des sous-réseaux clairement identifiables. Dans les deux cas, les transformations permettent de se ramener à la stricte causalité en éliminant complètement cet aspect structuré.

L'introduction de la préemption a été appliquée à la sémantique d'un langage de programmation parallèle dans lequel nous avons ajouté un mécanisme d'exceptions et un système de tâches. Nous y avons aussi introduit des temporisations en utilisant une des horloges causales que nous avons défini. Nous avons donné la sémantique de ces extensions par une fonction sémantique dirigée par la syntaxe, ce qui correspond à la méthode utilisée pour le reste du langage.

Nous avons étendu l'algèbre des M-nets avec des *liens asynchrones* permettant d'utiliser certaines horloges causales en respectant mieux les aspects composables de notre modèle. Cette extension a été utilisée pour améliorer la sémantique d'un langage de programmation parallèle. En effet, nous avons proposée une notation permettant de définir la sémantique de façon beaucoup plus lisible ; cette notation a été utilisée pour réduire considérablement la taille de la sémantique des canaux de communication. Nous avons aussi étudié en détails les conséquences de l'introduction des liens asynchrones dans le cadre du modèle servant de fondement théorique aux M-nets.

Un contexte purement causal nous semble donc adapté à la modélisation structurée de la temporisation des systèmes concurrents. Nos résultats indiquent qu'il n'est pas nécessaire de sortir de la stricte causalité pour définir un modèle temporisé. Cela peut même être fait en utilisant un modèle composable à la manière des algèbres de processus.

En ce qui concerne la préemption, la causalité seule nous semble insuffisante pour obtenir une représentation structurée des systèmes préemptibles. Cela nécessite en effet une vision globale des parties à interrompre ce qui entre en contradiction avec la localité sous-entendue par la concurrence dans un cadre causal. Cependant, nous avons montré comment nous ramener à un cadre causal dans le cas de systèmes finis mais au détriment de la structure et de l'aspect composable des systèmes.

Travaux en cours et perspectives. De nombreux travaux sont envisagés à la suite de cette thèse, certains ayant déjà commencé.

En premier lieu, il nous semble important d'établir la validité de la conjecture que nous avons formé à propos des automates temporisés. En effet, nous avons une forte intuition qu'il est possible de représenter une large classe d'automates avec des M-nets munis d'une horloge causale. Nos résultats expérimentaux viennent aussi à l'appui de cette intuition. Il nous faut donc déterminer avec exactitude quelle classe nous pouvons représenter et définir comment un automate peut être simulé par son équivalent M-net et comment l'inverse est aussi possible. Nous avons déjà observé qu'il est nécessaire dans certains cas d'avoir plusieurs compteurs d'horloge causale pour simuler un seul

chronomètre d'automate. Il nous faut par ailleurs établir une correspondance précise entre le temps d'horloge des M-nets et l'axe de temps continu utilisé par les automates temporisés.

Dans le même domaine, nous pensons dans un futur proche nous intéresser au problème d'explosion de l'espace des états lors de la vérification des M-nets avec horloges causales. Des travaux récents semblent pouvoir aider dans cette direction puisqu'ils proposent des moyens d'obtenir des réseaux d'occurrences généralisés de haut niveau (ces réseaux permettant une représentation particulièrement efficace de l'espace des états d'un réseau de Petri en tirant profit de sa sémantique par ordre partiel). Nous pensons aussi pouvoir nous inspirer de l'approche par graphe de régions utilisée dans les automates temporisés.

Toujours en ce qui concerne les applications à la modélisation et la vérification de systèmes, nous avons commencé une comparaison avec les principaux modèles de réseaux de Petri intégrant du temps. Les expérimentations dans cette direction sont pour le moment suspendues en attendant que les outils dont nous avons besoin soient mis à jour (les développements sont en cours au moment où nous écrivons ces lignes). Par ailleurs, sur un plan plus formel, une comparaison de l'expressivité des différents modèles a été commencée et nous avons déjà pu établir quelques transformations entre certains modèles à temps explicite et le modèle du temps causal.

Afin d'exploiter pleinement le temps causal dans le domaine de la vérification, il nous semble nécessaire de définir des logiques temporelles temporisées le prenant en compte. Les premières réflexions que nous avons menées sur le sujet semblent indiquer la possibilité de définir des abréviations des logiques temporelles déjà utilisées. En effet, le temps causal étant représenté par une horloge explicitement modélisée, ces logiques donnent déjà le moyen de s'exprimer à son sujet mais d'une manière extrêmement inconfortable puisqu'il faut faire référence à l'horloge et ses états. Nous pensons qu'il est possible de prendre en compte l'existence d'une telle horloge au niveau d'une logique qui serait ensuite traduite vers une logique temporelle déjà utilisée. Cette approche se base sur le principe même qui nous a guidé en introduisant le temps causal : préserver l'existant.

Au niveau des applications à la sémantique des langages de programmation, plusieurs thèmes sont envisagés. En particulier, il nous semble important de renforcer le lien entre le temps causal et la préemption pour définir des constructions liant les deux (comme des chiens de garde, des échéances avec avortement, etc.). Il nous faut aussi grandement améliorer la sémantique des procédures que nous avons dû considérablement simplifier pour contourner les difficultés liées à l'introduction de la préemption. Tous ces travaux pourraient être unifiés dans le cadre de travaux sur la sémantique du langage Ada. Nous pensons que nos méthodes peuvent s'appliquer à ce langage et que les travaux envisagés ci-dessus peuvent se faire dans ce cadre.

Concernant la préemption, nous envisageons une étude poussée de ce qu'elle implique au niveau de la causalité et de la structure des systèmes. Nous avons expliqué qu'il existe une contradiction entre ces aspects ; il nous faut cependant étudier précisément cette question. Nous souhaitons formaliser les notions de préemption, de concurrence et de composabilité ; si un cadre peut être défini pour représenter ces trois principes, il devrait être possible d'y faire apparaître ou bien qu'il existe des contradictions intrinsèques, ou

bien qu'il est possible de les unifier en choisissant une autre approche. Notre intuition nous guide vers la première possibilité mais cela reste entièrement à préciser. Cette étude sera peut-être menée dans le cadre d'un projet franco-canadien soumis dont le sujet porte sur la sémantique concurrente de la préemption.

Finalement, cette thèse montre comment introduire le temps dans un modèle composable mais il nous manque la *compositionnalité* de la sémantique associée. Nous avons présenté un premier pas dans cette direction avec l'étude théorique d'une version étendue du modèle sur lequel les M-nets sont basés. Nous avons enrichi ce modèle de bas niveau avec des liens asynchrones et montré qu'on pouvait toujours lui associer une sémantique compositionnelle. Le second pas sera basé sur ce résultat : nous pensons pouvoir définir une syntaxe d'expressions dans le haut niveau (les M-nets) et lui donner une sémantique opérationnelle. En étendant aux expressions la notion de dépliage, permettant de passer des réseaux de Petri de haut niveau aux réseaux de bas niveau, il devrait être possible d'hériter de toutes les propriétés prouvées dans le bas niveau. Ce point devrait faire l'objet de travaux dans les mois qui viennent. Un troisième pas sera alors d'étendre la syntaxe pour y intégrer le temps et de baser la sémantique réseau sur l'utilisation des horloges causales. On obtiendrait alors un modèle concurrent et compositionnel pour les systèmes temporisés. Finalement, nous envisageons de définir une sémantique opérationnelle pour les opérateurs que nous avons introduit pour la préemption. En l'intégrant soigneusement au cadre déjà obtenu, nous devrions aboutir à un modèle compositionnel (et concurrent), prenant en compte les principaux aspects du temps-réel, et sur lequel on pourrait appliquer les nombreuses techniques de vérification issues des réseaux de Petri.

Le domaine des M-nets comporte encore d'autres branches d'activités, en particulier des travaux en cours sur la mobilité (avec une sémantique concurrente de π-calcul). Nous espérons que les résultats dans ce domaine seront compatibles avec ceux présentés dans ce mémoire, ce qui nous permettrait d'envisager l'intégration de la mobilité au modèle du temps-réel que nous souhaitons développer.

Bibliographie

Mes publications

[1] H. Klaudel et F. Pommereau. *Asynchronous links in the PBC and M-nets.* ASIAN'99, LNCS 1742. Springer, 1999.

[2] F. Pommereau. *FIFO buffers in tie sauce.* DAPSYS'00. Kluwer Academic Publishers, 2000.

[3] H. Klaudel et F. Pommereau. *A concurrent and compositional Petri net semantics of preemption.* IFM'00, LNCS 1945. Springer, 2000

[4] H. Klaudel et F. Pommereau. *A concurrent semantics of static exceptions in a parallel programming language.* ICATPN'01, LNCS 2075. Springer, 2001

[5] H. Klaudel et F. Pommereau. *A class of composable and preemptible high-level Petri nets with an application to multi-tasking systems.* Fundamenta Informaticae, 50(1). IOS Press, 2002.

[6] R. Devillers, H. Klaudel, M. Koutny, E. Pelz et F. Pommereau. *Operational Semantics for PBC with Asynchronous Communication.* HPC'02. SCS, 2002.

[7] R. Devillers, H. Klaudel, M. Koutny et F. Pommereau. *An Algebra of Non-safe Petri Boxes.* AMAST'02, LNCS. Springer, à paraître.

[8] C. Bui Thanh, H. Klaudel et F. Pommereau. *Petri nets with causal time for system verification.* MTCS'02 (workshop de CONCUR'02). ENTCS, Elsevier, à paraître.

Autres publications

[AM96] L. Aceto et D. Murphy. *Timing and causality in process algebra.* Acta Informatica, 33(4). Springer, 1996.

[AD94] R. Alur et D. Dill. *A theory of timed automata.* Theoretical Computer Science, 126(2). Elsevier, 1994.

[Alu+93] R. Alur, C. Courcoubetis et D. Dill. *Model-checking in dense real-time.* Information and Computation, 104(1). Academic Press, 1993.

[BB91] J. Baeten et J. A. Bergstra. *Real time process algebra.* Formal Aspects of Computing, 3(2). Springer, 1991.

[Benv+91] A. Benveniste, P. Le Guernic et C. Jacquemot. *Synchronous programming with events and relations: the Signal language and its semantics.* Science of Computer Programming, 16(2). Elsevier, 1991.

[Ben+98] V. Benzaken, N. Hugon, H. Klaudel, E. Pelz et R.-C. Riemann. *M-net based semantics for triggers.* ICATPN'98, LNCS 1420. Springer, 1998.

[BK89] J. A. Bergstra et J. W. Klop. *Process theory based on bisimulation semantics.* Linear time, branching time and partial orders in logic and models for concurrency. LNCS 354. Springer, 1989.

[Ber93] G. Berry. *Preemption in concurrent systems.* FSTTCS'93, LNCS 761. Springer, 1993.

[Ber98] G. Berry. *The foundations of Esterel.* Language and Interaction: Essays in Honour of Robin Milner. MIT Press, 1998.

[Bes96] E. Best. *Semantics of sequential and parallel programs.* Prentice Hall, 1996.

[Bes97] E. Best. *A memory module specification using composable high-level Petri nets.* Formal Systems Specifications, The RPC-Memory Specification Case Study, LNCS 1169. Springer, 1997.

[BD87] E. Best et R. Devillers. *Sequential and concurrent behaviour in Petri net theory.* Theoretical Computer Science, 55(87). Elsevier, 1987.

[Bes+92] E. Best, R. Devillers et J. Hall *The Petri Box Calculus: a new causal algebra with multilabel communication.* Advances in Petri Nets 1992, LNCS 609. Springer, 1992.

[Bes+01a] E. Best, R. Devillers et M. Koutny. *A unified model for nets and process algebras.* Handbook of process algebra. Elsevier, 2001.

[Bes+01b] E. Best, R. Devillers et M. Koutny. *Petri Net Algebra.* EATCS Monographs on TCS. Springer, 2001.

[BF88] E. Best et C. Fernández. *Non sequential processes.* EATCS Monographs on Theoretical Computer Science, 13. Springer, 1988.

[Bes+95] E. Best, H. Fleischhack, W. Frączak, R. P. Hopkins, H. Klaudel et E. Pelz. *An M-net semantics of B(PN)^2.* Structures in Concurrency Theory 1995, Workshops in Computing. Springer, 1995.

[Bes+98] E. Best, W. Frączak, R. P. Hopkins, H. Klaudel et E. Pelz. *M-nets: an algebra of high level Petri nets, with an application to the semantics of concurrent programming languages.* Acta Informatica, 35. Springer, 1998.

[BH93] E. Best et R. P. Hopkins *B(PN)^2 — A basic Petri net programming notation.* PARLE'93, LNCS 694. Springer, 1993.

[BK92] E. Best et M. Koutny. *Petri net semantics of priority systems.* Theoretical Computer Science, 96(1). Elsevier 1992.

[BW00] E. Best et H. Wimmel. *Reducing k-safe Petri nets to pomset-equivalent 1-safe Petri nets.* ICATPN'2000, LNCS 1825. Springer, 2000.

[BC89] T. Bolognesi et P. Cremonese. *The weakness of some timed models for concurrent systems.* Rapport CNUCE C89-29, CNUCE-CNR, Pise, 1989.

[Bou98] E. Boufaïed. *Machines d'exécution pour langages synchrones.* Thèse de doctorat, Université de Nice-Sophia Antipolis, 1998.

[Boy01] M. Boyer. *Contribution à la modélisation des systèmes à temps contraint et application au multimédia.* Thèse de doctorat, Université Toulouse 3, 2001.

[BS91] J. Brzozowski et C. Seger. *Advances in asynchronous circuit theory, Part II: Bounded inertial delay models, MOS circuits, design techniques.* EACTS Bulletin, 43. 1991.

[Cas+87] P. Caspi, D. Pilaud, N. Halbwachs et J. Plaice. *Lustre: a declarative language for programming synchronous systems.* POPL'87. ACM Inc., 1987.

[Cer99] A. Cerone et A. Maggiolo-Schettini. *Time-based expressivity of time Petri nets for system specification.* Theoretical Computer Science, 216. Elsevier, 1999.

[Cla+96] E. Clarke, E. A. Emerson et A. P. Sistla. *Automatic verification of finite-state concurrent systems using temporal-logic specifications.* ACM Transactions on Programming Languages and Systems, 8(2). ACM Inc., 1986.

[Cla+99] E. Clarke, O. Grumberg et D. A. Peled. *Model checking.* MIT Press, 1999.

[CN96] R. Cleaveland et V. Natarajan. *An algebraic theory of process efficiency.* LICS'96. IEEE Computer Society, 1996.

[CZ91] R. Cleaveland et A. Zwarico. *A theory of testing for real-time.* LICS'91. IEEE Computer Society, 1991.

[CR83] J. E. Coholahan et N. Roussopoulos. *Timed requirements for timed driven systems using augmented Petri nets.* IEEE Transactions in Software Engineering, 9. IEEE Computer Society, 1983.

[Corb94] J. Corbett. *Modeling and analysis of real-time Ada tasking programs.* RTSS'94. IEEE Computer Society Press, 1994.

[Cor98] F. Corradini. *On performance congruences for process algebra.* Information and Computation, 145. Academic Press, 1998.

[Cor+99] F. Corradini, D. D'Ortenzio et P. Inverardi. *On the relationship among four timed process algebras.* Fundamenta Informatica, 34. IOS Press, 1999.

[Dev+97] R. Devillers, H. Klaudel et R.-C. Riemann. *General Refinement for High Level Petri Nets.* FSTTCS'97, LNCS 1346. Springer, 1997.

[Dev+98] R. Devillers, H. Klaudel et E. Pelz. *An Algebraic Box Calculus.* Journal of Automata, Languages and Combinatorics, 5(2). Otto-von-Guericke-Universität, Magdeburg, 2000.

[Dev+02] R. Devillers, H. Klaudel et R.-C. Riemann. *General Parameterised Refinement and Recursion for the M-net Calculus.* Theoretical Computer Science. Elsevier, à paraître.

[Dur91] R. Durchholz. *Causality, time, and deadlines.* Data & Knowledge Engineering, 6. North-Holland, 1991.

[Esp94] J. Esparza. *Model checking using net unfoldings.* Science of Computer Programming, 23. Elsevier, 1994.

[FM95] G.-L. Ferrari et U. Montanari. *Dynamic matrices and the cost analysis of concurrent programs.* AMAST'95, LNCS 936. Springer, 1995.

[FG97] H. Fleischhack et B. Grahlmann. *A Petri Net Semantics for B(PN)^2 with Procedures.* Parallel and Distributed Software Engineering 1997. IEEE Computer Society, 1997.

[FG98] H. Fleischhack et B. Grahlmann. *A Compositional Petri Net Semantics for SDL.* Application and Theory of Petri Nets 1998, LNCS 1420. Springer, 1998.

[FP00] H. Fleischhack et E. Pelz. *Partial order based model checking with data types.* Logic Colloquium'2000. North-Holland, 2000.

[FP02] H. Fleischhack et E. Pelz. *Causal boxes and hierarcical time boxes.* Rapport interne LACL, Avril 2002.

[Gen+80] H. J. Genrich, K. Lautenbach, et P. S. Thiagarajan. *Elements of General Net Theory.* Net Theory and Applications, Advanced Course on General Net Theory of Processes and Systems, LNCS 84. Springer, 1980.

[GL94] R. Gerber et I. Lee. *A resource-based prioritized bisimulation for real-time systems.* Information and Computation, 113(1). Academic Press, 1992.

[Gor+95] R. Gorrieri, M. Roccetti et M. Stancampiano. *A theory of processes with durational actions.* Theoretical Computer Science, 140(1). Elsevier, 1995.

[GR83] U. Goltz et W. Reisig. *The Non-sequential Behaviour of Petri Nets.* Information and Control, Vol 57(2-3). Academic Press, 1983.

[Grab81] J. Grabowski. *On Partial Languages.* Fundamenta Informaticae, (4)1. IOS Press, 1981.

[Gra95] B. Grahlmann. *Verifying telecommunication protocols with PEP.* RELECTRONIC'95, Scientific Society for Telecommunications, 1995.

[Gra97] B. Grahlmann. *The PEP Tool.* Computer Aided Verification, LNCS 1254. Springer, 1997.

[GB96] B. Grahlmann et E. Best. *PEP – More than a Petri Net Tool.* Tools and Algorithms for the Construction and Analysis of Systems, LNCS 1055. Springer, 1996.

[GS96] J. F. Groote et J. Springintveld. *Focus point and convergent process operators. A proof strategy for protocol verification.* Logic Group Preprint Series 142. Departement of Philosophy, Utrecht University, 1995.

[Hal+91] N. Halbwachs, P. Caspi, P. Raymond et D. Pilaud. *The synchronous dataflow programming language Lustre.* Proceedings of the IEEE, 79(9). IEEE Inc., 1991.

[Här+94] M. G. Härbour, M. H. Klein et J. P. Lehoczky. *Timing analysis for fixed-priority scheduling of hard real-time systems.* IEEE Transactions in Software Engineering, 20(1). IEEE Computer Society, 1994.

[HR95] M. Hennessy et T. Regan. *A temporal process algebra.* Information and Computation, 117. Academic Press, 1995.

[Hoa89] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1989.

[HV87] M. A. Holliday et M. K. Vernon. *A generalized timed Petri net model for performance analysis.* IEEE Transactions in Software Engineering, 13. IEEE Computer Society, 1987.

[Hoo+96] P. W. Hoogers, H. C. M. Kleijn et P. S. Thiagarajan. *An event structure semantics for general Petri nets.* Theoretical Computer Science, 153. Elsevier, 1996.

[Jan+99] P. Janecek, A. Ratzer et W. Mackay. *Redesigning Design/CPN: integrating interaction and Petri-nets-in-use.* International Workshop on Coloured Petri nets, Aarhus. 1999.

[Jen81] K. Jensen. *Coloured Petri nets and the invariant method.* Theoretical Computer Science, 14. Elsevier, 1981.

[Jen92] K. Jensen. *Coloured Petri nets. basic concepts, analysis methods and practical use.* EATCS Monographs on Theoretical Computer Science, 1. Springer, 1992.

[Kha+96] W. Khansa, P. Aygalinc et J. P. Denat. *Structural analysis of P-Time Petri nets.* CESA'96, IMACS Multiconférence. 1996.

[Kha+96b] W. Khansa, J. P. Denat et S. Collart-Dutilleul. *P-Time Petri nets for manufacturing systems.* WODES'96. Institute of Electronical Engineers, Computing and Control Division, 1996.

[Kha97] W. Khansa. *Réseaux de Petri p-temporels : contribution à l'étude des systèmes à événements discrets.* Thèse de doctorat, Université de Savoie, 1997.

[Kis+97] M. Kishinevsky, J. Cortadella, A. Kondratyev, L. Lavagno, A. Taubin et A. Yakovlev. *Coupling asynchrony and interrupts: Place Chart Nets.* ICATPN'97, LNCS 1248. Springer, 1997.

[Kla95] H. Klaudel. *Modèles algébriques, basés sur les réseaux de Petri, pour la sémantique des langages de programmation concurrents.* Thèse de doctorat, LRI, Université Paris XI, Orsay, 1995.

[Kla97] H. Klaudel et R.-C. Riemann. *High-level expressions and their SOS semantics.* CONCUR'97, LNCS 1243. Springer, 1997.

[Kla98] H. Klaudel, E. Pelz et R.-C. Riemann. *Relating M-expressions and M-nets.* DAPSYS'98, Report TR-120, University of Vienna, 1998.

[Kla00] H. Klaudel. *Parameterized M-expression semantics of parallel procedures.* DAPSYS'00. Kluwer Academic Publishers, 2000.

[Kla01] H. Klaudel. *Compositional high-level Petri net semantics of a parallel programming language with procedures.* Sciences of Computer Programming, 41(3). Elsevier, 2001.

[Kou00] M. Koutny. *A compositional model of time Petri nets.* ICATPN'00, LNCS 1825. Springer, 2000.

[Kou+02] V. Khomenko, M. Koutny et W. Vogler. *Canonical prefixes of Petri net unfoldings.* CAV'02, LNCS. Springer, 2002 (à paraître).

[Kri71] S. A. Kripke. *Semantical considerations on modal logic.* Reference and Modality. Oxford University Press, 1971.

[Lar+97] K. G. Larsen, P. Pettersson et W. Yi. *UPPAAL in a nutshell.* International Journal on Software Tools and Technology Transfer, 1(1-2). Springer, 1997.

[Lil96] J. Lilius. *OB(PN)²: An Object Based Petri Net Programming Notation.* Euro-Par'96 volume 1, LNCS 1123. Springer, 1996.

[LP96] J. Lilius et E. Pelz. *An M-net semantics for B(PN)² with procedures.* Eleventh International Symposium on Computer and Information Science, 1. Middle East Technical University, 1996.

[McM95] K. MacMillan. *A technique of state space search based on unfoldings.* Formal Methods in System Design, 6. Kluwer Academic Publishers, 1995

[MSW90] A. Maggiolo-Schettini et J. Winkowski. *A compositional semantics for timed Petri nets.* Fundamenta Informaticae, 13. IOS Press, 1990.

[Mäk99] M. Mäkelä. *MARIA: modular reachability analyser for algebraic system nets.* Manuel en ligne, `http://www.tcs.hut.fi/maria`, 1999.

[Mar89] F. Maraninchi. *Argonaute, graphical description, semantics and verification of reactive systems by using a process algebra.* Workshop on Automatic Verification Methods for Finite State Systems, LNCS 407. Springer, 1989.

[MF01] O. Marroquín Alonzo et D. de Frutos Escrig. *Extending the Petri Box Calculus with time.* ICATPN'01, LNCS 2075. Springer, 2001.

[Maz86] A. Mazurkiewicz. *Trace theory.* Advances in Petri Nets 1986, Part 2, LNCS 255. Springer, 1986.

[Mer74] P. M. Merlin. *A study of recoverability of communication protocols.* Ph.D. Thesis, Department of Computer Science, University of California, 1974.

[MF76] P. M. Merlin et D. J. Farber. *Recoverability of communication protocols – implications of a theoretical study.* IEEE Transactions on Communications, 24. IEEE Communications Society, 1976.

[Mill89] D. L. Mills. *Network Time Protocol (version 2), specification and implementation.* Request For Comments 1119. Network working group, 1989.

[Mil80] R. Milner. *A Calculus of Communicating Systems.* LNCS 92. Springer, 1980.

[Mil83] R. Milner. *Calculi for synchrony and asynchrony.* Theoretical Computer Science, 25. Elsevier, 1983.

[Mil89] R. Milner. *Communication and concurrency.* International series in computer science. Prentice Hall, 1989.

[Mil+92] R. Milner, J. Parrow et D. Walker. *A calculus of mobile processes. Part I and II.* Information and Computation, 100. Academic Press, 1992.

[MT90] F. Moller et C. Tofts. *A temporal Calculus of Communicating Systems.* CONCUR'90, LNCS 459. Springer, 1990.

[MT91] F. Moller et C. Tofts. *Relating processes with respect to speed.* CONCUR'91, LNCS 527. Springer, 1991.

[Mur89] T. Murata. *Petri nets: properties, analysis and applications.* Proceedings of the IEEE, 77(4). IEEE Inc., 1989.

[NS94] X. Nicollin et J. Sifakis. *The Algebra of Timed Processes ATP: theory and applications.* Information and Computation, 114. Academic Press, 1994.

[Nis97] N. Nissanke. *Realtime systems.* Series in Computer Science. Prentice Hall, 1997.

[Petri62] C. A. Petri. *Kommunikation mit Automaten.* Schriften des Instituts für instrumentelle Mathematik. Universität Bonn, 1962.

[Pet81] J. L. Peterson. *Petri net theory.* Prentice Hall, 1981.

[Pett99] P. Petterson. *Modelling and verification of real-time systems using timed automata: theory and practice.* Ph.D. Thesis, Department of Computer Systems, Uppsala University, Sweden, 1999.

[Pin+98] S. Pinchinat, É. Rutten et R. K. Shyamasundar. *Taxonomy and expressiveness of preemption: a syntactic approach.* ASIAN'98, LNCS 1538. Springer, 1998.

[Plo81] G. D. Plotkin *A structural approach to operational semantics.* Report FN-19, Computer Science Department, University of Aarhus, 1981.

[Pnu77] A. Pnueli. *The temporal logic of programs.* FOCS'77. IEEE Computer Society, 1977

[RH80] C. V. Ramamoorhy et G. S. Ho. *Performance evaluation of asynchronous concurrent systems using Petri nets.* IEEE Transactions in Software Engineering, 6. IEEE Computer Society, 1980.

[Ram74] C. Ramchandani. *Analysis of asynchronous concurrent systems using Petri nets.* Ph.D. Thesis, Project MAC, MAC-TR 120, MIT, 1974.

[RR88] G. M. Reed et A. W. Roscoe. *A timed model for Communicating Sequential Processes.* Theoretical Computer Science, 58. Elsevier, 1988.

[Ric85] G. Richter. *Clocks and their use for time modeling.* Information systems: Theoretical and Formal Aspects. North-Holland, 1985.

[Ric98] G. Richter. *Counting interfaces for discrete time modeling.* Technical report 26, GMD. Septembre 1998.

[Rie99] R.-C. Riemann. *Modelling of concurrent systems: structural and semantical methods in the high-level Petri net calculus.* Thèse de doctorat, LRI, Université Paris XI, Orsay, 1999.

[RS01] S. Roch et P. H. Starke. *INA: Integrated Net Analyser.* Manuel en ligne, `http://www.informatik.hu-berlin.de/lehrstuehle/automaten/ina`, 2001.

[Sif77] J. Sifakis. *Use of Petri nets for performance evaluation.* Measuring modeling and evaluating computer systems. North-Holland, 1977.

[Sif79] J. Sifakis. *Performance evaluation of systems using nets.* Net theory and applications, Advanced course on general net theory of processes and systems, LNCS 84. Springer, 1979.

[Sny] P. Snyder. *tmpfs: a virtual memory file system.* White Papers, Sun Microsystems Inc.

[Sta81] P. H. Starke. *Processes in Petri nets.* FCT'81, LNCS 117. Springer, 1981.

[Sta78] P. H. Starke. *Free Petri nets languages.* Mathematical Foundations of Computer Science, LNCS 64. Springer, 1978.

[Vog91] W. Vogler. *Failures semantics based on interval semiwords is a congruence for refinement.* Distributed Computing, 4. Springer, 1991.

[Vog92] W. Vogler. *Modular Construction and Partial Order Semantics of Petri Nets.* LNCS 625. Springer, 1992.

[Wal83] B. Walter. *Timed Petri-nets for modelling and analyzing protocols with real-time characteristics.* Third IFIP workshop on protocols specification, testing and verification. North-Holland, 1983.

[Yi90] W. Yi. *Real-time behaviour of asynchronous agents.* CONCUT'90, LNCS 458. Springer, 1990.

[Yov97] S. Yovine. *Kronos: A verification tool for real-time systems.* International Journal of Software Tools for Technology Transfer, 1(1/2). Springer, 1997.

[Zwa01] M. B. van der Zwaag. *The cones and foci proof technique for timed transition systems.* Information Processing Letters, 80(1). Elsevier, 2001.

Résumé. Cette thèse traite de la modélisation des systèmes temps-réel à l'aide de réseaux de Petri. Nous considérons séparément la question de représentation du temps et celle de la préemption (interruption des processus d'un système) qui est d'un usage courant pour les applications temps-réel.

Nous utilisons des réseaux de Petri sans extension par des informations concernant le temps. Notre approche consiste alors à introduire le temps par des sous-réseaux spécifiques représentant les horloges du système modélisé. Le résultat est appelé *temps causal* puisque seule la causalité définit la relation de précédence entre les occurrences des événements. Afin d'obtenir une formulation élégante de l'approche causale du temps, nous utilisons le modèle des M-nets, une classe de réseaux de Petri colorés composables à la manière des algèbres de processus. Nous étendons ce modèle de façon à permettre la représentation efficace des communications asynchrones entre processus et les bases théoriques liées à cette extension sont revisitées et mises à jour. Nous utilisons les M-nets ainsi étendus pour modéliser plusieurs horloges aux fonctionnalités différentes. Nous montrons comment des systèmes à plusieurs horloges, synchronisées ou non, peuvent être assez simplement obtenus. La pertinence de notre approche est évaluée par une étude de cas et appliquée à la sémantique d'une extension d'un langage de programmation parallèle, appelé B(PN)^2, par des instructions liées au temps.

Pour introduire la préemption nous proposons une nouvelle extension des M-nets avec des opérations permettant la suspension/reprise et l'avortement. Le modèle obtenu est étudié sur le plan théorique et appliqué à l'extension de la sémantique de B(PN)^2 par des exceptions et un système de tâches.

Mots clefs. Réseaux de Petri, temps-réel, modélisation, vérification, composabilité, concurrence, sémantique.

Abstract. This thesis is concerned with the modelling of real-time systems using Petri nets. We investigate both the problem of time representation, and of preemption (*i.e.*, interruption of the processes of a system) which is often needed for real-time applications.

We consider Petri nets without explicit temporal information. The key idea of our approach consists in introducing time through specific sub-nets representing clocks present in the system being modelled. This results in what we refer to as *causal time* since causality is the only means by which the precedence relationship between event occurrences is specified. For a succinct formulation of the causal time model, we use M-nets which are a class of coloured Petri nets endowed with compositional operators similar to those used by the standard process algebras. We extend the M-nets model in order to allow an efficient representation of asynchronous interprocess communication, and establish a number of theoretical results concerning such an extension. We use the resulting M-nets to model several clocks with varied characteristics, showing how they can be synchronised or kept independent. The suitability of the proposed approach is evaluated using a case study, and it is then used to give a semantics of the parallel programming language B(PN)^2 extended with real-time constructs.

In order to deal with preemption, we propose another extension of the M-nets model supporting operators for process suspension/resuming and abortion. The resulting model is investigated at the theoretical level, and applied to B(PN)^2 enhanced with exceptions and tasks.

Keywords. Petri nets, real-time, modelling, verification, composability, concurrency, semantics.