École Centrale de Nantes **Université de Nantes**

École Doctorale
Sciences et Technologies
de l'Information et des Materiaux

Année 2001

Thèse de Doctorat

Diplôme délivé conjointement par
L'École Centrale de Nantes et l'Université de Nantes

Spécialité : Automatique et Informatique Appliquée

Présentée et soutenue publiquement par :

Frédéric Herbreteau

le 14 décembre 2001
à l'École Centrale de Nantes, Nantes

Titre

Automates à file réactifs embarqués
Application à la vérification de systèmes temps-réel

Jury

Rapporteur	Bernard Boigelot	(PR Univ. de Liège, Belgique)
Examinateur	Franck Cassez	(CR CNRS, IRCCyN)
Président du jury	Alain Finkel	(PR ENS de Cachan)
Rapporteur	Paul Gastin	(PR Univ. Paris VII)
Examinateur	Olivier Roux	(PR ECN)
Rapporteur	François Vernadat	(MCF Univ. de Toulouse)

Directeur de thése : Olivier Roux
Co-encadrant : Franck Cassez
Laboratoire : IRCCyN (UMR CNRS 6597), 1, rue de la Noë - BP 92101
44321 Nantes cedex 3, FRANCE.

N°ED 0366-055

Remerciements

Une thèse constitue une longue histoire, et comme toutes les histoires, elle a un commencement. La mienne débute en 1986, lorsque mon instituteur de CM2, M. Christian André, me fit utiliser pour la première fois un ordinateur (en l'occurrence un Thomson TO7). Il avait mis en place un petit programme de calcul mental où des chiffres apparaissaient et disparaissaient, de plus en plus rapidement, augmentant progressivement la difficulté de les additionner ou de les multiplier. Je suis tombé sous le charme de cette incroyable machine lorsqu'au lieu de lui donner la réponse qu'elle attendait, par exemple 35, je lui ai retourné l'expression $17 + 8 + 10$, qu'elle est parvenu à interpréter correctement. Du haut de mes 10 ans et demi, c'était la révélation de l'intelligence de l'ordinateur.

Depuis, cette illusion s'est envolée, particulièrement lorsque j'eus commencé à programmer. En cela, je dois une fière chandelle à mon petit frère, Guillaume, d'être né pendant le mois de juillet 1986, nous privant du coup, ma soeur et moi, de vacances. Nos parents jugèrent bon de compenser ce manque en nous offrant un Amstrad CPC 6128, sur lequel je fis mes premieres armes, et je perdis, rapidement, mes illusions d'une quelconque intelligence de l'engin. Malgré cela, programmer devint une vraie passion et me poussa à étudier l'informatique.

C'est ainsi que quelques années après, j'en suis venu, en septembre 1998, à commencer ma thèse de doctorat en automatique et informatique appliquée, et que je l'ai défendue le 14 décembre 2001.

Ces 3 années n'auraient pas été si bonnes sans la qualité de l'accueil dont j'ai bénéficié à l'IRCCyN (Institut de Recherche en Communication et Cybernétique de Nantes). Un grand merci à tous ceux qui œuvrent et qui ont œuvré au sein de l'institut pour que les doctorants soient aussi bien intégrés. La vie au laboratoire ne se limitant pas au travail, je tiens aussi à remercier les doctorants (et les autres) avec qui j'ai pu partager les pauses cafés, les déjeuners, des soirées, et les inénarrables parties de xblast.

Je tiens à remercier chaleureusement les acteurs de l'informatique libre. Ce travail n'aurait pas abouti sans un accès aussi libre aux travaux et aux idées qui ont guidé mes recherches. Je n'aurais pas non plus produit ce document, ni les outils auxquels je fais mention, sans l'existence des logiciels libres. Un grand merci à tous ceux qui, par leur *éthique hacker*[1], m'ont permis de travailler quotidiennement.

J'ai une très grande reconnaissance pour les personnes qui ont travaillé plus particulièrement à cet aboutissement : Franck Cassez et Olivier Roux qui ont su me diriger, et faire preuve d'une grande patience, pendant ces 3 ans. Je considère que les félicitations qui m'ont été accordées

[1] Au sens de Pekka Himanen, *L'Éthique hacker et l'esprit de l'ère de l'information*, Exils, 2001.

par le jury, leur reviennent, en témoignage de la qualité de leur contribution à la réussite de ce travail, chacun à leur façon.

Je souhaite aussi remercier Bernard Boigelot, Paul Gastin et François Vernadat pour avoir pris le temps de lire ce mémoire, pour avoir évalué mon travail, et pour m'avoir indiqué les erreurs et les imprécisions qui s'y trouvaient.

Un grand merci à Alain Finkel, non seulement pour avoir présidé le jury lors de la soutenance de ma thèse, mais aussi pour la collaboration que nous avons développée, et pour les bons conseils qu'il m'a prodigués quant à la réalisation de ce travail et à mes perspectives professionnelles.

Un aussi grand merci à Grégoire Sutre pour sa contribution aux résultats du chapitre 4, et pour avoir relu en partie ce document.

Et puis, il est dit que l'on garde le meilleur pour la fin, alors je n'y ferai pas exception. La préparation de ma thèse m'a demandé 3 années pendant lesquelles la vie ne s'est pas arrêtée. La réalisation de ce travail est emprunte du chaos continuel qui fait la vie quotidienne, et qui n'est rendue possible que par la présence et le soutient de mes amis et de ma famille. Je n'en dresserais pas la liste, mais que tous ceux qui s'y reconnaitreront sachent qu'ils ont ma plus grande gratitude.

Bordeaux, le 18 janvier 2002, Frédéric Herbreteau.

Table des matières

Introduction

De la nécessité de «vérifier» les systèmes critiques

Notre environnement familier regorge d'appareils contrôlés par des systèmes informatiques. Des équipements électro-ménagers à domicile, aux automobiles, aux métros ou aux avions pour le transport, des centrales éléctriques pour l'énergie, aux satellites et aux équipements de télécommunication pour l'information : ces systèmes sont partout. Notre société est de plus en plus dépendante du bon fonctionnement des systèmes informatiques, et pourtant, chacun a à l'esprit des «bugs» célèbres[2].

- Le 21 novembre 1985, la «Bank of New York» connaissait une forte fièvre due à une crise sur le marché des bons du Trésor. Les agents de la banque se mirent alors à investir dans les métaux précieux (or,argent,etc.) pour sauver ce qui pouvait encore l'être. Cependant, ce «krach» complètement fictif était provoqué par un simple problème de représentation des données dans le système informatique de la banque, et donc local à la banque. La «Bank of New York» dut emprunter 20 milliards de dollars à l'état américain pour éponger les dettes occasionnées par cette erreur.
- Le 15 Janvier 1990, une partie du réseau téléphonique d'AT&T tombait en panne suite à une erreur de programmation (une simple instruction «`break`» manquante). Neuf heures durant, il fût impossible de passer le moindre coup de téléphone, et l'on estime à 7.000.000 le nombre d'appels qui n'ont ainsi pas pu être passés. Il fallut de nombreuses années à AT&T pour refaire sa réputation.
- Le 2 novembre 1994, la société Intel dut reconnaître un grave bug de calcul d'une simple division de nombres réels sur son nouveau processeur «Pentium». La société Intel a du proposer l'échange standard des processeurs déjà commercialisés, sous la pression de ses concurrents.
- Le 4 juin 1996, la fusée Ariane V (vol 501) explose en vol après des changements successifs de trajectoires incompréhensibles. L'enquète qui suivit l'accident conclût à l'utilisation, par erreur, d'une partie du code de contrôle de la fusée Ariane IV. Il s'est alors produit un dépassement de capacité d'une variable, à cause de la différence des caractéristiques physiques des deux lanceurs.
- Le 4 juillet 1997, la mission *Mars Pathfinder* débute réellement lorsque la sonde *Sojourner* se pose sur la planète Mars. Elle est chargée d'analyser son environnement (minéral, atmosphérique, météorologique, etc.), puis de transmettre les données recueuillies à la Terre. Les différentes tâches qui composent le système informatique utilisent une même ressource,

[2]Les exemples que nous présentons ici, nous ont été inspirés par la thèse d'Emmanuel Fleury [43].

accédée en exclusion mutuelle, pour écrire, lire et transmettre les données. Ces tâches disposaient de priorités différentes, et c'est là qu'est apparu le problème. Des conflits de priorité entres les tâches de gestion des données météorologiques conduisaient à la situation où :
- La tâche de communication (de moyenne priorité) attendait la fin de la tâche de lecture, pour accéder à la ressource,
- La tâche de lecture (de haute priorité), attendait que la tâche d'écriture ait achevé son travail pour commencer le sien,
- La tâche d'écriture (de faible priorité), attendait que la tâche de communication, plus prioritaire, se termine pour demander la ressource.

Le système entrait alors dans une situation de **blocage** (ou **deadlock**), et, après un certain temps, un processus chargé d'entretenir la vivacité du système redemarrait l'unité informatique, perdant du coup l'ensemble des données recueillies. Après plusieurs jours d'inutilisabilité de la sonde, les ingénieurs du *Jet Propulsion Laboratory (NASA)*, ont «résolu» le problème en désactivant purement et simplement les priorités d'accès à la ressource commune.

- Ces différents problèmes ont causé de graves pertes financières, mais il y a bien pire. Le «Therac-25» était un accélérateur linéaire médical, pour le traitement des tumeurs par irradiation localisée, commercialisé par la société publique Canadienne «Atomic Energy Commission Limited» (AECL). Entre juin 1985 et janvier 1987, un dysfonctionnement du système informatique qui contrôllait cet appareil causa la mort de 4 personnes. Le problème était causé par une situation de «race condition» : lorsque l'opérateur entrait les paramètres de traitement du patient en moins de 8 secondes, ceux-ci étaient remplacés par des valeurs issues d'autres champs de saisie. Certains patients ont ainsi reçu des doses de radiations supérieures de 20 fois la normale.

Ces quelques exemples plaident en la faveur de la certification des systèmes informatiques. La solution idéale consiste à *vérifier* ces systèmes, c'est à dire, à examiner chacun de leurs comportements et à s'assurer qu'ils satisfont les propriétés (dites *comportementales*) souhaitées. Pour réaliser cette opération, il est nécessaire de définir un modèle formel (i.e. mathématique) pour les systèmes considérés et pour les propriétés examinées, ainsi que les techniques les manipulant.

Les *propriétés comportementales* des systèmes sont usuellement définies par des formules de *logique temporelle*, comme Ltl [86, 32] et Ctl [25, 32]. Elles permettent, entre autres, l'expression de :
- *propriétés de vivacité* qui indiquent que «sous certaines conditions, quelque chose finira par avoir lieu». Par exemple, pour le systèmes de Sojourner (dont le problème est décrit ci-dessus), les ingénieurs du *Jet Propulsion Laboratory* étaient persuadés que la formule de vivacité «si la tâche d'écriture est activée inévitablement la tâche de lecture s'exécutera» était satisfaite, grace aux priorités choisies et à la causalité liant ces deux tâches. Ils n'avaient par contre pas prévu que la tâche de communication viendrait empêcher la réalisation de la condition «si la tâche d'écriture est activée» conduisant au blocage du système.
- *propriétés de sûreté* qui modélisent le fait que «rien de néfaste» ne se produit pendant toute exécution d'un système. Par exemple, pour le cas du Therac-25 (décrit ci-dessus), la

vérification de la satisfaction de la propriété de sûreté «La dose de radiations émises ne dépasse jamais n rad», aurait permi d'éviter les décès survenus (en admettant qu'il eut été possible de la vérifier).

Les propriétés de sûreté ne rassemblent généralement pas l'ensemble des propriétés qu'un système doit satisfaire pour être *correct*, mais ce sont généralement les plus importantes, en quantité comme en qualité.

Il reste maintenant à modéliser les systèmes informatiques de contrôle. Mais, que désignons-nous intuitivement par le terme «système informatique de contrôle» ?

Les systèmes réactifs asynchrones

Nous considérons les *applications réactives* [57, 87, 12], qui forment la classe des systèmes évoluant au rythme des sollicitations provenant de leur environnement. Ces sollicitations sont modélisées par des *événements discrets*, et nous distinguons deux approches de la programmation des applications réactives, suivant leur perception de ces événements.

1. La première, la *programmation synchrone* [11, 53], fait l'hypothèse que l'application réagit «suffisament rapidement» pour considérer qu'elle évolue au même rythme que son environnement. Les langages Argos [75] (proche des Statecharts [56]), Esterel [13, 34], Lustre [54] et Signal [44] sont les langages les plus couramment utilisés pour la programmation synchrone des applications réactives. Les événements qui cadencent l'exécution sont alors des signaux dont on ne s'intéresse qu'à la présence, ou à l'absence, dans l'instant considéré, et sans remanescence. La compilation de ces programmes donne donc lieu à des systèmes de transitions finis, dont un parcours exhaustif permet d'assurer la satisfaction des propriétés comportementales [74, 55, 30, 52, 67].

   L'approche synchrone de la programmation des systèmes réactifs, s'appuie sur la «rapidité de réaction» de l'application. Cependant, cette hypothèse n'est généralement pas satisfaite par les systèmes modélisés. Il faut alors créer les conditions nécessaires à la validité de l'approche synchrone : la différence entre les rythmes de fonctionnement de l'application et de son environnement est «gommée» au moyen d'un *gestionnaire d'événements*. Servant d'interface entre l'environnement et le programme, il stocke les événements émis par l'environnement et les restitue à l'application lorsqu'elle est disposée à les traiter, donnant l'impression qu'ils évoluent au même rythme.

2. À l'inverse, la *programmation asynchrone* ne fait aucune hypothèse sur les rythmes relatifs de l'application par rapport à son environnement. Il faut donc considérer que l'application n'est parfois pas disposée à répondre, immédiatement, aux événements provenant de son environnement. C'est le cas dès que le traitement en cours est plus prioritaire que celui qui serait déclenché par la nouvelle occurrence. Dans l'approche asynchrone, les occurrences d'événements sont gérées directement par le programme.

   Le langage Electre [31, 88, 23] est particulièrement destiné à la programmation asynchrone des applications réactives. La sémantique des programmes Electre est définie par les *automates à file réactifs (AFR)* [23, 92, 21]. Ces automates possèdent la capacité de mémoriser les événements qui ne peuvent pas être pris en compte immédiatement par l'application. Ces occurrences mémorisées sont alors traitées *dès que possible*, et avec *priorité à la plus ancienne* : c'est la politique FIFFO (First In First Fireable Out). Le modèle sémantique des programmes ELECTRE est donc *a priori* infini : il tient compte de la file de mémorisation. La vérification des propriétés du modèle ne peut donc plus s'effectuer par un parcours exhaustif de son espace d'états.

La gestion FIFFO de la file offre un très bon compromis entre les politiques FIFO (classique) et SET utilisées pour les gestionnaires d'événements des langages synchrones [14, 26]. En effet, la politique FIFO a l'inconvénient de fournir les événements dans un ordre strict et statique : toujours la première occurrence de la file. Ainsi, si à un instant donné le système réactif modélisé n'est pas en mesure de traiter le premier événement de la file, alors qu'il pourrait traiter le second, il doit attendre de changer d'état, puis de traiter l'occurrence qui est en tête de la file pour pouvoir, enfin, traiter celle qui était en seconde position. Le système n'est donc pas «aussi réactif» qu'on pourrait le souhaiter. La politique SET, à l'inverse, n'ordonne pas les occurrences d'événements mémorisés. Dans ce cas, les occurrences peuvent effectivement être traitées aussitôt que c'est possible, mais rien ne garantit qu'une occurrence mémorisée sera inévitablement traitée. Avec cette politique, on s'expose à un risque d'inéquité. La politique FIFFO [20] fournit justement les avantages des deux politiques précédentes (FIFO et SET), tout en évitant leurs inconvénients. En effet, le choix de l'occurrence mémorisée à traiter se fait en fonction de l'état courant du système : c'est un ordonnancement dynamique. Le contenu de la file est examiné en faisant abstraction des occurrences qui ne peuvent pas être traitées dans l'état courant du système, et, la première occurrence (de fait, la plus ancienne) est traitée.

Objectifs de cette thèse

Les événements mémorisables des programmes ELECTRE et des automates à file réactifs peuvent être de deux types :

- soit à mémorisation unique, alors une occurrence qui ne peut pas être traitée (au moment où elle survient) n'est mémorisée que si aucune occurrence du même événement n'est déjà en mémoire,
- sinon, à mémorisation multiple, auquel cas toute occurrence qui ne peut pas être traitée immédiatement est mémorisée.

Le nombre d'occurrences mémorisées simultanément n'est *a priori* pas borné, dès lors que le programme ELECTRE (ou l'automate à file réactif) dispose d'un événement à mémorisation multiple. Cependant, la bornitude de la file de mémorisation est une propriété particulièrement importante :

- pour la *vérification* des applications, puisque les techniques qui s'appliquent à des modèles infinis diffèrent fondamentalement de celles qui sont utilisées pour les modèles finis.
- pour l'*implantation* des applications, puisqu'il n'existe aucun dispositif physique qui permette de mémoriser un nombre non borné d'éléments. Il est donc indispensable qu'une application mémorise un nombre borné d'événements, simultanément, pour qu'elle puisse être implantée.

Nous considérons donc principalement la question suivante :

La file de mémorisation des automates à file réactifs est-elle bornée ?

La bornitude est alors considérée comme un critère de *bonne spécification* des automates à file réactifs et des programmes ELECTRE. Elle dépend fortement du rythme auquel surviennent les événements, et donc, de l'*environnement* du système considéré. Cette spécification fait défaut aux programmes ELECTRE et aux automates à file réactifs qui ne modélisent que le comportement des applications réactives, c'est à dire, des systèmes réactifs *ouverts*. C'est pourquoi nous introduisons les *systèmes à file réactifs embarqués* qui modélisent le plongement des automates à file réactifs (donc des programmes ELECTRE) dans un environnement fixé, conduisant ainsi à une spécification *close* des applications réactives. Nous nous intéressons alors à la vérification des propriétés comportementales de ces systèmes, et particulièrement, à leur bornitude.

Contexte de cette thèse et travaux précédents

Le projet ELECTRE a vu le jour au milieu de années 80 [31]. Depuis une quinzaine d'années, de nombreux travaux ont été réalisés autour de ce langage et des automates à file réactifs [31, 88, 15, 20, 89, 23, 92, 16, 61, 21], pour n'en citer que quelques uns. Notre thèse s'inscrit donc dans la continuité de ces travaux, en considérant le problème particulièrement crucial qu'est la bornitude.

Un programme ELECTRE, et un automate à file réactif, peut donc être vu comme une machine à états finie (comme beaucoup de programmes informatiques), augmentée d'une file, dont la gestion FIFFO (First In First Fireable Out), fait la particularité. En ce sens, notre modèle est proche des *machines communicantes* [18, 1, 2, 38, 24, 40] qui sont constituées d'un ensemble des machines à états finies (qui peut être vue comme une seule machine) et d'un ensemble de files FIFO pour modéliser la communication entre les machines.

La vérification de la satisfaction des propriétés de logique temporelle par un modèle infini est un problème généralement indécidable. C'est à dire qu'il n'existe pas d'algorithme qui indique, pour toute propriété, et pour toute instance du modèle, s'il satisfait, ou non, cette propriété. La vérification exhaustive des systèmes informatiques relève généralement de l'utopie. Cependant, des techniques de vérification adaptées, introduites dans [1], pour les machines communicantes avec des canaux non fiables, permettent de décider certains problèmes de vérification comme l'accessibilité d'une configuration de la machine considérée. Cette possibilité permet la vérification des propriétés de sûreté en s'assurant qu'une mauvaise configuration (par exemple, lorsque le Therac-25 émet un niveau de radiations trop élevé) n'est pas accessible. Ces techniques ont par la suite été adaptées à d'autres modèles infinis (par exemple [81, 19]), puis généralisées [3, 4, 41]. L'intense activité qui entoure la vérification des systèmes infinis s'explique simplement par le fait que la grande majorité des données traitées par les systèmes informatiques prennent leurs valeurs dans des domaines infinis. La vérification des propriétés de sûreté prend alors tout son sens puisque les systèmes informatiques ne peuvent représenter qu'une partie finie de ces domaines.

Notre étude s'inscrit, par ailleurs, dans la suite de [91, 92] où les auteurs montrent que :

– L'ensemble des états accessibles d'un automate à file réactif est effectivement reconnaissable,
– Le model-checking de LTL\X et des formules existentielles équitables de CTL\X est décidable[3].

Enfin, avant de nous intéresser à la bornitude des automates à file réactifs (et des programmes ELECTRE), nous devons spécifier leurs environnements, c'est à dire la façon dont les événements du système surviennent. Cette spécification fait, en effet, défaut aux programmes ELECTRE et aux AFR, tout en étant déterminante pour leur bornitude. Notons que certains travaux récents [14, 26] font apparaître la spécification de l'environnement des programmes synchrones pour la vérification des systèmes temps-réel.

Notre contribution

1. Dans la première partie (page 23), après avoir rappelé les caractéristiques des automates à file réactifs, nous montrons que :

   (a) l'ajout de l'opération *reset*, qui consomme toutes les occurrences mémorisées d'un événement donné en une seule transition, aux automates à file réactifs, n'altère en rien ce modèle,

   (b) en conséquence, les événements à mémorisation unique peuvent être simulés par des événements à mémorisation multiple.

   Nous pouvons donc considérer par la suite que tous les événements mémorisables des programmes ELECTRE et des automates à file réactifs sont à mémorisation multiple.

2. Ensuite, dans la partie II (page 67), nous considérons la vérification des systèmes à file réactifs embarqués. Cette dernière particularité (l'aspect embarqué) constitue la part majeure de notre contribution présentée dans cette thèse. Les environnements sont spécifiés par des systèmes de transitions étiquetés, et nous considérons la décidabilité des problèmes classiques de vérification, tels que l'accessibilité, la bornitude et le model-checking des logiques LTL [86, 32] et CTL [25, 32]. Nous montrons, principalement, les résultats suivants :

   (a) Les *systèmes à file réactifs embarqués (SFR Embarqués)* avec 2 événements mémorisables et un environnement périodique ont l'expressivité des machines de Turing. Par conséquent, tous les problèmes de vérification sont indécidables.

   (b) L'ensemble des configurations accessibles d'un système à file réactif embarqué avec 1 événement mémorisable et un environnement rationnel est effectivement reconnaissable.

   (c) L'ensemble des prédecesseurs d'une configuration d'un système à file réactif embarqué *avec perte*[4], avec un environnement bien structuré est effectivement calculable.

[3] LTL\X est le fragment de la logique LTL [86, 32] obtenu en supprimant l'opérateur «next» X, et les formules existentielles de CTL\X sont obtenues depuis CTL [25, 32] en ne considérant que la modalité EF.

[4] C'est à dire que les occurrences mémorisées peuvent être perdues par le système, de façon non déterministe et non contrôlée.

Ce dernier résultat permet, entre autres, de vérifier les propriétés exprimées dans le fragment[5] (EF) de la logique CTL. Par conséquent, il est possible de décider la satisfaction de la grande majorité des formules de sûreté pour les systèmes à file réactifs embarqués *avec perte*, munis d'un environnement bien structuré. La sémantique de perte conserve l'ensemble des comportements du modèle sans perte (et en ajoute de nouveaux). Par conséquent, lorsqu'une formule de sûreté («aucune mauvaise situation n'est accessible») est vérifiée par un système avec perte, elle l'est aussi par le système sans perte. Ce résultat nous permet d'envisager la vérification des propriétés de sûreté pour les systèmes à file réactifs embarqués.

Le premier des résultats précédents indique, cependant, que :

la bornitude de la file n'est pas décidable dès que le système à file réactif embarqué dispose de 2 événements mémorisables, et d'un environnement périodique.

Nous introduisons donc une méthode de test de non-bornitude, inspirée de [66], basée sur la recherche d'un cycle itérable qui fait croître la taille de la file. Nous montrons que :

(a) l'itérabilité d'un cycle dans un système à file réactif embarqué,

(b) et le fait qu'il fait croître la taille de la file de mémorisation,

sont décidables, et nous proposons un algorithme pour trouver une réponse à ces questions. Toutefois, bien sûr, cet algorithme n'apporte pas une réponse absolue à la question de la bornitude.

3. Enfin, dans la partie III (page 125), nous considérons la vérification des *applications temps-réel* spécifiées en ELECTRE. Ce sont des applications réactives dont les temps de réaction importent pour leur correction. Le temps n'est alors plus simplement une composante qualitative du système, mais aussi quantitative.

   (a) En nous inspirant de deux approches précédentes [89, 16], nous définissons le langage $\mathcal{H}$-ELECTRE de spécification temporelle des programmes ELECTRE.

   (b) À partir d'un programme ELECTRE et de la spécification temporelle qui lui est associée, nous construisons un *système à file réactif hybride linéaires*, qui emprunte aux *systèmes hybrides (linéaires)* [5, 60, 8, 6, 59, 58] pour la modélisation quantitative du temps.

   (c) Les résultats d'indécidabilité, obtenus dans la partie précédente, s'appliquent naturellement dans ce cadre puisque tout environnement périodique peut être spécifié de façon temporelle. En particulier, le problème de la bornitude n'est pas décidable pour les systèmes à file réactifs hybrides linéaires.

   (d) Nous adaptons donc notre test de non-bornitude aux systèmes à file réactifs hybrides linéaires, en particulier, la recherche des cycles, en nous inspirant de [71]. La

[5]Le fragment (EF) est obtenu en considérant uniquement les modalités existentielles EF et EX de CTL. Ce fragment est noté UB^- dans [33], UB [10] étant la logique «Unified Systems of Branching Time».

décidabilité de l'itérabilité d'un cycle dans un système à file réactif hybride linéaire est un problème ouvert.

Pour terminer, lorsque la bornitude de la file de mémorisation d'un programme ELECTRE est acquise, c'est-à-dire, lorsque le système à file réactif embarqué correspondant est fini, il est malgré tout souvent impossible de vérifier, en pratique, qu'il satisfait aux propriétés exigées. En effet, la taille de son espace d'états est bien souvent rédhibitoire pour l'utilisation d'un parcours exhaustif, outil classique de la vérification des systèmes finis, en vue de vérifier les propriétés du système modélisé. Nous introduisons, dans le dernier chapitre, une technique de réduction du modèle de vérification des programmes ELECTRE, basée sur les *techniques d'ordre-partiel* [82, 93, 50, 48, 69, 79, 49, 46]. Pour cela, nous définissons une relation d'équivalence sur les contenus de la file de mémorisation :

(a) qui peut être calculée en temps linéaire en le nombre d'opérateurs ELECTRE qui apparaîssent dans le programme considéré.

(b) qui, malgré l'abstraction engendrée, préserve les résultats de vérification pour la plupart des propriétés.

Plan du document

Ce document est divisé en trois parties. La **partie I** (page 23) installe le cadre de notre étude. Le premier chapitre (page 25) introduit les définitions et les notations générales.

Le chapitre 2 (page 33) présente les principaux aspects du langage ELECTRE, en mettant l'accent sur la mémorisation des événements, point central de notre étude. Puis, nous y introduisons, les automates à file réactifs et les systèmes à file réactifs, que nous montrons clos par ajout de l'opération *reset* (qui consomme toutes les occurrences mémorisées d'un événement donné en une seule transition). Enfin, nous définissons, dans ce chapitre, les systèmes à file réactifs embarqués, obtenus par le plongement d'un système à file réactif dans son environnement.

La **seconde partie** (page 67) concerne l'étude des problèmes de vérification pour les applications réactives spécifiées par un système à file réactif embarqué. Elle est divisée en trois chapitres. Dans le chapitre 3 (page 69), nous montrons l'indécidabilité des problèmes de vérification pour les systèmes avec deux événements mémorisables, et leur décidabilité pour ceux qui ne disposent pas de plus d'un événement mémorisable. Enfin, nous y étudions le comportement des systèmes à file réactifs embarqués plongés dans des environnement périodiques.

Le chapitre 4 (page 89) est consacré aux *Reset systèmes à file réactifs embarqués (Reset SFR Embarqués)* et aux *Lossy systèmes à file réactifs embarqués (Lossy SFR Embarqués)*, deux extensions du modèle précédent.

Puis, ayant prouvé l'indécidabilité du problème de la bornitude de la file de mémorisation, nous définissons, dans le chapitre 5 (page 107), un test de non-bornitude pour les systèmes à file réactifs embarqués.

Enfin, la troisième et **dernière partie** (page 125) est consacrée aux applications temps-réel modélisée en ELECTRE. Dans le chapitre 6 (page 127), nous définissons le langage $\mathcal{H}$-ELECTRE pour l'introduction des spécifications temporelles des programmes ELECTRE. Puis, nous montrons comment traduire ces spécifications en systèmes à file réactifs hybrides linéaires, pour lesquels les résultats d'indécidabilité du chapitre 3 (page 69) s'appliquent, en particulier, pour le problème de la bornitude. Nous adaptons finalement notre test de non-bornitude à ce nouveau contexte.

Pour finir, le chapitre 7 (page 151) introduit une méthode de réduction du modèle de vérification des programmes Electre dont la file est effectivement bornée.

Première partie

Spécification et modélisation des systèmes réactifs avec mémorisation

Cette première partie installe le cadre de notre étude. Le premier chapitre est consacré aux notions de bases et permet d'introduire la plupart des notations adoptées par la suite.

Nous entrons dans le vif du sujet dans le second chapitre : il est en effet dédié au langage ELECTRE [31, 88, 23] et aux *automates à file réactifs* [23, 92, 21] qui en décrivent la sémantique. Ces deux formalismes servent à modéliser les applications réactives avec mémorisation. Les automates à file réactifs permettent donc de spécifier qu'une occurrence d'événement qui ne peut pas être traitée lorsqu'elle survient, est mémorisée. Elle est ensuite traitée dès que possible, en suivant la politique FIFFO (First In First Fireable Out). Celle-ci consiste à traiter en priorité les occurrences mémorisées, en choisissant la plus ancienne de celles qu'il est possible de prendre en compte. Le choix de l'occurrence traitée s'effectue par l'application de l'ordre FIFO classique, tout en ignorant les occurrences mémorisées qu'il n'est pas possible de traiter.

Nous distinguons deux types d'événements mémorisables :

– les événements *à mémorisation unique*, dont au plus une occurrence est mémorisée à la fois,
– les événements *à mémorisation multiple*, dont toutes les occurrences qui ne peuvent pas être traitées sont mémorisées.

Les événements à mémorisation multiple posent cependant le problème de la bornitude des automates à file réactifs, puisqu'il n'y a, *a priori*, aucune borne sur le nombre d'occurrences mémorisées simultanément. L'existence d'une telle limite repose intrinsèquement non seulement sur l'application modélisée et sur sa rapidité de réaction, mais aussi sur le rythme d'occurrence des événements. Cette dernière composante fait défaut aux programmes ELECTRE et aux automates à file réactifs, qui produisent donc des spécifications *ouvertes*, où seuls les comportements des applications sont modélisés.

Nous enrichissons donc les programmes ELECTRE et les automates à file réactifs de la spécification du mode d'occurrence des événements auxquels ils réagissent : leur *environnement*. Le modèle ainsi obtenu : *les systèmes à file réactifs embarqués*, permet de produire des spécifications *closes* des applications réactives.

Nous montrons par ailleurs, dans ce chapitre, que l'ajout de l'opération *reset* qui consiste à consommer toutes les occurrences mémorisées d'un événement donné en une seule transition, ne modifie en rien le modèle. Ceci nous permet de simuler les événements à mémorisation unique par des événements à mémorisation multiple, et de normaliser ainsi la mémorisation pour les automates à file réactifs.

Chapitre 1

Préliminaires

Nous introduisons, dans ce chapitre, l'ensemble des définitions de base et des notations que nous utilisons tout au long de notre étude. Il se présente en trois sections : la première est dédiée aux notions ensemblistes et aux relations d'ordre. Dans la section 1.2, nous donnons les opérateurs que nous utilisons ensuite sur les mots et les langages. Enfin, la dernière section introduit nos notations pour les systèmes de transitions étiquetés, ainsi que les problèmes de vérification les concernant, que nous étudions dans la suite de notre thèse.

1.1 Ensembles et relations d'ordre

Cette section est inspirée de [41].

1.1.1 Beau-quasi-ordre et bel-ordre-partiel

Soit X un ensemble et $\leq \subseteq X \times X$ une relation binaire sur X. La relation $\leq$ est :
- un *quasi-ordre*[1] *(qo)* si elle est réflexive et transitive,
- un *ordre-partiel (op)* si elle est un quasi-ordre anti-symétrique,
- un *beau-quasi-ordre (bqo)* (resp. *bel-ordre-partiel (bop)*) si elle est un (qo) (resp. (op)) tel que dans toute séquence infinie : $x_0, x_1, x_2, \ldots$ d'éléments de X, il existe deux entiers $i < j$ tels que $x_i \leq x_j$.

Remarque 1.1
Un beau-quasi-ordre (resp. un bel-ordre-partiel) est bien fondé : il n'admet pas de séquence infinie strictement décroissante. ♦

La relation *stricte* $<$, générée par $\leq$, est définie par : $\forall x, x' \in X,\ x < x'$ si $x \leq x'$ et $x' \not\leq x$.

1.1.2 Ensembles clos par le haut et clos par le bas

Soient x un élément de X et $\leq$ un quasi-ordre sur X. L'élément x est *minimal* si $\forall x' \in X,\ x \leq x'$. *L'ensemble clos par le haut* $\uparrow x$ (resp. *L'ensemble clos par le bas* $\downarrow x$) généré par x est défini par : $\uparrow x = \{x' \in X \mid x \leq x'\}$ (resp. $\downarrow x = \{x' \in X \mid x' \leq x\}$). La notion de clôture s'étendent

[1] Les quasi-ordres sont parfois aussi appelés *pré-ordres*.

naturellement aux ensembles générateurs : $\forall X' \subseteq X,\ \uparrow X' = \cup_{x' \in X'} \uparrow x'$ et $\downarrow X' = \cup_{x' \in X'} \downarrow x'$. De plus, $\uparrow \emptyset = \emptyset$ et $\downarrow \emptyset = \emptyset$.

Une *base finie* pour un ensemble clos par le haut $X' \subseteq X$ est un ensemble fini $Y \subseteq X$ tel que $\uparrow Y = X'$.

Remarque 1.2
Le complémentaire d'un ensemble clos par le haut est clos par le bas, et vice-versa. ♦

Par la définition de la clôture vers le haut, nous avons la propriété de distributivité vis-à-vis de l'union :

Lemme 1.3
Soient X un ensemble, $\leq$ un quasi-ordre sur X et, $X', X'' \subseteq X$ deux sous-ensembles de X :

$$\uparrow(X' \cup X'') = \uparrow X' \cup \uparrow X''$$

♦

Lemme 1.4 (Higman [62])
Si $\leq$ est un beau-quasi-ordre, tout ensemble X clos par le haut admet une base finie. ♦

Preuve. L'ensemble des éléments minimaux de X forment une base puisque $\leq$ est bien fondé. Ces éléments minimaux sont en nombre fini, sinon ils forment une séquence infinie d'éléments incomparables, contredisant le fait que $\leq$ est un beau-quasi-ordre. ♦

Lemme 1.5
Toute séquence infinie et croissante d'ensembles clos par le haut : $X_0 \subseteq X_1 \subseteq X_2 \subseteq \ldots$ converge : $\exists k \in \mathbb{N}$ t.q. $X_k = X_{k+1} = X_{k+2} = \ldots$. ♦

Preuve. Considérons une sous-séquence infinie et strictement croissante : $X_{i_0} \subset X_{i_1} \subset X_{i_2} \subset \ldots$ qui est supposée être un contre-exemple. Alors, nous construisons la séquence des éléments définis par : $x_k \in X_{i_k} \setminus X_{i_{k-1}}$ qui, par la définition d'un beau-quasi-ordre contient deux éléments $x_j \leq x_k$ avec $j < k$. Puisque X_{i_j} est clos par le haut, $x_j \in X_{i_j}$ implique que $x_k \in X_{i_j}$, ce qui est impossible par hypothèse. ♦

1.2 Langages

Les définitions et notations introduites dans cette section proviennent principalement de [76].

Un *alphabet* Σ est un ensemble fini de symboles. Un *mot* u, sur Σ, est une séquence, finie ou infinie, de symboles de Σ. L'ensemble des mots finis sur Σ est noté Σ^*, celui des mots infinis est noté Σ^ω. Enfin, $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ représente l'ensemble des mots sur l'alphabet Σ. Un *langage* $\mathcal{L}$ sur Σ est un sous-ensemble de Σ^∞ : $\mathcal{L} \subseteq \Sigma^\infty$.

La *longueur* d'un mot $u \in \Sigma^\infty$ est notée $|u|$. Si $u = u_1 \dots u_n$ est un mot fini, $|u| = n$. Si u est infini, $|u| = \omega$. Le *mot vide* (de longueur égale à zéro) est noté ε. Soit $u \in \Sigma^\infty$, la $n^{ième}$ *puissance* de u est définie par : $u^n = u.u^{n-1}$, et $u^0 = \varepsilon$.

Le $i^{ième}$ *symbole* de u est noté u_i avec $i \in \{1, \dots, |u|\}$. Le *nombre d'occurrences* de $a \in \Sigma$ dans un mot $u \in \Sigma^\infty$ est $|u|_a = Card(\{i \in \{1, \dots, |u|\} \,|\, u_i = a\})$. L'*alphabet d'un mot* $u \in \Sigma^\infty$ est l'ensemble des symboles qui composent le mot : $\Sigma_u = \{u_i \,|\, i \in \{1, \dots, |u|\}\}$.

Soient $u \in \Sigma^*$ un mot fini, et $v \in \Sigma^\infty$ un mot fini ou infini. Le mot u est un *préfixe* de v s'il existe un mot $v' \in \Sigma^\infty$ tel que $v = u.v'$. La relation $\leq \subseteq \Sigma^* \times \Sigma^\infty$ est l'ordre-partiel, nommé *ordre préfixe*, défini par : $u \leq v$ si et seulement si u est un préfixe de v. La relation $^{-1}$ est définie pour tous les mots $u \in \Sigma^*$ et $v \in \Sigma^\infty$, tels que $u \leq v$, par : $v = u.(u^{-1}v)$. Enfin, l'*ensemble des préfixes* du mot v est noté *prefix*(v).

L'opérateur *shuffle* $\sqcup\!\sqcup$ associe à deux mots finis $u, v \in \Sigma^*$ l'ensemble des mots définis par : $u \sqcup\!\sqcup v = \{u_1 v_1 \dots u_k v_k \,|\, u = u_1 \dots u_k,\ v = v_1 \dots v_k \text{ et } u_i, v_i \in \Sigma^*\}$. La relation *sous-mot*, notée $\preccurlyeq$, est définie par : $\forall u, v \in \Sigma^*,\ u \preccurlyeq v \Leftrightarrow v \in (u \sqcup\!\sqcup \Sigma^*)$.

Soit $\Sigma = \{a_1, \dots, a_n\}$ un alphabet. Le *morphisme de Parikh* [83] est l'application de $(\Sigma^*, .)$ dans $(\mathbb{N}^n, +)$ définie par : $\forall u \in \Sigma^*,\ \Psi(u) = \begin{pmatrix} |u|_{a_1} & \dots & |u|_{a_n} \end{pmatrix}$. L'application inverse du morphisme de Parikh définit l'ensemble des mots : $\Psi^{-1}\begin{pmatrix} x_1 & \dots & x_n \end{pmatrix} = ((a_1)^{x_1} \sqcup\!\sqcup \dots \sqcup\!\sqcup (a_n)^{x_n})$.

1.2.1 Propriétés de fermeture des langages

Dans la suite, nous nous intéressons particulièrement à la fermeture vers le haut, relativement à la relation $\preccurlyeq$, de langages de mots finis pour lesquels nous avons les propriétés suivantes :

Lemme 1.6
Soient Σ un alphabet fini et, $L \subseteq \Sigma^*$ et $L' \subseteq \Sigma^*$ deux langages. Nous avons :

1. $\uparrow(L.L') = (\uparrow L).(\uparrow L')$,
2. $\uparrow(L \sqcup\!\sqcup L') = (\uparrow L) \sqcup\!\sqcup (\uparrow L')$.

♦

1.2.2 Fiffo-soustraction

Soient u et v deux mots finis sur un alphabet Σ. La *fiffo-soustraction* de u à v, définie si et seulement si $\Psi(u) \leq \Psi(v)$, et notée $u^{\ominus 1}v$, est définie par :

1. $\varepsilon^{\ominus 1}v = v$,
2. $\forall a \in \Sigma,\ a^{\ominus 1}v = v'.v'',$ avec $v', v'' \in \Sigma^*,\ a \notin \Sigma_{v'}$ et $v = v'.a.v''$,
3. $\forall a \in \Sigma,\ (a.u)^{\ominus 1}v = u^{\ominus 1}(a^{\ominus 1}v)$.

L'opérateur *fiffo-soustraction* retire donc les premières occurrences de symboles de u qui apparaîssent dans v. Dans la suite, nous notons $u^{\ominus n}v$ l'opération $(u^n)^{\ominus 1}v$.

Lemme 1.7
Soient u, v et w trois mots finis sur un alphabet Σ, nous avons :

1. $\forall a, b \in \Sigma,\ a^{\ominus 1}b^{\ominus 1}u = b^{\ominus 1}a^{\ominus 1}u$,

2. Notons $\Sigma = \{a_1, \ldots, a_k\}$, nous avons : $u^{\ominus 1} v = a_1^{\ominus |u|_{a_1}} \ldots a_k^{\ominus |u|_{a_k}} v$,
3. Si $\Psi(u) = \Psi(v)$, alors : $u^{\ominus 1} w = v^{\ominus 1} w$,
4. Si $u \preccurlyeq v$, alors : $w^{\ominus 1} u \preccurlyeq w^{\ominus 1} v$,
5. $(v^{\ominus 1} u).w = v^{\ominus 1}(u.w)$,
6. $w^{\ominus 1}(u^{\ominus 1} v) = (u.w)^{\ominus 1} v$.

♦

Le point (1) est une conséquence directe de la définition de $^{\ominus 1}$. Les points (2) et (3) expriment que l'ordre des symboles dans u est sans effet sur le résultat de l'opération $^{\ominus 1}$: seul le nombre de chaque symbole compte. Ces deux résultats découlent du premier : il suffit de permutter les symboles de u pour obtenir le résultat. Le point (4) exprime la monotonicité de $^{\ominus 1}$ vis-à-vis de $\preccurlyeq$. Les trois derniers points sont des conséquences directes des deux premiers.

1.3 Vérification des systèmes de transitions étiquetés

1.3.1 Systèmes de transitions étiquetés

Un *système de transitions étiqueté* [9] est un quadruplet $S = (Q, q_0, L, \rightarrow)$ où Q est un ensemble d'états, $q_0 \in Q$ est l'état initial de S, L est un ensemble d'étiquettes et $\rightarrow \subseteq Q \times L \times Q$ définit la relation de transition de S.

Pour un système de transitions étiqueté S, nous notons $Pred^l(q) = \{q' \in Q \mid q' \xrightarrow{l} q\}$ (resp. $Succ^l(q) = \{q' \in Q \mid q \xrightarrow{l} q'\}$) l'ensemble des *prédecesseurs par l'étiquette* $l \in L$ (resp. *successeurs par l'étiquette* $l \in L$) d'un état $q \in Q$. Ces applications s'étendent naturellement à des ensembles d'états : $\forall Q' \subseteq Q$, $Pred^l(Q') = \bigcup_{q \in Q'} Pred^l(q)$, et $Succ^l(Q') = \bigcup_{q \in Q'} Succ^l(q)$.

Notons $in(q)$ (resp. $out(q)$) l'*ensemble des étiquettes entrantes* (resp. *sortantes*) d'un état $q \in Q$, défini par : $in(q) = \{l \in L \mid \exists q' \in Q,\ q' \xrightarrow{l} q\}$ (resp. $out(q) = \{l \in L \mid \exists q' \in Q,\ q \xrightarrow{l} q'\}$).

L'ensemble des *prédecesseurs* d'un état $q \in Q$ d'un système de transitions étiqueté est défini par $Pred(q) = \bigcup_{l \in in(q)} Pred^l(q)$. De même nous définissons l'ensemble des *successeurs* par $Succ(q) = \bigcup_{l \in out(q)} Succ^l(q)$.

Définissons $Pred^n(q) = Pred(Pred^{n-1}(q))$ l'*ensemble des prédécesseurs en n pas* de $q \in Q$, avec $Pred^0(q) = q$. Le *pred-ensemble* de $q \in Q$ est l'ensemble de tous les prédécesseurs de q, défini par $Pred^*(q) = \bigcup_{i \geq 0} Pred^i(q)$. De même, nous définissons $Succ^n(q) = Succ(Succ^{n-1}(q))$ l'*ensemble des successeurs en n pas* de $q \in Q$, et, $Succ^0(q) = q$. Alors, $Succ^*(q) = \bigcup_{i \geq 0} Succ^i(q)$ est le *succ-ensemble* de q.

Un système de transitions étiqueté est *déterministe* si $\forall q \in Q$, $\forall l \in L$, il existe au plus un état $q' \in Q$ tel que $q \xrightarrow{l} q'$, i.e. $Card(Succ^l(q)) \leq 1$. Il est *fini* si Q, L et $\rightarrow$ sont finis.

Un *chemin* dans un système de transitions étiqueté S est une séquence de transitions finie ou infinie : $q_0 \xrightarrow{l_0} \ldots \xrightarrow{l_{n-1}} q_n \xrightarrow{l_n} \ldots$. La *longueur d'un chemin* est donnée par le nombre de transitions qui le forment. Un chemin est fini si sa longueur est finie. L'*ensemble des chemins* (finis ou infinis) d'un système de transitions S est noté $[\![S]\!]$. Un chemin est *maximal* s'il est infini, ou s'il se termine dans un état q sans successeur : $Succ(q) = \emptyset$.

Nous notons $\rightarrow^*$ la fermeture réflexive et transitive de $\rightarrow$. Un état $q' \in Q$ est *accessible* depuis $q \in Q$ s'il existe un chemin de q à q' : $q \rightarrow^* q'$. L'*ensemble des états accessibles* (ou *Reachability Set*) d'un système de transitions étiqueté S est $RS(S) = \{q \in Q \mid q_0 \rightarrow^* q\} = Succ^*(q_0)$.

L'*arbre d'accessibilité* (ou *Reachability Tree*) de S est la structure arborescente notée $RT(S)$, définie par :

– la racine est le nœud $n_0 : q_0$,
– chaque noeud $n : q$ admet un fils $n' : q'$ par successeur de q.

1.3.2 Automates, reconnaissabilité et logique

Un *automate fini* est un quintuplet $A = (Q, q_0, Q_F, L, \rightarrow)$ où $(Q, q_0, L, \rightarrow)$ est un système de transitions étiqueté fini et $Q_F \subseteq Q$ est un ensemble d'*états terminaux*.

Le *langage reconnu* par un automate $A = (Q, q_0, Q_F, L, \rightarrow)$ est l'ensemble des mots $\sigma \in L^*$ tel qu'il existe un chemin fini de q_0 à un état de Q_F étiqueté par σ :

$$\mathcal{L}(A) = \{\sigma \in L^* \mid \exists q_F \in Q_F \text{ t.q. } q_0 \xrightarrow{\sigma}{}^* q_F\}$$

Un langage $\mathcal{L}$ est *rationnel* ou *reconnaissable* si et seulement si il est reconnu par un automate fini. De plus, $\mathcal{L}$ est *effectivement reconnaissable* s'il est possible de calculer A.

Enfin, nous considérons dans la suite la satisfaction des formules des logiques temporelles Ltl [86, 32] et Ctl [25, 32].

– La logique Ltl permet d'exprimer des propriétés linéaires d'un système de transitions. Ses formules ϕ sont construites de la façon suivante :

$$\begin{aligned} \phi \ ::= \ & p_1 \mid p_2 \mid \ldots && \text{(propositions atomiques)} \\ & \neg\,\phi \mid \phi \vee \phi && \text{(opérateurs booléens)} \\ & \mathsf{X}\,\phi \mid \phi\,\mathsf{U}\,\phi && \text{(opérateurs temporels)} \end{aligned}$$

Notons qu'il existe deux opérateurs : «always», noté G, et «eventually», noté F qui sont des abbréviations courantes pour les formules : $\mathsf{F}\phi \equiv \mathit{true}\,\mathsf{U}\,\phi$ et $\mathsf{G}\phi \equiv \neg\,\mathsf{F}\,\neg\,\phi$.

Les formules de Ltl sont interprétées sur des mots infinis, et plus généralement, des langages de mots infinis. Pour un mot infini σ, nous notons $\sigma \models \phi$ si la formule ϕ est *satisfaite* par σ, et de la même façon, $\mathcal{L} \models \phi$ si $\forall \sigma \in \mathcal{L}$, $\sigma \models \phi$. Nous dirons alors qu'un système de transitions étiqueté S *satisfait* une formule ϕ, que nous notons $S \models \phi$, si et seulement si $[\![S]\!] \models \phi$. Voici quelques exemples de formules Ltl et des séquences qui les satisfont ou non :

$b\mathbf{a}bbbb\ldots \models \mathsf{X}\,a$	$bbb\mathbf{a}aba\ldots \models \mathsf{F}\,a$	$\mathbf{aaaa}\ldots \models \mathsf{G}\,a$	$\mathbf{aaa}\mathbf{b}bb\ldots \models a\,\mathsf{U}\,b$
«next a»	«eventually a»	«always a»	«a until b»
$a\mathbf{b}baab\ldots \not\models \mathsf{X}\,a$	$\mathbf{bbbbb}\ldots \not\models \mathsf{F}\,a$	$aaa\mathbf{b}bab\cdots \not\models \mathsf{G}\,a$	$\mathbf{aaaaaa}\ldots \not\models a\,\mathsf{U}\,b$

Les formules de Ltl permettent de caractériser les propriétés d'une exécution donnée d'un système.

– La logique Ctl permet d'exprimer les propriétés arborescentes d'un système de transitions.

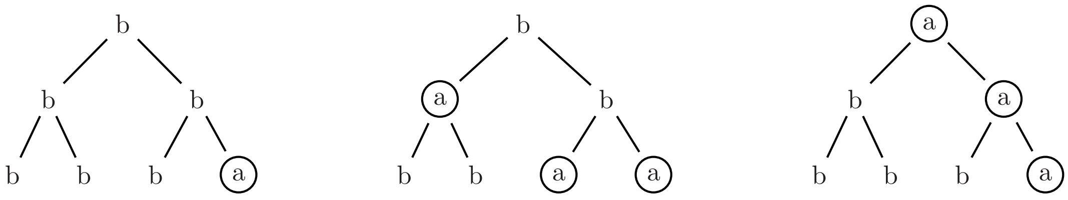
Ses formules sont construites de la façon suivante :

$$\begin{array}{lll} \phi & ::= p_1 \,|\, p_2 \,|\, \ldots & \text{(propositions atomiques)} \\ & \quad \neg\, \phi \,|\, \phi \vee \phi & \text{(opérateurs booléens)} \\ & \quad \mathsf{E}\, \phi\, \mathsf{U}\, \phi \,|\, \mathsf{A}\, \phi\, \mathsf{U}\, \phi & \text{(opérateurs temporels)} \end{array}$$

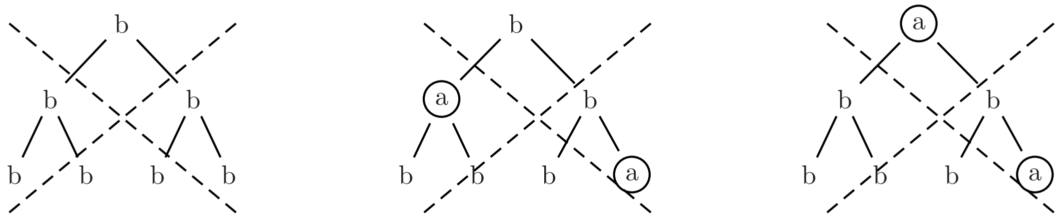
Les abbréviations «exists eventually», noté EF, «forall eventually», noté AF, «exists always», noté EG et «forall always», noté AG sont couramment utilisés. Elles correspondent aux formules :

- $\mathsf{EF}\, \phi \;\equiv\; \mathsf{E}\, \mathit{true}\, \mathsf{U}\, \phi$
- $\mathsf{AF}\, \phi \;\equiv\; \mathsf{A}\, \mathit{true}\, \mathsf{U}\, \phi$
- $\mathsf{EG}\, \phi \;\equiv\; \neg\, (\mathsf{AF}\, \neg\, \phi)$
- $\mathsf{AG}\, \phi \;\equiv\; \neg\, (\mathsf{EF}\, \neg\, \phi)$

Les formules de CTL sont interprétées sur des arbres. Pour un arbre a, nous notons $a \vDash \phi$ si la formule ϕ est *satisfaite* par a. Nous dirons alors qu'un système de transitions étiqueté S *satisfait* ϕ, ce que nous notons $S \vDash \phi$, si et seulement si $RT(S) \vDash \phi$. Voici quelques exemples de formules CTL avec des arbres qui les satisfont (au-dessus) et d'autres qui ne les satisfont pas (en-dessous) :



«exists eventually a» «forall eventually a» «exists always a»



Remarque 1.8

Lorsqu'un système de transitions étiqueté S satisfait une formule LTL ϕ, alors, chacune des exécutions de S satisfait ϕ **indépendamment** des autres exécutions. Dans le cas de CTL, cette indépendance entre les exécutions n'est pas exprimable. Ainsi, la propriété «infiniment souvent a», qui s'écrit $\mathsf{GF}a$ en LTL, et qui est vérifiée par toute séquence infinie qui contient une infinité de a, n'est pas expressible avec CTL, à cause de la quantification E ou A qui précède tout opérateur F et G. ♦

1.3.3 Problèmes d'intérêt sur les systèmes de transitions étiquetés

Nous dressons maintenant, la liste des problèmes de vérification que nous considérons dans la suite de notre étude. Chaque problème est défini par une *instance* : les éléments sur lesquels se pose le problème, et par une *question* qui détermine le problème.

- Le problème de la bornitude (B) :
  Instance : Un système de transitions étiqueté S.
  Question : $RS(S)$ est-il fini ?
- Le problème de l'accessibilité (R) :
  Instance : Un système de transitions étiqueté S et un état $q \in Q$.
  Question : Est-ce que $q \in RS(S)$?
- Le problème de la couverture (C) :
  Instance : Un système de transitions étiqueté S, une relation de quasi-ordre $\leq \subseteq Q \times Q$, sur les états de S, et deux états $q, q' \in Q$.
  Question : Existe-t-il $q'' \in RS(S)$ tel que $q \rightarrow^* q''$ et $q' \leq q''$?
- Le problème de la terminaison (T) :
  Instance : Un système de transitions étiqueté S.
  Question : Est-ce que tout chemin σ dans $[\![S]\!]$ est fini ?
- Le problème du model-checking de LTL (MC-LTL) :
  Instance : Un système de transitions étiqueté S et une formule de LTL ϕ.
  Question : Est-ce que $S \vDash \phi$?
- Le problème du model-checking de CTL (MC-CTL) :
  Instance : Un système de transitions étiqueté S et une formule de CTL ϕ.
  Question : Est-ce que $S \vDash \phi$?

En plus des problèmes exprimés ci-dessus, il reste deux problèmes importants liés à la vérification des systèmes infinis. De façon générale, dès qu'un système de transitions étiqueté S décrit la sémantique d'un modèle muni d'une donnée à valeur dans un domaine infini : pile, file, variable non bornée, etc., l'espace de ses états accessibles, $RS(S)$ est infini. La résolution des problèmes de vérification s'appuie pourtant essentiellement sur la possibilité de représenter cet ensemble. C'est pourquoi, pour les systèmes infinis, il est important de déterminer une représentation finie pour l'ensemble, éventuellement infini, $RS(S)$. Nous considérons deux problème distincts : l'existence d'une telle représentation finie, et la possibilité de la calculer.

- Le problème de la reconnaissabilité (RP) :
  Instance : Un système de transitions étiqueté S.
  Question : $RS(S)$ est-il reconnaissable ?
- Le problème de la reconnaissabilité effective (ERP) :
  Instance : Un système de transitions étiqueté S.
  Question : $RS(S)$ est-il effectivement reconnaissable ?

Chapitre 2

Le langage Electre et les automates à file réactifs

Les systèmes réactifs ont pour but de contrôler des dispositifs physiques, en répondant à leurs requêtes par les actions appropriées. Il s'agit typiquement de signaux en provenance de capteurs (par exemple la détection d'une température trop élevée) qui informent de l'état du dispositif physique, auquel le système de contrôle répond (ici, en réduisant l'intensité du chauffage) dans le but de maintenir le dispositif physique dans un état stable. Le langage ELECTRE [31, 88, 23] permet de définir le comportement d'une application réactive lors de la réception des signaux provenant de son environnement. Un programme ELECTRE consiste donc en la description de l'activation, la préemption et la terminaison des tâches qui le composent lors des occurrences de ces signaux, indépendamment de la façon qu'ont ceux-ci de survenir.

ELECTRE se distingue d'autres langages réactifs (dits *synchrones*) tels que ARGOS [75] (proche des STATECHARTS [56]), ESTEREL [13, 34], LUSTRE [54] ou SIGNAL [44] par ses hypothèses *asynchrones* fondamentales. Celles-ci expriment l'existence d'un grain de temps assez fin pour distinguer deux événements consécutifs dans l'évolution d'un programme ELECTRE. Par conséquent, les tâches qui composent le programme ont une durée non nulle : leurs débuts et leurs fins se produisent à deux instants distinguables. Une occurrence d'événement peut donc survenir pendant l'exécution d'une tâche. Or, l'exécution de cette tâche peut ne pas pouvoir être interrompue pour traiter l'occurrence qui survient, par exemple lorsque cette tâche gère une situation critique pour le système modélisé. Certains événements peuvent être ignorés lorsqu'ils ne sont pas attendus, mais cela n'est pas correct pour tous les événements. La solution consiste alors à mémoriser les occurrences inattendues, pour les traiter ultérieurement. Nous distinguons alors deux types d'événements :

- ceux qui signalent l'état d'une partie du système modélisé, par exemple la détection d'une température trop élevée, et s'apparentent donc à une information binaire : la présence ou l'abscence d'un signal. Il est suffisant de mémoriser une seule occurrence de ces événements lorsqu'ils ne peuvent pas être pris en compte. C'est en effet, par exemple, la détection de la température trop élevée qui constitue l'information, et non pas le nombre de fois où le capteur est activé avant que le problème soit résolu.
- et ceux dont chaque occurrence est importante, par exemple, un événement qui signale

le passage d'un objet sur un tapis roulant d'une chaîne de production. Dans ce cas, la correction du programme dépend de sa capacité à traiter chaque occurrence, et il faut donc mémoriser toutes les occurrences qui ne peuvent pas être traitées lorsqu'elles surviennent.

Le point fondamental pour cette étude réside dans le fait que dans ce dernier cas, il n'existe *a priori* aucune limite quant au nombre d'occurrences qui peuvent être mémorisées simultanément. Pourtant, une telle limite existe dans toute application réelle : aucun dispositif physique ne peut contenir un nombre infini, ni même fini mais non borné, d'objets. Dans ce chapitre nous considérons la possibilité de déterminer cette borne.

En section 2.1 (page 34) nous présentons le langage Electre [31, 88, 23] en insistant principalement sur les aspects liés à la mémorisation des événements, qui nous concerne particulièrement. En fin de section nous montrons comment un programme est compilé pour obtenir un *automate à file réactif (AFR)* [23, 92, 21] qui en donne la sémantique. La section 2.2 (page 41) est donc naturellement consacrée à la présentation de ces automates et de leurs principales propriétés, en particulier la politique Fiffo (First In First Fireable Out) pour la gestion de la file de mémorisation, qui est bien adaptée aux systèmes réactifs. Nous abordons ensuite, en section 2.3.2, page 51, la spécification de l'*environnement* d'un automate à file réactif : la façon dont les événements surviennent. C'est en effet un point déterminant pour l'étude de la bornitude du nombre d'occurrences mémorisées, dont la spécification fait volontairement défaut aux programmes Electre et aux automates à file réactifs. Le modèle correspondant est appelé «*systèmes à file réactifs embarqués (SFR Embarqués)*». Finalement, en section 2.4 (page 54), nous normalisons la mémorisation dans les automates à file réactifs en introduisant les *Reset AFR*, simplifiant ainsi le modèle et nos argumentations.

2.1 Le langage Electre

Le langage Electre [31, 88, 23] sert à la modélisation de systèmes réactifs en indiquant comment les tâches qui les composent évoluent au gré des occurrences d'événements.

2.1.1 Un exemple : les lecteurs/écrivains

Nous considérons dans la suite l'exemple d'une ressource à laquelle il est possible d'accéder en lecture comme en écriture, suivant le schéma classique des lecteurs/écrivains [27]. L'application est composée de deux serveurs d'accès en lecture, et d'un serveur pour l'accès en écriture. Les requêtes d'accès en lecture, au premier et au second serveur, sont signifiées par les événements $\mathtt{r}_1$ et $\mathtt{r}_2$, respectivement. Une demande d'écriture est véhiculée par l'événement `w`. Les tâches qui assurent le service de lecture sont notées $\mathtt{R}_1$ et $\mathtt{R}_2$, et celle qui gère l'accès en écriture est nommé `W`. Finalement, il n'est pas possible d'accéder simultanément en lecture et en écriture à la ressource, alors que les deux serveurs en lecture peuvent satisfaire leurs requêtes simultanément. Pour éviter les situations de famine, après chaque accès en lecture (accès de $\mathtt{R}_1$ seul, de $\mathtt{R}_2$ seul ou accès simultanés de $\mathtt{R}_1$ et $\mathtt{R}_2$), et après chaque accès en écriture, le système examine à nouveau l'ensemble des requêtes et répond à la plus ancienne. La suite de cette section est consacrée à la description du langage Electre pour parvenir à la programmation de cette application.

2.1.2 Syntaxe du langage Electre sur l'exemple des lecteurs/écrivains

Nous introduisons brièvement la syntaxe du langage ELECTRE en nous appuyant sur l'exemple des lecteurs/écrivains décrit dans la section précédente. La grammaire complète du langage ELECTRE est présentée en annexe A, section A.1, page 193.

Éléments principaux de la syntaxe

Les programmes ELECTRE sont construits à partir des *modules* qui composent l'application modélisée, des *événements* auxquels elle doit réagir, et des opérateurs du langage. Rappelons qu'ELECTRE est un langage asynchrone. Ceci a pour conséquence que :

– la durée d'exécution d'un module est finie et non nulle,

– deux occurrences successives d'un même événement ne peuvent pas être simultanées.

Considérons à nouveau l'exemple de la section précédente. Nous souhaitons que cette application ait le comportement suivant :

1. attente d'une requête $\mathtt{r}_1$, $\mathtt{r}_2$ ou $\mathtt{w}$,
2. traitement de la requête reçue, $\mathtt{r}_1$ et $\mathtt{r}_2$ pouvant être traitées en parallèle,
3. reprise au point (1) lorsque le traitement est terminé.

La figure 2.1 présente le programme ELECTRE qui modélise l'application des lecteurs/écrivains.

```
loop [
      await {
           { #r1:R1 ||| #r2:R2}
        |
           #w:W
      }
]
```

FIG. 2.1 – Le programme ELECTRE pour les lecteurs/écrivains.

Le traitement des requêtes (point (2) ci-dessus) consiste simplement à lancer le module qui effectue l'opération requise. Pour la requête $\mathtt{r}_1$, ceci revient à exécuter $\mathtt{R}_1$ qui va effectuer les lectures souhaitées. Ce comportement est spécifiée par le programme : «$\mathtt{r}_1 : \mathtt{R}_1$». De même, les traitements des requêtes $\mathtt{r}_2$ et $\mathtt{w}$ s'effectue en exécutant le module correspondant : $\mathtt{R}_2$ et $\mathtt{W}$, respectivement. Donc ceci est indiqué par les programmes : «$\mathtt{r}_2 : \mathtt{R}_2$» et «$\mathtt{w} : \mathtt{W}$», respectivement.

Après que le traitement des requêtes soit achevé, l'exécution du programme reprend à son début pour traiter de nouvelles requêtes (point (3) ci-dessus). L'instruction «`loop`» permet de spécifier ce comportement. Ainsi, lorsque la structure identifiée entre crochets achève son exécution, elle en démarre immédiatement une nouvelle. L'application ainsi spécifiée propose donc, infiniment longtemps, de traiter les requêtes.

L'attente d'une requête (point (1) ci-dessus) est exprimée par l'opérateur ELECTRE «`await`», qui modélise, de façon plus générale, la préemption. Dans le cas présent, le module qui est implicitement préempté est celui qui modélise la tâche de fond de l'application. Il n'est habituellement pas représenté dans les programmes ELECTRE.

Nous utilisons la composition pour représenter l'attente d'une occurrence de r_1 ou r_2 ou w. La composition parallèle, notée « ||| », indique que les traitements associés aux occurrences de r_1 et de r_2 (c'est à dire les exécutions de R_1 et de R_2 respectivement) peuvent se dérouler en parallèle. L'exécution de la structure d'événements, identifiées par les accolades, débute avec une occurrence de r_1 ou r_2, et s'achève dès que possible. C'est à dire, en considérant que r_1 survient avant r_2 :

- si r_2 se produit avant la fin de R_1, la structure achève son exécution avec le module (R_1 ou R_2) qui se termine le dernier.
- si r_2 ne survient pas avant que R_1 soit terminé, l'exécution de la structure se termine en même temps que celle de R_1.

La composition exclusive, représentée par l'opérateur « | », indique que seule l'une des structures composées sera exécutée. Si r_1 ou r_2 se produit avant w, l'exécution du module R_1 débute, et il reste possible de traiter une occurrence de r_2. Par contre, les occurrences de w n'ont plus la possibilité de lancer l'exécution de W. À l'inverse, si w survient en premier, W commence son exécution, mais les occurrences de r_1 et r_2 qui surviennent alors ne peuvent pas lancer les exécutions de R_1 et R_2.

Cependant, ces occurrences de r_1, r_2 et w qui surviennent quand le système ne peut pas les traiter ne doivent pas être perdues. Elles modélisent en effet des requêtes qui, si elles ne peuvent pas être satisfaites immédiatement, doivent malgré tout l'être ultérieurement. Ces occurrences doivent donc être mémorisées.

La mémorisation des événements

Le langage ELECTRE propose trois types d'événements distingués par la gestion reservée à leurs occurrences qui ne peuvent pas être traitées lorsqu'elles surviennent. Nous considérons donc une occurrence d'événement inattendue, et nous décrivons la façon dont elle gérée suivant son type.

- Les événements *fugaces*, identifiés par l'opérateur « @», ne sont jamais mémorisés. Leurs occurrences qui ne peuvent pas être traitées sont alors simplement perdues. Il s'agit généralement d'événements qui donnent une information éphémère dont la validité se limite au moment de leur occurrence. Par exemple, un événement qui signale le retour du système dans un état stable après la correction d'une erreur, n'est valable qu'au moment où il survient : le système peut se trouver confronté à un nouveau problème avant qu'une occurrence mémorisée de cet événement n'ait pu être traitée.
- Les événements *à mémorisation unique* (gestion appliquée par défaut) servent à modéliser des informations binaires non éphémères. Une occurrence, au plus, de ces événements est présente dans la mémoire du programme. Ainsi, l'information modélisée est la présence ou l'abscence d'un signal. Il s'agit par exemple de la détection d'une température trop élevée par un capteur. Tant que le programme n'agit pas pour la réduire, la température reste trop élevée, et il faut donc mémoriser cette occurrence pour la traiter ultérieurement.
- Enfin, les événements *à mémorisation multiple*, spécifiés par l'opérateur « #», sont mémorisés **à chaque fois** qu'il ne peuvent pas être traités lorsqu'ils surviennent. C'est le cas des requêtes de l'application des lecteurs/écrivains (présentée en section 2.1.1, page 34) :

aucune requête ne doit être ignorée par l'application.

Remarque 2.1
Les événements à mémorisation multiple revêtent un caractère tout particulier au sein de notre étude puisque ce sont eux qui la motivent. En effet, le leitmotiv de notre thèse est la bornitude des programmes Electre. Ce problème ne présente un réel interêt que pour des programmes disposant d'événements à mémorisation multiple. ♦

Remarque 2.2
Le type d'un événement : fugace, à mémorisation unique ou à mémorisation multiple est **global au programme** et ne dépend en aucun cas de son exécution. Sinon, il serait délicat de définir une sémantique pour les programmes ELECTRE. En effet, quel traitement doit alors être appliqué aux occurrences mémorisées d'un événement qui serait localement fugace ? ♦

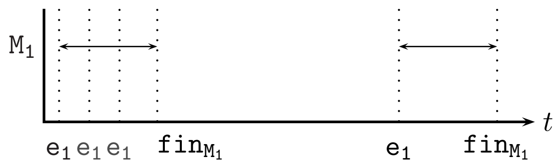
La gestion des occurrences mémorisées

Les occurrences qui sont mémorisées doivent ensuite être traitées. Le langage ELECTRE donne la **priorité au traitement des occurrences mémorisées** vis-à-vis des nouvelles occurrences qui surviennent. De plus, les occurrences mémorisées sont traitées **dès que possible** en donnant la **priorité à la plus ancienne**. Intuitivement, lors de son exécution, un programme ELECTRE commence par chercher une occurrence mémorisée qu'il peut traiter dans son état courant. Par exemple, le programme des lecteurs/écrivains (figure 2.1, page 35), dans son état de départ, examine sa mémoire à la recherche d'une occurrence de $\mathtt{r}_1$ ou de $\mathtt{r}_2$ ou de $\mathtt{w}$. S'il en existe une, il choisit alors la plus ancienne et la traite. Il arrive alors dans un nouvel état où à nouveau il cherche une occurrence mémorisée qu'il lui est possible de traiter. S'il n'y a aucune occurrence mémorisée qu'il peut traiter, il débute l'exécution des instructions correspondant à cet état. Nous développons à nouveau ce point dans la section 2.2, en particulier la définition 2.11 (page 44).

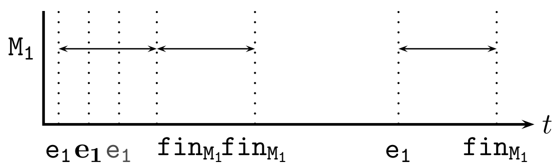
Reprenons notre exemple des lecteurs/écrivains. Il peut traiter les occurrences de $\mathtt{r}_1$,$\mathtt{r}_2$ et $\mathtt{w}$. Supposons que l'occurrence mémorisée la plus ancienne soit une occurrence de $\mathtt{r}_1$. Lors de sa prise en compte, le programme change d'état, où, par la définition des opérateurs « ||| », et « | », il peut maintenant traiter les occurrences de $\mathtt{r}_2$ uniquement. S'il y a une occurrence mémorisée de $\mathtt{r}_2$, il consomme la plus ancienne. Sinon, il attend une nouvelle occurrence de $\mathtt{r}_2$, ou la fin du module $\mathtt{R}_1$ démarré par le traitement de $\mathtt{r}_1$.

Les chronogrammes de la figure 2.2 (page 38) montrent les différences entre les trois types d'événements. Les occurrences de $\mathtt{e}_1$ qui apparaîssent en gras ($\mathbf{e_1}$) sont mémorisées puis traitées par la suite. Au contraire, les occurrences de $\mathtt{e}_1$ qui sont grisées ($\mathtt{e}_1$) ne sont pas traitées lors de l'exécution du programme (elles sont perdues). Enfin, les événements $\mathtt{fin}_{\mathtt{M}_1}$ indiquent la fin de l'exécution du module $\mathtt{M}_1$.

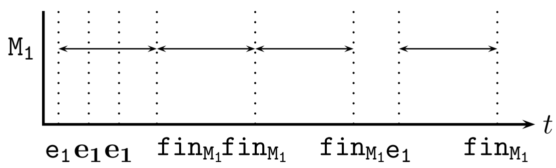
Remarque 2.3
Notons la différence entre le *traitement* d'une occurrence d'événement, qui consiste à répondre à cette requête, et la *prise en compte* d'une occurrence d'événement qui correspond à gérer cette occurrence.



(a) fugace : `loop [ await @e1 : M1 ]`.



(b) m´morisation unique : `loop [ await e1 : M1 ]`.



(c) mémorisation multiple : `loop [ await #e1 : M1 ]`.

FIG. 2.2 – Différentes gestions de la mémorisation des événements.

Lors du *traitement* d'une occurrence d'événement, l'action qui lui est associée est exécutée. Par exemple, pour notre application des lecteurs/écrivains (section 2.1.1, page 34), le traitement d'une occurrence de $\texttt{r}_1$ déclenche l'exécution du module $\texttt{R}_1$.

Lors de sa *prise en compte*, un événement est :
- soit traité, immédiatement ou de façon différée,
- sinon, c'est à dire, lorsque son traitement est impossible, il est soit mémorisé, soit ignoré suivant son type et le contenu de la mémoire du programme.

♦

Dans la suite, nous parlons de *traitement immédiat* lorsqu'une occurrence est traitée au moment où elle survient, et de *traitement différé* ou *consommation* lorsqu'elle a été mémorisée avant d'être traitée.

2.1.3 Sémantique et compilation des programmes Electre

Nous avons présenté dans la section précédente quelques aspects syntaxiques du langage ELECTRE. Dans, cette section, nous montrons comment un programme est compilé, ce qui en définit la sémantique.

Les deux sections suivantes donnent les points essentiels de la sémantique du langage ELECTRE. Nous occultons volontairement certains points qui relèvent de la sémantique «intuitive» du langage puisque notre but n'est pas de décrire le langage lui-même. Nous détaillons légèrement plus ces éléments dans l'introduction de l'annexe A (page 193) auxquels [15] consacre par ailleurs une étude complète.

Obtention du graphe de contrôle

La sémantique opérationnelle du programme est obtenue en considérant les événements auquel il peut réagir, et l'effet de ceux-ci. Dans le cadre de la compilation, cet effet consiste en la *réécriture* du programme au moyen des règles présentées dans [23] et en annexe A (section A.2, page 194).

L'ensemble des événements auxquels réagit un programe donné est connu : il est partitionné en trois ensembles : $\mathcal{E}_F$, les événements fugaces, $\mathcal{E}_{M_1}$, les événements à mémorisation unique, et finalement $\mathcal{E}_{M_*}$, les événements à mémorisation multiple. Rappelons que le type d'un événement est global au programme (remarque 2.2, page 37). Considérons le programme des lecteurs/écrivains de la figure 2.1 (page 35). Il possède trois événements (à mémorisation multiple), donc l'exécution de ce programme débute par la prise en compte de l'un de ces trois événements. Les états suivants de ce programme sont alors obtenus par réécriture suivant une occurrence de $\mathtt{r}_1$, $\mathtt{r}_2$ ou $\mathtt{w}$. Les premières réécritures du programme des lecteurs/écrivains sont les suivantes[1] :

$$\begin{tikzcd} & (\mathtt{W})\,;\mathtt{loop}\,[\dots] & \\ & \mathtt{loop}\,[\mathtt{await}\,\{\{\#\mathtt{r}_1:\mathtt{R}_1 \,|||\, \#\mathtt{r}_2:\mathtt{R}_2\}\mid \#\mathtt{w}:\mathtt{W}\}] \arrow[u, "{\mathtt{w},\ A=[\mathtt{W}]}"'] \arrow[dl, "{\mathtt{r}_1,\ A=[\mathtt{R}_1]}"'] \arrow[dr, "{\mathtt{r}_2,\ A=[\mathtt{R}_2]}"] & \\ (\mathtt{R}_1 \,||\, \mathtt{when}\ \#\mathtt{r}_2:\mathtt{R}_2)\,;\mathtt{loop}\,[\dots] & & (\mathtt{when}\ \#\mathtt{r}_1:\mathtt{R}_1\,||\,\mathtt{R}_2)\,;\mathtt{loop}\,[\dots] \end{tikzcd}$$

où «`loop` [. . .]» représente le programme de la figure 2.1 (page 35). L'opérateur « `when` » est le second opérateur de préemption du langage. L'opérateur « `await` » indique une attente forcée d'un événement : l'attente se prolonge jusqu'à ce que l'événement survienne, contrairement à « `when` » (les règles v, page 195, et vi, page 195, caractérisent les différences entre ces deux opérateurs). L'opérateur « || » réalise la composition parallèle des programmes Electre : ceux-ci s'exécutent en concurrence. Il se différencie de l'opérateur « ||| » par les conditions de terminaison des sous-programmes qu'ils définissent (le lecteur est invité à se réferrer aux règles xvii, page 198, et xviii, page 198, pour plus de détails).

En plus d'être étiquetées par les événements qui provoquent la réécriture, les transitions sont décorées par les ensembles des modules *activés* (noté A), *préemptés* (noté P) et/ou *démarrés* (noté D) qui décrivent les *actions* associées à la transition. Ainsi, dans notre exemple, l'occurrence de `w` provoque, en plus de la réécriture, l'activation du module `W`.

Les modules ayant une durée d'exécution finie, il se produit tôt ou tard, dans le système, un événement **fugace** correspondant à la fin d'un module actif. Par exemple, une évolution possible du programme : «$(\mathtt{when}\ \#\mathtt{r}_1:\mathtt{R}_1\,||\,\mathtt{R}_2)\,;\mathtt{loop}\,[\dots]$» consiste en la terminaison de l'exécution de $\mathtt{R}_2$, menant ainsi au programme de départ : «$\mathtt{loop}\,[\mathtt{await}\,\{\{\#\mathtt{r}_1:\mathtt{R}_1 \,|||\, \#\mathtt{r}_2:\mathtt{R}_2\}\mid \#\mathtt{w}:\mathtt{W}\}]$». En

[1] Dans ces programmes les parenthèses ont été placées pour faciliter la lecture, mais elles ne figurent pas dans la syntaxe d'Electre.

considérant ainsi tous les événements qui peuvent être traités (immédiatement) par les programmes successifs (obtenus par réécriture), ainsi que les terminaisons de modules actifs dans ceux-ci, nous obtenons un graphe qui décrit toutes les évolutions du programme initial. Ce graphe est appelé *graphe de contrôle* du programme. Par exemple, le graphe de contrôle du programme de la figure 2.1 (page 35) est représenté en figure 2.3.

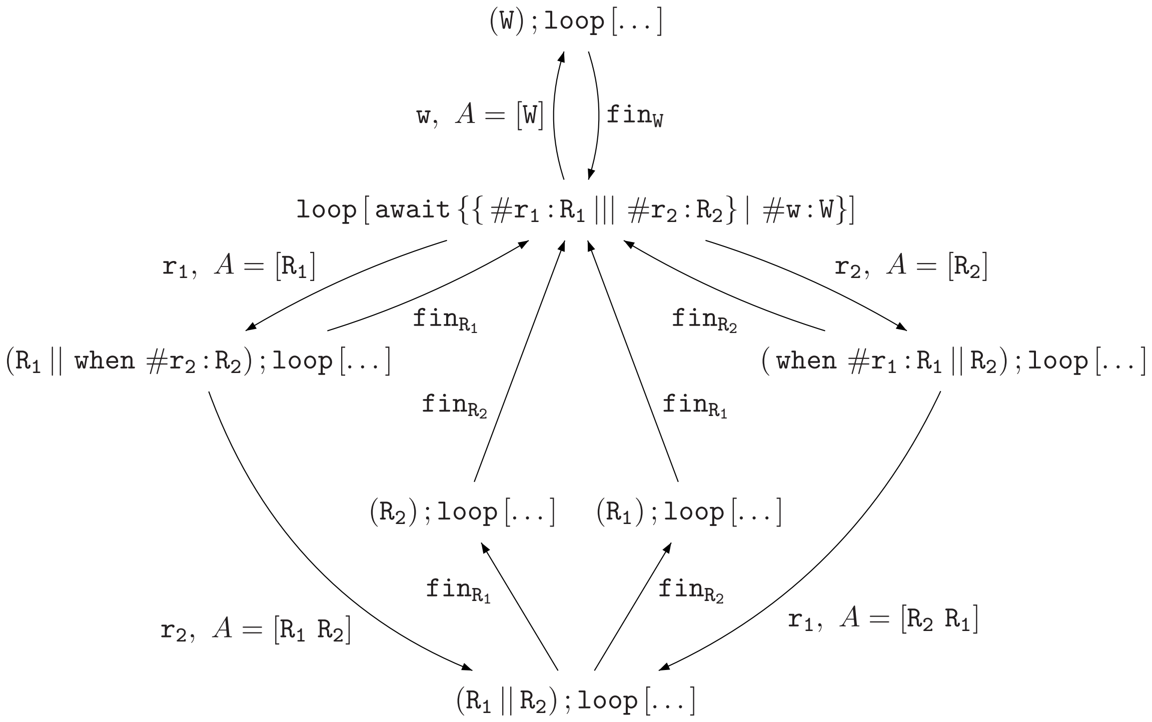


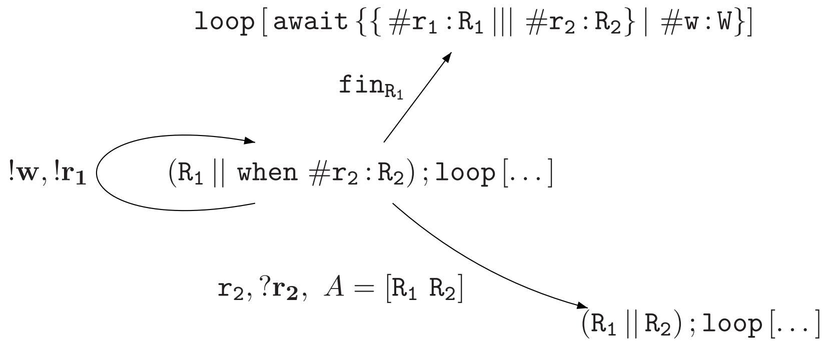
FIG. 2.3 – Graphe de contrôle pour le programme ELECTRE des lecteurs/écrivains.

Lemme 2.4 (Cassez & Roux [23])
Le graphe de contrôle obtenu par compilation d'un programme ELECTRE est fini. ♦

Gestion de la mémorisation

Le graphe de contrôle ne présente que les traitements immédiats des occurrences. Il n'indique que partiellement les prises en compte d'événements : rien n'est spécifié pour les occurrences qui ne peuvent pas être traitées. Cependant, le système modélisé réagit, lui, à toutes les occurrences d'événements, et il faut donc que la modélisation soit fidèle. En particulier, il faut explicitement indiquer le comportement à adopter vis-à-vis des événements mémorisables qui ne peuvent pas être traités.

Lors du traitement d'une occurrence d'événement, le programme ne doit pas distinguer la provenance de cette occurrence. Donc le traitement d'une occurrence d'événement a le même effet, que l'événement vienne d'être émis par l'environnement, ou qu'il provienne de la file de mémorisation. Le graphe de contrôle est ainsi enrichi de façon à ce que chaque transition de traitement d'une occurrence d'événement mémorisable e (à mémorisation unique ou multiple) soit doublée par une transition de traitement d'une occurrence mémorisée de e notée ?e. De plus, en chaque état, chaque événement mémorisable `e` pour lequel aucun traitement n'est explicitement proposé, dans le graphe de contrôle, est mémorisé, ce que nous notons `!e`. Il est clair que le fait de mémoriser une occurrence d'événement ne modifie en rien le programme : l'occurrence est simplement stockée en vue d'un prochain traitement, et aucun module n'est préempté, ni activé, ni démarré. La mémorisation n'induit donc aucune tranformation du programme : toutes les transitions de mémorisation forment des boucles. À titre d'exemple, voici l'état : «$(\mathtt{R}_1 \,||\, \mathtt{when}\ \#\mathtt{r}_2 : \mathtt{R}_2)\,;\mathtt{loop}\,[\ldots]$» du graphe de contrôle précédent enrichi avec la gestion de la mémorisation.



Un tel graphe enrichi est appelé *automate à file réactif* [23, 92, 21]. La figure 2.4 (page 43) représente l'automate à file réactif obtenu depuis le graphe de contrôle de la figure 2.3 (page 40). Les ensembles A, D et P n'y sont pas indiqués, mais ils peuvent être déduits d'après le graphe de contrôle en remarquant que ces ensembles sont les mêmes pour tout couple de transitions associées `e` et ?`e`, et que bien évidemment, aucune action n'est associée à une transition de mémorisation. La suite de ce chapitre est dédiée aux automates à file réactifs qui décrivent la sémantique des programmes Electre.

2.2 Les automates à file réactifs

Comme nous venons de le montrer, les automates à file réactifs [23, 92, 21] ont été introduits pour modéliser le comportement des programmes Electre. D'une façon plus générale, les automates à file réactifs permettent de modéliser n'importe quel système réactif (sans qu'il soit nécessaire de le spécifier en Electre), avec la particularité intéressante de permettre la mémorisation des occurrences d'événements qui ne peuvent pas être traitées lorsqu'elles surviennent.

Le modèle est donc constitué d'un système de contrôle fini, et d'une file de mémorisation pour les événements. En ce sens, il se rapproche des automates à file [94] et des machines communicantes [18, 1, 2, 24, 40].

Dans cette section, nous introduisons tout d'abord les automates à file réactifs et leurs principales propriétés mises à jour dans [91, 92]. Lors de la construction de l'automate à file réactif associé à un programme Electre, les propriétés des événements sont préservées. Ainsi, l'automate possède des événements fugaces, d'autres qui sont à mémorisation unique et enfin, une dernière catégorie d'événements à mémorisation multiple. Nous montrons comment les événements à mémorisation unique peuvent être simulés par des événements à mémorisation multiple, ce qui nous permet de ne plus considérer que ces derniers par la suite.

2.2.1 Définition et propriétés

Définition 2.5 (Automate à file réactif (AFR))
Un *automate à file réactif (AFR)* [21] est un quadruplet $R = (Q_R, q_R^0, A_R, \rightarrow_R)$ où :

1. Q_R est un ensemble fini d'*états*,
2. q_R^0 est l'état initial de R,
3. $A_R = \Sigma \cup (\{!, ?\} \times \Sigma_M)$ est l'ensemble des *actions* de R, et Σ est l'ensemble des *événements* agissant sur R partitionné en :
   (a) Σ_M : l'ensemble des *événements mémorisables*, lui-même scindé en deux parties distinctes, Σ_{M_1} et Σ_{M_*}, respectivement, l'ensemble des événements à *mémorisation unique*, et l'alphabet des événements à *mémorisation multiple*,
   (b) Σ_F : l'ensemble des événements fugaces,
   et : $\Sigma_{M_1} \cap \Sigma_{M_*} = \emptyset$, $\Sigma_M = \Sigma_{M_1} \cup \Sigma_{M_*}$ et $\Sigma_M \cap \Sigma_F = \emptyset$, $\Sigma = \Sigma_M \cup \Sigma_F$.
4. $\rightarrow_R \subseteq Q_R \times A_R \times Q_R$ est la *relation de transition* telle que pour tout état $q \in Q_R$ et tout événement mémorisable $\mathsf{e} \in \Sigma_M$:
   (a) soit $q \xrightarrow{!\mathsf{e}}_R q$, l'occurrence de e est mémorisée,
   (b) sinon il existe un état $q' \in Q_R$ tel que $q \xrightarrow{\mathsf{e}}_R q'$ et $q \xrightarrow{?\mathsf{e}}_R q'$, l'événement est traité, soit immédiatement, soit de façon différée.

♦

Dans la suite, lorsque cela est nécessaire pour éviter toute ambiguïté, nous notons $(\Sigma_F)_R$ l'alphabet des événements fugaces de l'AFR R, et de même : $(\Sigma_M)_R$, $(\Sigma_{M_1})_R$ et $(\Sigma_{M_*})_R$ pour les autres ensembles d'événements.

Remarque 2.6
Un automate à file réactif définit un système réactif ouvert. C'est à dire que seule la façon dont le système réagit à une occurrence d'un événement donné, alors qu'il se trouve dans un état donné, est spécifiée. Un AFR ne contient aucune information sur la manière (la fréquence, l'origine, etc.) qu'ont les événements de survenir. ♦

La figure 2.4 (page 43) montre l'automate à file réactif obtenu par la compilation du programme Electre de la figure 2.1, page 35. Les états de cet automate à file réactif sont étiquetés par les noms des modules qui y sont actifs.

Comme nous l'avons décrit en section 2.1.3 (page 39), lors de la compilation d'un programme Electre en un AFR R, les ensembles d'actions associés à chaque transition sont calculés. Ceux-ci sont alors transcrits dans R au travers des fonctions A, D et P qui associent respectivement à chacune de ses transitions, les ensembles des modules activés, démarrés et préemptés.

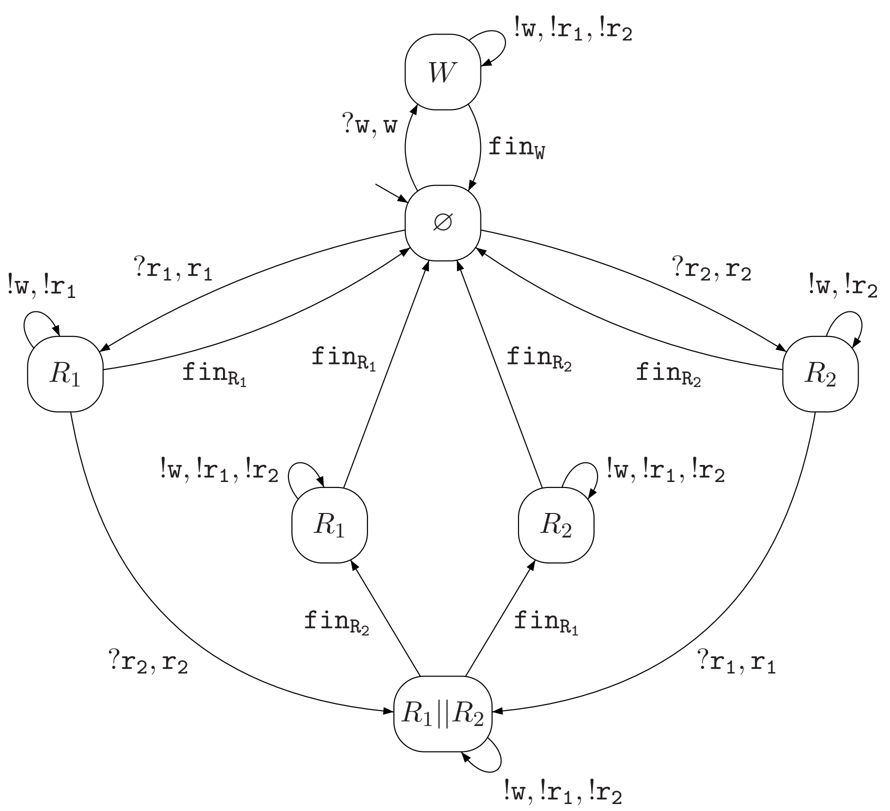


FIG. 2.4 – Automate à file réactif pour le programme des lecteurs/écrivains.

Remarque 2.7
Les fonctions A, D et P n'apparaîssent pas dans la définition des AFR (définition 2.5) afin d'en préserver la généralité : elles ne concernent que les AFR obtenus par compilation d'un programme ELECTRE. Pour la même raison, ces trois fonctions apparaîssent à nouveau au chapitre 6 (page 127) qui concerne les AFR obtenus depuis des programmes ELECTRE, mais elles sont ignorées ailleurs, où les résultats présentés concernent donc les AFR de façon générale. ♦

Dans la suite, $Pred_R(q) = \{q' \in Q_R \,|\, \exists a \in A_R,\ q' \xrightarrow{a}_R q\}$ désigne l'ensemble des prédécesseurs de l'état q, et de même $Succ_R(q) = \{q' \in Q_R \,|\, \exists a \in A_R,\ q \xrightarrow{a}_R q'\}$ représente l'ensemble des successeurs de q.

Structure d'automate à file réactif

La nécessité pour un AFR d'être complet vis-à-vis des événements mémorisables est naturelle, mais aussi assez contraignante. C'est pourquoi nous introduisons les *structures d'AFR* qui lèvent cette limitation. Bien entendu, elles ne sont pas utilisées pour la modélisation de systèmes réactifs, mais elles permettent de considérer une partie d'un AFR sans se soucier de complétude. Cette notion nous est particulièrement utile pour les preuves à venir.

Définition 2.8 (Structure d'AFR)
Une *structure d'AFR* est un quadruplet $R = (Q_R, q_R^0, A_R, \to_R)$ où Q_R, q_R^0 et A_R sont définis de la même façon que pour la définition 2.5 (page 42), mais $\to_R$ n'est pas nécessairement complète. Cependant, elle respecte la similarité entre les transitions de traitements immédiats et différés. Elle vérifie donc, $\forall q \in Q_R,\ \forall \mathsf{e} \in \Sigma_M$:

1. soit $q \xrightarrow{!\mathsf{e}}_R q$, l'occurrence de e est mémorisée,
2. soit il existe un état $q' \in Q_R$ tel que $q \xrightarrow{\mathsf{e}}_R q'$ et $q \xrightarrow{?\mathsf{e}}_R q'$, l'événement est traité, soit immédiatement, soit de façon différée,
3. sinon $\nexists q' \in Q_R$ tel que $q \xrightarrow{!\mathsf{e}}_R q'$ ou $q \xrightarrow{?\mathsf{e}}_R q'$ ou $q \xrightarrow{\mathsf{e}}_R q'$, toute occurrence de e qui survient en q est perdue.

♦

Remarque 2.9
À la différence d'un AFR, une structure d'AFR n'est donc pas nécessairement complète vis-à-vis des événements mémorisables. Par contre, il n'est pas possible que la prise en compte d'un événement soit partiellement spécifiée : il n'y a jamais de transitions $q \xrightarrow{\mathsf{e}}_R q'$ sans la transition associée $q \xrightarrow{?\mathsf{e}}_R q'$, et vice-versa, dans une structure d'AFR. ♦

Système à file réactif

La sémantique d'un AFR R est donnée par le système de transitions (infini) nommé *système à file réactif* [21] obtenu depuis R en considérant le contenu de la file de mémorisation et son évolution lors du franchissement des transitions. Les états d'un *système à file réactif* correspondent donc aux configurations (q, w), où $q \in Q_R$ est un état de R, et $w \in \Sigma_M^*$ représente le contenu de la mémoire. Dans la suite :

$$\Sigma_q^! = \{\mathsf{e} \in \Sigma_M \mid q \xrightarrow{!\mathsf{e}}_R q\} \tag{2.1}$$

désigne l'alphabet des événements mémorisés en $q \in Q_R$. Une configuration (q, w) est *stable* si $w \in (\Sigma_q^!)^*$, et *instable* dans le cas contraire.

Remarque 2.10
Par complémentarité, si e $\notin \Sigma_q^!$ et e est mémorisable, alors ses occurrences sont traitées (immédiatement ou de façon différée) en q. ♦

Définition 2.11 (Système à file réactif (SFR))
Un *système à file réactif (SFR)* [21] est le système de transitions $S = (Q_S, q_S^0, A_S, \rightarrow_S)$ qui donne la sémantique d'un AFR $R = (Q_R, q_R^0, A_R, \rightarrow_R)$ par :

1. Q_S est l'ensemble des *configurations* de S. Il est défini comme le plus petit sous-ensemble de $Q_R \times \Sigma_M^*$ obtenu depuis (q_R^0, ε) par l'application de $\rightarrow^*{}_S$,
2. $q_S^0 = (q_R^0, \varepsilon)$ est la configuration initiale,
3. $A_S = A_R$ est l'ensemble des actions de S,
4. $\rightarrow_S \subseteq Q_S \times A_S \times Q_S$ est la plus petite relation de transition définie pour toute configuration $(q, w) \in Q_S$ par :
   (a) $(q, w) \xrightarrow{!\mathsf{e}}_S (q, w')$ si $q \xrightarrow{!\mathsf{e}}_R q$ et $w \in \left(\Sigma_q^!\right)^*$, et $w' = w$ si $|w|_{\mathsf{e}} \geq 1$ et e $\in \Sigma_{M_1}$, et $w' = w.\mathsf{e}$ sinon,

(b) $(q, w) \xrightarrow{\mathsf{e}}_S (q', w)$ si $q \xrightarrow{\mathsf{e}}_R q'$ et $w \in \left(\Sigma^!_q\right)^*$,

(c) $(q, w) \xrightarrow{?\mathsf{e}}_S (q', w')$ si $q \xrightarrow{?\mathsf{e}}_R q'$ et $\exists w_1 \in \left(\Sigma^!_q\right)^*$, $w_2 \in (\Sigma_M)^*$ t.q. $w = w_1 \mathsf{e} w_2$ et $w' = w_1 w_2$,

(d) $(q, w) \xrightarrow{\mathsf{e}}_S (q, w)$ si $\mathsf{e} \notin (out(q) \cup \Sigma_M)$ et $w \in \left(\Sigma^!_q\right)^*$.

♦

Remarque 2.12

Le mot w qui apparaît dans une configuration (q, w) représente le *contenu de la file*. Notons que w **est toujours un mot fini**. ♦

La politique Fiffo (First In First Fireable Out), de gestion de la mémorisation, est exprimée, dans les points (4a) à (4d), par les conditions de la forme $w \in \left(\Sigma^!_q\right)^*$. En effet, elles entraînent :

- **La priorité au traitement des occurrences mémorisées :** puisque dans les points (4a), (4b) et (4d), cette condition ne permet ni de mémoriser, ni de traiter une nouvelle occurrence d'événement dans une configuration instable.
- **Le traitement des occurrences mémorisées «dès que possible» :** la condition $w_1 \in \left(\Sigma^!_q\right)^*$, dans le point (4c), permet que l'occurrence qui est prise en compte ne soit pas nécessairement celle qui est en tête de la file (contrairement à l'ordre Fifo classique).
- **La priorité à l'occurrence mémorisée la plus ancienne :** la condition $w_1 \in \left(\Sigma^!_q\right)^*$, dans le point (4c), impose que l'occurrence consommée de e est la plus ancienne qu'il est possible de traiter.

L'ordre Fiffo peut donc être vu comme l'ordre fifo classique, s'appliquant uniquement aux événement qui peuvent être consommés dans l'état q.

Dans le point (4a), un événement e, à mémorisation unique ($\mathsf{e} \in \Sigma_{M_1}$), n'est pas mémorisé si la file en contient déjà une occurrence ($|w|_{\mathsf{e}} \geq 1$). Finalement, le point (4d) spécifie le comportement du système à file réactif vis-à-vis des événements fugaces pour lesquels aucun comportement n'est explicitement spécifié. Ceci est nécessaire puisque les systèmes à file réactifs décrivent la sémantique opérationnelle des automates à file réactifs. Nous spécifions simplement que ces occurrences laissent le système dans la même configuration.

Une *séquence de stabilisation* est une séquence de transitions dans S menant d'une configuration instable à une configuration stable :

$$(q_0, w_0) \xrightarrow{?\mathsf{e}_1}_S (q_1, w_1) \xrightarrow{?\mathsf{e}_2}_S \ldots \xrightarrow{?\mathsf{e}_n}_S (q_n, w_n)$$

avec : $\forall i \in \{0, \ldots, n-1\},\ w_i \notin (\Sigma^!_{q_i})^*$ et $w_n \in (\Sigma^!_{q_n})^*$.

Remarque 2.13

Toute séquence de stabilisation a une longueur finie puisque la file contient toujours un nombre fini d'occurrences d'événements (remarque 2.12). ♦

Une *exécution* d'un système à file réactif S est une séquence de transitions maximale dans S débutant en q_S^0 :

$$q_S^0 = (q_R^0, \varepsilon) \xrightarrow{a_1}_S (q_1, w_1) \xrightarrow{a_2}_S \ldots (q_n, w_n) \xrightarrow{a_{n+1}}_S \ldots$$

L'ensemble des exécutions de S est noté $[\![\ S\ ]\!]$.

Lemme 2.14

Soient S un SFR et $\sigma \in [\![\ S\ ]\!]$ une exécution de S. Toute séquence de stabilisation maximale $\sigma_?$ dans σ est précédée d'une transition de traitement immédiat d'une occurrence d'événement. ♦

Preuve. Notons tout d'abord qu'aucune séquence de stabilisation ne peut débuter dans la configuration initiale (q_R^0, ε) où la file est vide.

Ensuite, puisque $\sigma_?$ est finie (remarque 2.13), elle est forcément précédée soit d'une transition de mémorisation, ou sinon, d'une transition de traitement immédiat. Supposons qu'il s'agisse d'une transition de mémorisation :

$$(q, w) \xrightarrow{!\mathtt{e}}_S (q, w\mathtt{e})$$

Alors, puisque la séquence de stabilisation $\sigma_?$ débute en $(q, w\mathtt{e})$, cette configuration est instable. Nous en déduisons que $w \notin \left(\Sigma_q^!\right)^*$ puisque par définition $\mathtt{e} \in \Sigma_q^!$. Alors, la configuration (q, w) est elle aussi instable et la transition de mémorisation ne peut être franchie (point (4a) de la définition 2.11, page 44).

La transition qui précède une séquence de stabilisation est donc nécessairement un traitement immédiat d'une occurrence d'événement. ♦

La figure 2.5 (page 47) montre le haut de l'arbre des exécutions de l'AFR de la figure 2.4 (page 43). Les nœuds qui y sont encadrés apparaîssent précédemment sur la même branche, c'est pourquoi nous ne les avons pas développés. Pour plus de clarté, nous n'avons pas dessiné les transitions de prise en compte des événements fugaces qui ne sont pas attendus. Ainsi, il faudrait par exemple ajouter en (R_1, ε) les transitions correspondant au traitement des occurrences de $\mathtt{fin}_{\mathtt{R_2}}$ et $\mathtt{fin}_{\mathtt{W}}$.

Nous voyons clairement apparaître la priorité donnée au traitement des occurrences mémorisées dans les configurations $(\varnothing, \mathtt{r_1})$, $(W, \mathtt{r_1wr_2})$ et $(R_1, \mathtt{wr_2})$ où c'est la seule action possible, conformément à la condition $w \in (\Sigma_q^!)^*$ des points (4a), (4b) et (4d) de la définition 2.11, page 44. Par ailleurs, la transition :

$$(R_1, \mathtt{wr_2}) \xrightarrow{?\mathtt{r_2}} (R_1 \,||\, R_2, \mathtt{w})$$

montre le traitement des occurrences mémorisées «dès que possible», puisque $\mathtt{r_2}$ n'est pas l'occurrence qui figure en tête de la file de mémorisation. C'est cependant bien la plus ancienne occurrence qui peut être consommée.

La définition 2.11 donne l'ensemble des successeurs d'une configuration d'un système à file réactif. Nous définissons son inverse, l'ensemble des prédécesseurs par :

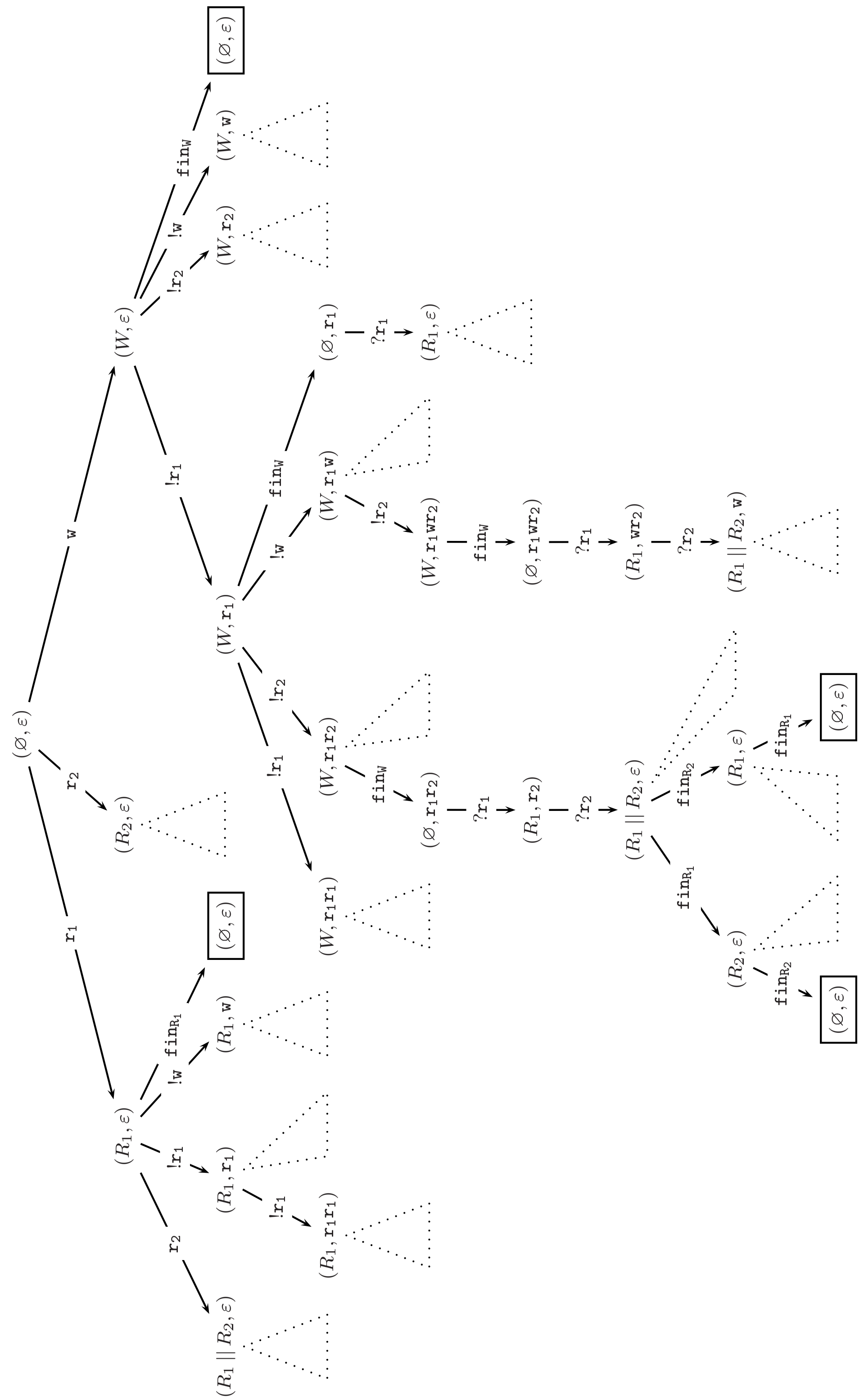


FIG. 2.5 – Arbre des exécutions de l'AFR des lecteurs/écrivains.

Définition 2.15
L'ensemble des prédecesseurs d'une configuration (q, w) est défini par :

$$Pred_S((q,w)) = \bigcup_{\mathsf{e}\in\Sigma} \left(Pred_S^{\mathsf{e}}((q,w)) \cup Pred_S^{!\mathsf{e}}((q,w)) \cup Pred_S^{?\mathsf{e}}((q,w))\right)$$

où :

$$Pred_S^{\mathsf{e}}((q,w)) = \{(q',w) \mid q' \xrightarrow{\mathsf{e}}_R q\}$$
$$Pred_S^{!\mathsf{e}}((q,w)) = \{(q,w') \mid q \xrightarrow{!\mathsf{e}}_R \text{ et } w = w'\mathsf{e}\}$$
$$Pred_S^{?\mathsf{e}}((q,w)) = \{(q',w') \mid q' \xrightarrow{?\mathsf{e}}_R q \text{ et } \exists w_1' \in (\Sigma_{q'}^!)^*,\ \exists w_2' \in \Sigma_M^* \text{ t.q. } w = w_1'w_2',\ w' = w_1'\mathsf{e}w_2'\}$$

♦

Notons qu'au moins $Pred_S^{?\mathsf{e}}((q_R,w))$ ou $Pred_S^{\mathsf{e}}((q_R,w))$ et $Pred_S^{!\mathsf{e}}((q_R,w))$ sont vides compte tenu de la priorité donnée au traitement des occurrences mémorisées.

Propriétés des automates et systèmes à file réactifs

Dans cette section, nous donnons les idées et les résultats les plus importants de [91, 92] pour situer le cadre de notre étude. Nous considérons un automate à file réactif R dont la sémantique est donnée par le système à file réactif S. Pour chaque configuration accessible de la forme : $(q', w_1'\mathsf{e}w_2')$, il existe une exécution où figure la transition de mémorisation de cette occurrence particulière de e :

$$(q_R^0, \varepsilon) \xrightarrow{\sigma_1} (q,w) \xrightarrow{!\mathsf{e}} (q, w\mathsf{e}) \xrightarrow{\sigma_2} (q', w_1'\mathsf{e}w_2')$$

Notons que les transitions de mémorisation des automates à file réactifs forment des boucles. Il s'ensuit donc qu'à chaque exécution avec mémorisation :

$$(q_R^0, \varepsilon) \xrightarrow{\sigma_1} (q,w) \xrightarrow{!\mathsf{e}} (q, w\mathsf{e}) \xrightarrow{\sigma_2} (q', w_1'\mathsf{e}w_2')$$

avec $|\sigma_2|_{?\mathsf{e}} \leq |w|_{\mathsf{e}}$, correspond une exécution où e n'est pas mémorisé :

$$(q_R^0, \varepsilon) \xrightarrow{\sigma_1} (q,w) \xrightarrow{\sigma_2} (q', w_1'w_2')$$

puisque le franchissement d'une transition de mémorisation n'est pas imposé. Nous en déduisons que lorsqu'une configuration (q,w) est accessible, toutes les configurations (q,w') avec $w' \preccurlyeq w$ sont aussi accessibles. L'ensemble des configurations accessibles est donc clos par le bas, et par conséquent reconnaissable.

Soit $X \subseteq \Sigma_M$, notons $\mathcal{S}(X^*) = \{w \in X^* \mid \forall \mathsf{e} \in \Sigma_{M_1},\ |w|_{\mathsf{e}} \leq 1\}$ l'ensemble des contenus de file valides[2] sur X. Notons que $\mathcal{S}(X^*)$ est rationnel. Ensuite, nous avons :

1. Si $q \in Q_R$ est accessible [3] dans R alors, quel que soit $w \in \mathcal{S}((\Sigma_q^!)^*)$, (q,w) est accessible dans S.

[2]Au sens où ils contiennent au plus une occurrence de chaque événement à mémorisation unique.
[3]Au sens où il existe un chemin dans R de q_R^0 à q.

Il suffit de constater que puisqu'il existe un chemin de q_R^0 à q dans R, alors la configuration (q, ε) est accessible, et en notant $w = \mathtt{e_1} \ldots \mathtt{e_n}$, la séquence $(q, \varepsilon) \xrightarrow{!\mathtt{e_1}}_S \ldots \xrightarrow{!\mathtt{e_n}}_S (q, w)$ prolonge l'exécution de R développée jusqu'à (q, ε).

2. Si (q, w) est accessible et stable, et $(q, w) \xrightarrow{\tau}_S (q', w) \xrightarrow{?\mathtt{e_1}}_S \ldots \xrightarrow{?\mathtt{e_n}}_S (q'', w'')$, avec $\tau \in \Sigma_F$, est une exécution de S depuis cette configuration, alors : $(\mathtt{e_1} \ldots \mathtt{e_n}) \in \mathcal{S}((\Sigma_q^!)^*)$ et $w'' \in \mathcal{S}(X^*)$ où $X = \Sigma_q^! \setminus (\Sigma_{M_1} \cap \{\mathtt{e_1}, \ldots, \mathtt{e_n}\})$.

   En effet, puisque (q, w) est accessible, $w \in \mathcal{S}(\Sigma_q^{!\,*})$, donc $\mathtt{e_1} \ldots \mathtt{e_n} \in \mathcal{S}(\Sigma_q^{!\,*})$. De plus, pour tout $\mathtt{e} \in \Sigma_{M_1}$, $|w''|_\mathtt{e} + |\mathtt{e_1} \ldots \mathtt{e_n}|_\mathtt{e} = |w|_\mathtt{e} \leq 1$. Nous en déduisons que $w'' \in \mathcal{S}((\Sigma_q^!)^*)$.

   Notons $CoReach(q'', X)$ l'ensemble des états $q \in Q_R$ prédécesseurs stables des configurations (q'', w), avec $w \in X$:

$$CoReach(q'', X) = \Big\{ q \in Q_R \,|\, q \xrightarrow{\tau}_R q' \xrightarrow{?\mathtt{e_1} \ldots ?\mathtt{e_n}}_R q'' \text{ et } (\mathtt{e_1} \ldots \mathtt{e_n}) \in \mathcal{S}(\Sigma_q^{!\,*}) \text{ et } X = \Sigma_q^! \setminus (\Sigma_{M_1} \cap \{\mathtt{e_1}, \ldots, \mathtt{e_n}\}) \Big\}$$

Nous en déduisons que pour un état $q \in Q_R$ donné, les mots w tels que (q, w) est accessible correspondent à l'un des deux cas précédents. Ces mots w, définissent l'ensemble :

$$\mathcal{S}(\Sigma_{q_R}^!) \cup \left[\bigcup_{X \subseteq \Sigma_M \,|\, CoReach(q_R, X) \neq \emptyset} \mathcal{S}(X^*) \right]$$

Finalement, il est possible de calculer une expression régulière pour ce langage, moyennant un modification de *CoReach* de façon à ne prendre en compte que les séquences de stabilisation sans cycle, ce qui laisse ce langage invariant. En effet, lorsqu'une séquence de stabilisation $q'_R \xrightarrow{?\mathtt{e_1}}_R \ldots \xrightarrow{?\mathtt{e_n}}_R q''_R$ (avec $X = \Sigma_{q_R}^! \setminus (\Sigma_{M_1} \cap \{\mathtt{e_1}, \ldots, \mathtt{e_n}\})$) est libérée de ses cycles, nous obtenons une séquence de stabilisation (éventuellement vide) $q'_R \xrightarrow{?\mathtt{e'_1}}_R \ldots \xrightarrow{?\mathtt{e'_k}}_R$, où $\mathtt{e'_1} \ldots \mathtt{e'_k}$ est un sous-mot de $\mathtt{e_1} \ldots \mathtt{e_n}$, et $X' = \Sigma_{q_R}^! \setminus (\Sigma_{M_1} \cap \{\mathtt{e'_1}, \ldots, \mathtt{e'_k}\}) \supseteq X$. Nous avons donc le résultat suivant :

Lemme 2.16 (Sutre *et al.* [92])
L'ensemble des états accessibles d'un système à file réactif est effectivement reconnaissable (problème (ERP)). ♦

Corollaire 2.17 (Sutre *et al.* [92])
Les problèmes d'accessibilité (R), de couverture (C), et de bornitude (B) sont décidables pour les systèmes à file réactifs. ♦

Preuve. La décidabilité de (R), (B) et (C) découle de la reconnaissabilité effective de l'ensemble des états accessibles des AFR (lemme 2.16). ♦

Remarque 2.18
Cependant, la résolution du problème (B) est encore plus simple : il n'est pas nécessaire de calculer explicitement l'ensemble des états accessibles. En effet, si un AFR R ne possède pas d'événement à

mémorisation multiple, alors, le SFR S qui donne sa sémantique, est borné.

Par contre, supposons que R possède au moins un événement à mémorisation multiple. Pour que S soit non borné, il est nécessaire qu'il existe une configuration stable et accessible (q, w), où un événement à mémorisation multiple $\mathsf{e} \in \Sigma_M$ peut être mémorisé : $q \xrightarrow{!\mathsf{e}}_R q$. Cette condition est aussi suffisante, puisque par le point (4a) de la définition 2.11 (page 44), toutes les configurations $(q, w(\mathsf{e})^n)$, avec $n \geq 0$, sont alors accessibles.

Une configuration stable (q, w), avec $w \in \left(\Sigma_q^!\right)^*$, est accessible si et seulement si (q, ε) est accessible : l'ensemble des états accessibles est clos par le bas. Rappelons que les transitions de mémorisation sont des boucles (il n'est pas nécessaire de les franchir), et à toute transition de consommation $q' \xrightarrow{?\mathsf{e}}_R q''$ correspond une transition de traitement immédiat $q' \xrightarrow{\mathsf{e}}_R q''$, avec $q', q'' \in Q_R$ et $\mathsf{e} \in \Sigma_M$. Par conséquent, la configuration stable (q, w) est accessible dans S si et seulement si q est accessible dans R, ce qui est décidable, R étant fini. ♦

Deux séquences de transitions, dans un système de transitions étiqueté, sont *équivalentes modulo bégaiement* si l'une peut être obtenue depuis l'autre en retirant les répetitions successives d'un même état :

$$\ldots q \to q' \to q' \to \ \ldots \to q' \to q'' \ldots \qquad \text{devient} \qquad \ldots q \to q' \to q'' \ldots$$

Une séquence de transitions définit implicitement une séquence d'états. Deux séquences infinies d'états équivalentes modulo bégaiement satisfont les même formules de $\text{LTL}\backslash\mathsf{X}$ [73]. Un automate à file réactif R et le système à file réactif S qui lui correspond satisfont les mêmes formules de $\text{LTL}\backslash\mathsf{X}$. En effet, dans le cas où $\Sigma_M = \emptyset$, tous les comportements de S sont dans R (la file est toujours vide), et puisque R est fini, nous en déduisons que le model-checking de $\text{LTL}\backslash\mathsf{X}$ est PSPACE-complet dans ce cas [90]. Lorsque $\Sigma_M \neq \emptyset$, nous construisons un AFR $\tilde{R}$ depuis R tel que :

- toutes les séquences d'états infinies de $\tilde{R}$ sont des séquences d'états infinies de S (relativement à la partie gauche du produit cartésien) qui sont elles-même par définition des séquences infinies d'états de R,
- toute séquence infinie d'états de R admet une séquence infinie d'états de $\tilde{R}$ qui lui est équivalente modulo bégaiement.

Nous en déduisons que $\tilde{R}$, R et S satisfont les mêmes formules de $\text{LTL}\backslash\mathsf{X}$, et par conséquent :

Lemme 2.19 (Sutre *et al.* [92])
Le model-checking de $\text{LTL}\backslash\mathsf{X}$ est décidable et PSPACE-complet pour les systèmes à file réactifs. ♦

La décidabilité du model-checking de LTL, (MC-LTL), et de CTL, (MC-CTL), sont cependant des problèmes ouverts.

2.3 Systèmes à file réactifs embarqués

Comme nous l'avons indiqué par la remarque 2.18 (page 49), le problème de la bornitude est trivialement décidable pour les automates à file réactifs. Cependant, ceci repose sur le fait que toute transition de mémorisation peut être franchie un nombre non borné et non déterminé de fois.

Les systèmes à file réactifs sont en effet *ouverts* : il n'est rien précisé quant à la façon dont surviennent les événements. Ce problème est du à une **lacune de spécification inhérente au modèle même des AFR** qui propose uniquement de spécifier les comportements des systèmes[4]. C'est pourquoi nous introduisons maintenant les *systèmes à file réactifs embarqués (SFR Embarqués)* qui sont obtenus par cloture des systèmes à file réactifs.

2.3.1 Environnement d'un système à file réactif

Nous décrivons en section suivante la synchronisation d'un système à file réactif avec son environnement.

Définition 2.20 (Environnement d'un SFR)
L'environnement d'un SFR S est un système de transitions $E = (Q_E, q_E^0, A_E, \to_E)$. ♦

Lorsque E est fini, l'environnement est *rationnel*. Un environnement est *périodique* s'il est déterministe, fini et s'il existe un mot fini $u \in (A_E)^*$ tel que tout chemin de E est étiqueté par un préfixe de u^*. Un environnement est *ultimement périodique* s'il existe un langage rationnel $L \subseteq (A_E)^*$ et un mot fini $u \in (A_E)^*$ tels que tout chemin de E est étiqueté par un préfixe de $L.(u)^\omega$.

Nous disons d'un environnement rationnel E qu'il est *non-contraignant* si et seulement si $\mathcal{L}(E) = \Sigma^*$, où Σ est l'ensemble des événements de S. Intuitivement, lorsqu'un système à file réactif est plongé dans un environnement non-contraignant, cela ne modifie en rien ses exécutions.

2.3.2 Système à file réactif embarqué

L'environnement d'un système à file réactif définit les instants (logiques) d'occurrences des événements. Ainsi, les transitions de traitement immédiat (étiquetées e) et les transitions de mémorisation (étiquetées !e) sont maintenant franchissables uniquement lorsque l'environnement propose les événements adéquats. Bien entendu, le franchissement d'une transition de consommation (étiquetée ?e) ne dépend pas de l'environnement : elle est interne au système.

Définition 2.21 (Système à file réactif embarqué (SFR Embarqué))
Soit $S = (Q_S, q_S^0, A_S, \to_S)$ un système à file réactif et $E = (Q_E, q_E^0, A_E, \to_E)$ un environnement. Le *système à file réactif embarqué (SFR Embarqué)*, $S||E$, est le système de transitions $S||E = (Q_{S||E}, q_{S||E}^0, A_{S||E}, \to_{S||E})$ défini par :

1. $Q_{S||E}$ est le plus petit sous-ensemble de $Q_S \times Q_E = Q_R \times \Sigma_M^* \times Q_E$ obtenu depuis $q_{S||E}^0$ par application de la fermeture reflexive et transitive de $\to_{S||E}$, définissant ainsi l'ensemble des *configurations* de $S||E$,
2. $q_{S||E}^0 = \langle q_R^0, \varepsilon, q_E^0 \rangle$ est la configuration initiale ($q_S^0 = (q_R^0, \varepsilon)$),
3. $A_{S||E} = A_S$ est l'ensemble des *actions* de $S||E$,

[4]Il s'agit bien sûr d'un choix volontaire lors de la définition du modèle.

4. $\rightarrow_{S||E} \subseteq Q_{S||E} \times A_{S||E} \times Q_{S||E}$ est la plus petite relation de transition vérifiant :
   (a) $\langle q_R, w, q_E\rangle \xrightarrow{!\mathsf{e}}_{S||E} \langle q_R, w\mathsf{e}, q'_E\rangle$ si $(q_R, w) \xrightarrow{!\mathsf{e}}_S (q_R, w\mathsf{e})$ et $q_E \xrightarrow{\mathsf{e}}_E q'_E$,
   (b) $\langle q_R, w, q_E\rangle \xrightarrow{\mathsf{e}}_{S||E} \langle q'_R, w, q'_E\rangle$ si $(q_R, w) \xrightarrow{\mathsf{e}}_S (q'_R, w)$ et $q_E \xrightarrow{e}_E q'_E$,
   (c) $\langle q_R, w_1\mathsf{e}w_2, q_E\rangle \xrightarrow{?\mathsf{e}}_{S||E} \langle q'_R, w_1w_2, q_E\rangle$ si $(q_R, w_1\mathsf{e}w_2) \xrightarrow{?\mathsf{e}}_S (q'_R, w_1w_2)$,

♦

Les notions de configurations stables, instables et de séquence de stabilisation se transmettent aux systèmes à file réactifs embarqués.

Remarque 2.22
Dans la définition 2.21, il est important de différencier les points (4a) et (4b) d'une part, du point (4c) d'autre part. En effet, dans les deux premiers cas, l'environnement conditionne le franchissement de la transition. Si E n'émet pas e, S ne peut pas franchir la transition correspondante. Dans le dernier point, en revanche, le franchissement de la transition de consommation ?e est du seul ressort du SFR. ♦

L'ensemble des prédécesseurs d'une configuration d'un SFR Embarqué est alors simplement défini en complétant la définition 2.15 (page 46) par la contrainte imposée par l'environnement :

Définition 2.23
L'ensemble des prédécesseurs d'une configuration $\langle q_R, w, q_E\rangle$ d'un SFR Embarqué est défini par :

$$\begin{aligned} Pred_{S||E}(\langle q_R, w, q_E\rangle) = & \bigcup_{\mathsf{e}\in in(q_E)} \Big(\big(Pred_S^{\mathsf{e}}((q_R, w)) \cup Pred_S^{!\mathsf{e}}((q_R, w))\big) \times Pred_E^{\mathsf{e}}(q_E) \Big) \\ & \cup \bigcup_{?\mathsf{e}\in in(q_R)} \Big(Pred_S^{?\mathsf{e}}((q_R, w)) \times \{q_E\} \Big) \end{aligned}$$

♦

Dans la suite la projection π_E est le morphisme de $(A_{S||E}, .)$ sur $(A_E, .)$ défini par : $\pi_E(\mathsf{e}) = \mathsf{e}$, $\pi_E(!\mathsf{e}) = \mathsf{e}$ et $\pi_E(?\mathsf{e}) = \varepsilon$. Cette projection capture les événements émis par E qui contraignent les exécutions de $S||E$.

Prenons à nouveau pour exemple l'automate à file réactif de la figure 2.4 (page 43). Nous spécifions ses comportements valides en le synchronisant au système de transitions représenté en figure 2.6 qui reconnaît le langage rationnel $(\mathtt{wr_1r_2fin_W}(\mathtt{fin_{R_1}fin_{R_2}} + \mathtt{fin_{R_2}fin_{R_1}}))^*$. Nous considérons donc que l'environnement de l'AFR propose toujours trois requêtes $\mathtt{w}$, $\mathtt{r_1}$ et $\mathtt{r_2}$, dans cet ordre, puis la terminaison du module d'écriture $\mathtt{W}$ survient, et enfin, ce sont les terminaisons des modules de lecture $\mathtt{R_1}$ et $\mathtt{R_2}$ qui surviennent, dans un ordre non déterminé.

En reprenant l'arbre des exécutions représenté en figure 2.5 (page 47), nous voyons qu'il n'est plus possible de traiter une occurrence de $\mathtt{r_1}$ dans la configuration initiale du système à file réactif embarqué puisque l'environnement (lui aussi dans son état initial) ne propose qu'une occurrence de $\mathtt{w}$. Les exécutions du système à file réactif embarqué sont représentées par l'arbre

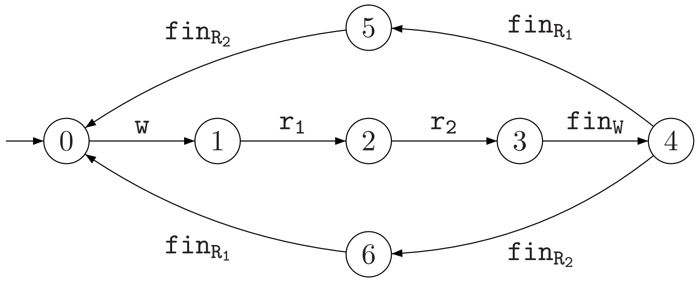


FIG. 2.6 – Exemple d'environnement pour l'AFR des lecteurs/écrivains.

de la figure 2.7 (page 53). La comparaison des deux figures (2.5 et 2.7) montre que la contrainte imposée par l'environnement conduit à ne considérer que certaines exécutions du système ouvert. Nous nous intéressons donc à des systèmes qui sont spécifiés avec une plus grande précision.

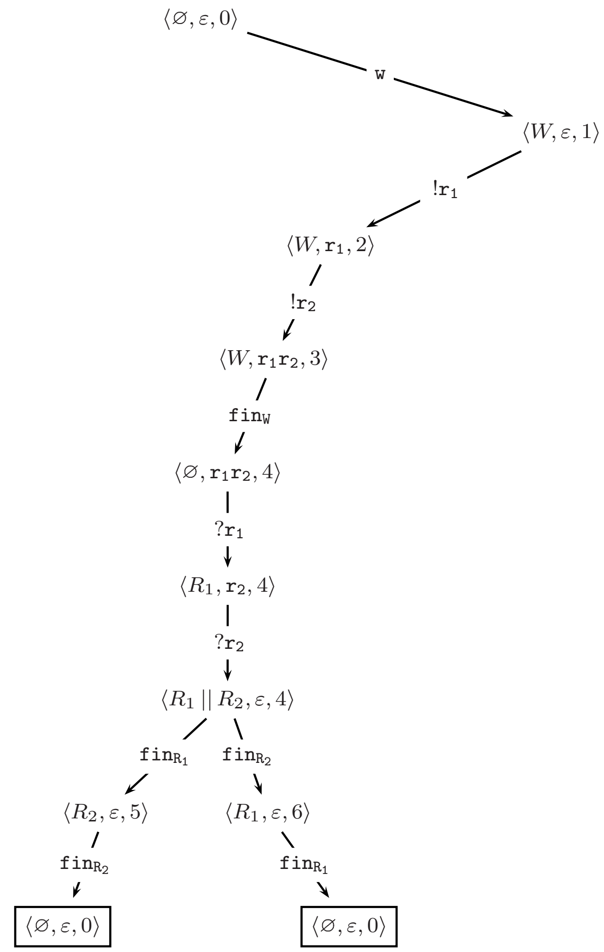


FIG. 2.7 – Arbre des exécutions du système à file réactif embarqué des lecteurs/écrivains.

Remarque 2.24

Notons que, par définition, un système à file réactif embarqué $S||E$ répond à toutes les sollicitations de son environnement : à toute transition de E correspond une transition dans $S||E$. De plus, un chemin fini de E conduit $S||E$ dans une configuration $\langle q_R, w, q_E \rangle$ où w est fini (par la remarque 2.12, page 45), depuis laquelle il ne peut donc franchir qu'une séquence de stabilisation (finie par définition, voir la remarque 2.13, page 45). **Par conséquent, $S||E$ termine ssi E termine.** ♦

Remarque 2.25
Un système à file réactif S, dont l'alphabet des événements est Σ, peut être vu comme un système à file réactif embarqué $S||E$, où E est un environnement non-contraignant. ♦

En ce sens, les auteurs de [91, 92] étudient la vérification des SFR Embarqués avec des environnements non contraignants.

Dans la suite, lorsque nous considérons un SFR Embarqué $S||E$, R désigne l'AFR qui engendre S.

2.4 Normalisation de la mémorisation : les Reset AFR

Les automates à file réactifs sont munis de deux types d'événements mémorisables : ceux dont au plus une occurrence est stockée et ceux dont toutes les occurrences qui ne peuvent pas être traitées immédiatement sont stockées. Dans cette section, nous montrons que les événements à mémorisation multiple permettent de simuler les événements à mémorisation unique. Pour cela, nous introduisons un nouvel opérateur, noté ?*, qui consomme toutes les occurrences mémorisées d'un événement au lieu de consommer uniquement l'exemplaire le plus ancien.

Définition 2.26 (Reset AFR)
Un *Reset automate à file réactif (Reset AFR)* est un quadruplet $R_{?*} = (Q_{R_{?*}}, q^0_{R_{?*}}, A_{R_{?*}}, \rightarrow_{R_{?*}})$ où $Q_{R_{?*}}$ et $q^0_{R_{?*}}$ sont définis de la même façon que Q_R et q^0_R pour les AFR (définition 2.5, page 42) et :

1. $A_{R_{?*}} = (\{!, ?, ?^*\} \times \Sigma_M) \cup \Sigma$ où les ensembles d'événements Σ, Σ_F, Σ_M, Σ_{M_1} et Σ_{M_*} sont définis comme pour les AFR (définition 2.5, page 42),
2. $\rightarrow_{R_{?*}} \subseteq Q_{R_{?*}} \times A_{R_{?*}} \times Q_{R_{?*}}$ est la relation de transition vérifiant $\forall q \in Q_{R_{?*}}$, $\mathsf{e} \in \Sigma_M$:
   (a) soit $q \xrightarrow{!\mathsf{e}}_{R_{?*}} q$,
   (b) sinon $\exists q \in Q_{R_{?*}}$ tel que $q \xrightarrow{\mathsf{e}}_{R_{?*}} q'$ et :
      i. soit $q \xrightarrow{?\mathsf{e}}_{R_{?*}} q'$,
      ii. sinon $\mathbf{q} \xrightarrow{?^*\mathsf{e}}_{R_{?*}} \mathbf{q}'$.

♦

L'opération reset est donc utilisée de la même façon que l'opération de traitement différé, c'est à dire couplée à une transition de traitement immédiat menant au même état.

Définition 2.27 (Reset SFR)
Le *Reset système à file réactif (Reset SFR)*, pour un Reset AFR $R_{?*} = (Q_{R_{?*}}, q^0_{R_{?*}}, A_{R_{?*}}, \rightarrow_{R_{?*}})$, est le système de transitions étiqueté $S_{?*} = (Q_{S_{?*}}, q^0_{S_{?*}}, A_{S_{?*}}, \rightarrow_{S_{?*}})$ dont les composantes $Q_{S_{?*}}$, $q^0_{S_{?*}}$, $A_{S_{?*}}$ sont définies de la même façon que Q_S, q^0_S et A_S pour les SFR (définition 2.11, page 44), et $\rightarrow_{S_{?*}}$ est étendue pour tenir compte de l'opération reset :

4(e). $(q,w) \xrightarrow{?^*\mathsf{e}}_{S_{?^*}} (q',w')$ si $q \xrightarrow{?^*\mathsf{e}}_{R_{?^*}} q'$ et $\exists w_1 \in (\Sigma^!_q)^*,\ \exists w_2,\ldots,w_n \in (\Sigma_M \setminus \{\mathsf{e}\})^*$ t.q. $w = w_1\mathsf{e}w_2\mathsf{e}\ldots\mathsf{e}w_n$ et $w' = w_1w_2\ldots w_n$.

♦

Ainsi, la condition $w_1 \in (\Sigma^!_q)^*$ assure que c'est la plus ancienne occurrence mémorisée consommable (par opération reset ou par consommation Fiffo classique) qui est traitée, alors que puisque $\forall i \in \{2,\ldots,n\},\ w_i \in (\Sigma_M \setminus \{\mathsf{e}\})^*$, toutes les occurrences mémorisées de e sont consommées.

La notion de configuration stable ou instable, et celle de séquence de stabilisation est naturellement étendue aux transitions reset. Les Reset SFR Embarqués sont obtenus depuis la définition 2.21 (page 51) en considérant que l'opération $?^*$ est interne.

Définition 2.28 (Reset SFR Embarqué)
Le *Reset système à file réactif embarqué (Reset SFR Embarqué)* défini par un par un Reset SFR $S_{?^*}$ et un environnement E, en appliquant la définitio 2.21 (page 51) étendue avec la règle :

(4d). $\langle q_R, w_1\mathsf{e}w_2\mathsf{e}\ldots\mathsf{e}w_n, q_E\rangle \xrightarrow{?^*e}_{S_{?^*}\,||\,E} \langle q_R, w_1w_2\ldots w_n, q_E\rangle$ si :

$$(q_R, w_1\mathsf{e}w_2\mathsf{e}\ldots\mathsf{e}w_n) \xrightarrow{?^*e}_{S_{?^*}} \langle q_R, w_1w_2\ldots w_n\rangle.$$

♦

Nous notons $d^o(R_{?^*}) = Card\,\{(q,\mathsf{e},q') \,|\, q \xrightarrow{?^*\mathsf{e}}_{R_{?^*}} q'\}$ le nombre de transitions reset (le *degré*) de $R_{?^*}$. Un Reset AFR $R_{?^*}$ est *reset-libre* si $d^o(R_{?^*}) = 0$.

Remarque 2.29
Notons que par définition de $?^*$, le long d'une séquence de stabilisation, il y a au plus une transition reset pour chacun des événements mémorisables. Donc $d^o(R_{?^*})$ est un majorant du nombre de transitions reset qui apparaîssent dans une séquence de stabilisation. ♦

Nous dirons qu'un (Reset) AFR est *normalisé* s'il ne possède aucun événement à mémorisation unique : $\Sigma_M = \Sigma_{M_*}$. Dans la suite de cette section, nous montrons les résultats suivants :

1. (Théorème 2.31, page 57) Pour tout AFR R, nous pouvons construire un Reset AFR Normalisé R_N tel que les ensembles des états de contrôle de R et R_N, accessibles dans S (le Reset SFR défini par R) et S_N (le SFR défini par R_N) sont les mêmes.
2. (Théorème 2.34, page 61) Pour tout Reset AFR $R_{?^*}$ (de sémantique $S_{?^*}$) plongé dans l'environnement $E_{?^*}$, nous pouvons construire un Reset-libre AFR R (de sémantique S) et un environnement E, tels que les ensembles des états de contrôle accessibles dans $S_{?^*}||E_{?^*}$ et dans $S||E$ sont les mêmes.

Alors, par le point (1) ci-dessus, à partir d'un AFR R, nous pouvons construire un Reset AFR Normalisé R_N. Puis, par le point (2), à partir de R_N, nous obtenons un nouvel AFR R', **sans opération reset** et **normalisé** qui préserve l'accessibilité des états de contrôle de R (en considérant l'embarquement approprié). Ceci nous permet dans la suite de nous affranchir des événements à mémorisation unique pour ne considérer que ceux à mémorisation multiple.

2.4.1 Normalisation de la mémorisation dans les AFR

Nous montrons maintenant, qu'il est possible de remplacer les événements à mémorisation unique d'un AFR par des événements à mémorisation multiple par l'ajout de l'opération *reset*, notée ?*. En effet, en transformant toutes les transitions de consommation correspondant à des événements à mémorisation unique par des transitions reset, du point de vue du système à file réactif obtenu, tout se passe comme si une seule occurrence de l'événement est mémorisée. L'interêt de cette transformation est de conduire la suite de cette étude en ne considérant que les événements à mémorisation multiple, plus simples à gérer que ceux à mémorisation unique.

Définition 2.30 (Reset AFR Normalisé)
Soit R un AFR sans transitions Reset, *le Reset AFR Normalisé* R_N est obtenu depuis R par :

1. $Q_{R_N} = Q_R$ et $q^0_{R_N} = q^0_R$,
2. $(\Sigma_{M_*})_{R_N} = (\Sigma_M)_R$, $(\Sigma_{M_1})_{R_N} = \emptyset$, $(\Sigma_F)_{R_N} = (\Sigma_F)_R$ et $A_{R_N} = A_R \cup (\{?^*\} \times (\Sigma_{M_1})_R)$,
3. $\rightarrow_{R_N} \subseteq Q_{R_N} \times A_{R_N} \times Q_{R_N}$ est la relation de transition obtenue depuis $\rightarrow_R$ par :

$$\forall\, q_R \xrightarrow{a}_R q'_R \qquad \begin{cases} q_{R_N} \xrightarrow{?^*\mathsf{e}}_{R_N} q'_{R_N} & \text{si } a = ?\mathsf{e} \text{ et } \mathsf{e} \in (\Sigma_{M_1})_R \\ q_{R_N} \xrightarrow{a}_{R_N} q'_{R_N} & \text{sinon} \end{cases}$$

♦

Soit S le SFR défini par R (définition 2.11, page 44) et S_N le Reset SFR qui donne la sémantique de R_N (définition 2.27, page 54). Nous prouvons que la normalisation des AFR préserve l'accessibilité des états de contrôle de $Q_R = Q_{R_N}$. Pour cela, nous introduisons une relation d'équivalence entre les configurations de S et S_N, de façon à capturer la simulation des événements à mémorisation unique de R. Soient (q_R, w_R) et (q_{R_N}, w_{R_N}) deux configurations de S et S_N respectivement. Soit $\sim_N$ la relation d'équivalence définie par :

$$(q_R, w_R) \sim_N (q_{R_N}, w_{R_N}) \;\Leftrightarrow\; q_R = q_{R_N} \quad \text{et} \begin{cases} \forall \mathsf{e} \in (\Sigma_{M_1})_R,\ |w_R|_\mathsf{e} = 0 \Leftrightarrow |w_{R_N}|_\mathsf{e} = 0 \\ \forall \mathsf{e} \in (\Sigma_{M_1})_R,\ |w_R|_\mathsf{e} = 1 \Leftrightarrow |w_{R_N}|_\mathsf{e} \geq 1 \\ \forall \mathsf{e} \in (\Sigma_{M_*})_R,\ |w_R|_\mathsf{e} = |w_{R_N}|_\mathsf{e} \end{cases}$$

Nous définissons la simulation de S par S_N de la façon suivante :

- Pour toutes configurations $(q_R, w_R), (q'_R, w'_R) \in RS(S)$ et $(q_{R_N}, w_{R_N}) \in RS(S_N)$ telles que :
  - $(q_R, w_R) \sim_N (q_{R_N}, w_{R_N})$,
  - et $(q_R, w_R) \rightarrow_S (q'_R, w'_R)$,

  il existe une configuration $(q'_{R_N}, w'_{R_N}) \in RS(S_N)$ telle que :
  - $(q_{R_N}, w_{R_N}) \rightarrow_{S_N} (q'_{R_N}, w'_{R_N})$,
  - $(q'_R, w'_R) \sim_N (q'_{R_N}, w'_{R_N})$.
- Et réciproquement, pour toutes configurations $(q_{R_N}, w_{R_N}), (q'_{R_N}, w'_{R_N}) \in RS(S_N)$ et $(q_R, w_R) \in RS(S)$ telles que :
  - $(q_R, w_R) \sim_N (q_{R_N}, w_{R_N})$,

– et $(q_{R_N}, w_{R_N}) \rightarrow_{S_N} (q'_{R_N}, w'_{R_N})$,
il existe une configuration $(q'_R, w'_R) \in RS(S)$ telle que :
– $(q_R, w_R) \rightarrow_S (q'_R, w'_R)$,
– et $(q'_R, w'_R) \sim_N (q'_{R_N}, w'_{R_N})$.

Théorème 2.31
Pour tout AFR R, de sémantique S, il existe un Reset AFR Normalisé R_N, de sémantique S_N, tel que S_N simule S. ♦

Preuve. Notons que la différence entre R et R_N réside dans la gestion des occurrences mémorisées des événements à mémorisation unique de R. Nous ne nous intéressons par conséquent qu'aux transitions correspondantes. Nous montrons le résultat de simulation par induction.

Les configurations initiales de S et S_N, (q^0_R, ε) et $(q^0_{R_N}, \varepsilon)$ respectivement, sont $\sim_N$-équivalentes et accessibles.

Examinons tout d'abord le cas des transitions de mémorisation pour les événements à mémorisation unique de R. Ces transitions sont laissées intactes par la normalisation, seul le type de l'événement change. Il vient donc aisément qu'à toute transition de mémorisation de R correspond une transition de mémorisation de R_N et réciproquement. Finalement, considérons deux configurations $\sim_N$-équivalentes : $(q_R, w_R) \in RS(S)$ et $(q_{R_N}, w_{R_N}) \in RS(S_N)$ et la mémorisation de e, à mémorisation unique dans R. Deux cas se présentent :

– soit $|w_R|_{\mathsf{e}} = |w_{R_N}|_{\mathsf{e}} = 0$ et après la mémorisation de e, les configurations atteintes sont $(q_R, w_R.\mathsf{e})$ et $(q_{R_N}, w_{R_N}.\mathsf{e})$ dans S et S_N respectivement. Alors, la simulation est prouvée puisque $|w_R.\mathsf{e}|_{\mathsf{e}} = |w_{R_N}.\mathsf{e}|_{\mathsf{e}} = 1$,
– soit $|w_R|_{\mathsf{e}} = 1$ et $|w_{R_N}|_{\mathsf{e}} \geq 1$ et après la mémorisation de e, les configurations atteintes sont (q_R, w_R) et $(q_{R_N}, w_{R_N}.\mathsf{e})$ dans S et S_N respectivement. Alors, la simulation est prouvée puisque $|w_R|_{\mathsf{e}} = 1$ et $|w_{R_N}.\mathsf{e}|_{\mathsf{e}} \geq |w_{R_N}.|_{\mathsf{e}} \geq 1$.

Finalement, il reste à examiner le traitement des occurrences mémorisées des événements à mémorisation unique de R. Nous allons détailler ce cas qui est le point clef de la construction.

– Soient trois configurations $(q_R, w_R), (q'_R, w'_R) \in RS(S)$ et $(q_{R_N}, w_{R_N}) \in RS(S_N)$ telles que :
  – $(q_R, w_R) \sim_N (q_{R_N}, w_{R_N})$,
  – et $(q_R, w_R) \xrightarrow{?\mathsf{e}}_S (q'_R, w'_R)$ avec $\mathsf{e} \in (\Sigma_{M_1})_R$, et :
    – $|w'_R|_{\mathsf{e}} = 0$,
    – quel que soit $\mathsf{e}' \neq \mathsf{e}$, $|w'_R|_{\mathsf{e}'} = |w_R|_{\mathsf{e}'}$.

  Par construction, S_N franchit la transition reset $(q_{R_N}, w_{R_N}) \xrightarrow{?^*\mathsf{e}}_{S_N} (q'_{R_N}, w'_{R_N})$, avec $q'_{R_N} = q'_R$. Par la sémantique des Reset SFR (définition 2.27, page 54), nous avons :
  – $|w'_{R_N}|_{\mathsf{e}} = 0$,
  – quel que soit $\mathsf{e}' \neq \mathsf{e}$, $|w'_{R_N}|_{\mathsf{e}'} = |w_{R_N}|_{\mathsf{e}'}$.

  Nous en déduisons la $\sim_N$-équivalence de (q'_R, w'_R) et (q'_{R_N}, w'_{R_N}).
– Réciproquement, soient deux configurations $(q_{R_N}, w_{R_N}), (q'_{R_N}, w'_{R_N}) \in RS(S_N)$ et une configuration $(q_R, w_R) \in RS(S)$ telles que :
  – $(q_R, w_R) \sim_N (q_{R_N}, w_{R_N})$,

– et $(q_{R_N}, w_{R_N}) \xrightarrow{?^*\mathsf{e}}_{S_N} (q'_{R_N}, w'_{R_N})$ avec $\mathsf{e} \in (\Sigma_{M_1})_R$ et, par la sémantique des Reset SFR (définition 2.27, page 54) :
  – $|w'_{R_N}|_{\mathsf{e}} = 0$,
  – quel que soit $\mathsf{e}' \neq \mathsf{e}$, $|w'_{R_N}|_{\mathsf{e}'} = |w_{R_N}|_{\mathsf{e}'}$.

Par construction, S franchit la transition $(q_R, w_R) \xrightarrow{?\mathsf{e}}_S (q'_R, w'_R)$ avec $q'_R = q'_{R_N}$ et :
– $|w'_R|_{\mathsf{e}} = 0$,
– et quel que soit $\mathsf{e}' \neq \mathsf{e}$, $|w'_R|_{\mathsf{e}'} = |w_R|_{\mathsf{e}'}$.

Nous en déduisons : $(q'_R, w'_R) \sim_N (q_{R_N}, w_{R_N})$.

♦

Remarque 2.32
Notre construction ne préserve pas la propriété de bornitude (B). En effet, les occurrences d'un événement à mémorisation unique qui surviennent alors qu'il n'est pas possible de les traiter sont oubliées s'il y a déjà une occurrence de cet événement qui est mémorisée. Par contre, lorsque cet événement est simulé par un événement à mémorisation multiple, toutes les occurrences de ce dernier sont mémorisées lorsqu'il n'est pas possible de les traiter immédiatement, d'où la possibilité d'avoir un modèle normalisé non borné alors que le modèle initial est borné. ♦

Désormais nous considérons que tous les événements mémorisables des automates à file réactifs sont à mémorisation multiple, et nous identifions Σ_M à Σ_{M_*}. Le point (4a) de la définition 2.11 (page 44) est simplifié de la façon suivante :

4a. $(q, w) \xrightarrow{!\mathsf{e}}_S (q, w\mathsf{e})$ si $q \xrightarrow{!\mathsf{e}}_R q$ et $w \in \left(\Sigma^!_q\right)^*$

2.4.2 Des Reset AFR aux Reset-libres AFR

Principe de la simulation

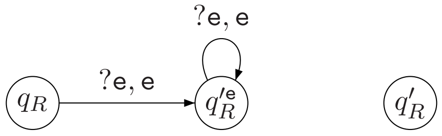
Considérons un Reset AFR $R_{?*}$ et l'une de ses transitions reset (avec, bien sûr, sa transition associée de traitement immédiat) :

$$q_{R_{?*}} \xrightarrow{?^*\mathsf{e},\, \mathsf{e}} q'_{R_{?*}} \tag{2.2}$$

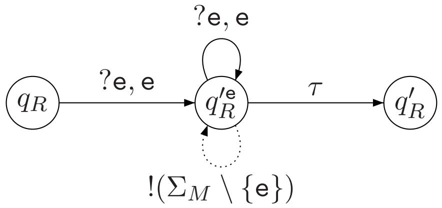
Nous construisons une structure d'AFR R qui simule cette transition sans utiliser l'opération reset ?*. La simulation que nous réalisons est telle que toute configuration (q, w), $q \in Q_{R_{?*}}$, est accessible dans $R_{?*}$ si et seulement si elle l'est dans R. Nous considérons qu'à $R_{?*}$ est associé un environnement $E_{?*}$ quelconque, et nous construisons R et son environnement E, pour parvenir à ce résultat de simulation.

Il est clair que nous devons utiliser l'opération de consommation pour simuler l'opération reset. Comme l'opération reset permet de consommer un nombre non borné d'occurrences d'événements mémorisées, il faut utiliser une boucle de traitement différé. Cependant, si q_R et q'_R sont les deux états de R qui simulent $q_{R_{?*}}$ et $q'_{R_{?*}}$ respectivement, il n'est pas possible de placer cette boucle sur q_R ni sur q'_R, puisque l'opération reset doit être simulée malgré les priorités entre les occurrences mémorisées. En effet, s'il y a en $q_{R_{?*}}$ une transition de consommation d'un événement e' (quittant l'état $q_{R_{?*}}$) en concurrence avec la transition $?^*\mathsf{e}$, et si une occurrence de e' précède, dans la file, certaines des occurrence de e, ces dernières ne seront pas consommées lors de l'itération de la boucle de consommation de e, alors qu'elles devraient l'être.

Par conséquent, il est nécessaire d'ajouter un nouvel état, nommé $q_R'^{\mathsf{e}}$, ce qui conduit pour le moment à la structure suivante :



Lorsqu'à l'arrivée en $q_{R_{?*}}$ il n'y a aucune occurrence mémorisée de e, $S_{?*}$ passe de $q_{R_{?*}}$ à $q'_{R_{?*}}$ par la transition de traitement immédiat de e. Pour que R simule la transition de (2.2), il faut donc que le passage de q_R à q'_R ne nécessite qu'une seule occurrence de e parmi tous les événements de $R_{?*}$. Nous introduisons donc dans R un nouvel événement fugace, τ, et R devient alors :



(2.3)

Notons que nous avons complété la structure en $q_R'^{\mathsf{e}}$ vis-à-vis des événements mémorisables[5]. Par construction, la relation de transition en q_R et q'_R est aussi complète.

L'environnement E de R est obtenu depuis $E_{?*}$ en faisant suivre chaque occurrence d'événement, de $d^o(R_{?*})$ occurrence de τ, un nouvel événement. Cette hypothèse nous permet de simuler n'importe quelle séquence de stabilisation de $R_{?*}$. En effet, supposons que nous devons simuler l'enchaînement de deux transitions reset : $?^*\mathsf{e}$ et $?^*\mathsf{e}'$. D'après notre construction, R franchit autant de transitions de consommation $?\mathsf{e}$ qu'il est nécessaire, puis il doit franchir une transition étiquetée τ, puis il franchit autant de transition de consommation $?\mathsf{e}'$ qu'il le faut, et finalement, il franchit une transition d'étiquette τ. Il est par conséquent nécessaire que l'environnement de R lui fournisse deux occurrences successives de τ pour que son comportement soit conforme à nos attentes. Notons que :

– durant le franchissement des deux transitions reset, aucun événement n'a été pris en compte depuis $E_{?*}$, et il doit en être de même pour E (exception faite des occurrences de τ). D'une façon générale, pendant le franchissement d'une séquence de stabilisation dans $R_{?*}$, aucune occurrence d'événement n'est prise en compte depuis $E_{?*}$, et il doit en être de même lorsque R simule cette séquence de stabilisation (exception faite des occurrences de τ). Les $d^o(R_{?*})$ occurrences consécutives de τ sont nécessaires pour cela.
– toute séquence de stabilisation dans $R_{?*}$ contient au plus $d^o(R_{?*})$ transitions reset par la remarque 2.29, page 55. Nous en déduisons la suffisance des $d^o(R_{?*})$ occurrences consécutives de τ.

[5] L'étiquette $!(\Sigma_M \setminus \{\mathsf{e}\})$ représente les étiquettes $!\mathsf{e}'$, avec $\mathsf{e}' \in (\Sigma_M \setminus \{\mathsf{e}\})$

Ainsi, en supposant que chaque occurrence d'événement $\mathsf{e} \in (\Sigma)_{R_{?^*}}$ est suivie de $d^o(R_{?^*})$ occurrences de τ, nous avons la certitude que R peut simuler les séquences de stabilisation de $R_{?^*}$.

Simulation d'un Reset SFR par son Reset-libre SFR associé

En suivant la construction décrite dans le précédent paragraphe, nous obtenons pour tout Reset AFR $R_{?^*}$ un Reset-libre AFR R qui simule $R_{?^*}$.

Définition 2.33 (Reset-libre AFR associé)
Soit $R_{?^*} = (Q_{R_{?^*}}, q^0_{R_{?^*}}, A_{R_{?^*}}, \rightarrow_{R_{?^*}})$ un Reset AFR. Le *Reset-libre AFR associé à* R est l'AFR $R = (Q_R, q^0_R, A_R, \rightarrow_R)$ défini par :

1. $Q_R = \{q_R \,|\, q_{R_{?^*}} \in Q_{R_{?^*}}\} \cup \{q'^{\mathsf{e}}_R \,|\, q_{R_{?^*}} \xrightarrow{?^*\mathsf{e}}_{R_{?^*}} q'_{R_{?^*}}\}$,
2. $q^0_R = q^0_{R_{?^*}}$,
3. $A_R = \big(A_{R_{?^*}} \setminus (\{?^*\} \times \Sigma_M)\big) \cup \{\tau\}$, avec $\tau \notin (\Sigma)_{R_{?^*}}$,
4. $\rightarrow_R \subseteq Q_R \times A_R \times Q_R$ est définie par :
   (a) $q_R \xrightarrow{!\mathsf{e}}_R q_R$ si $q_{R_{?^*}} \xrightarrow{!\mathsf{e}}_{R_{?^*}} q_{R_{?^*}}$,
   (b) $q_R \xrightarrow{?\mathsf{e},\,\mathsf{e}}_R q'_R$ si $q_{R_{?^*}} \xrightarrow{?\mathsf{e},\,\mathsf{e}}_{R_{?^*}} q'_{R_{?^*}}$,
   (c) $q_R \xrightarrow{?\mathsf{e},\,\mathsf{e}}_R q'^{\mathsf{e}}_R$, $q'^{\mathsf{e}}_R \xrightarrow{?\mathsf{e},\,\mathsf{e}}_R q'^{\mathsf{e}}_R$, $q'^{\mathsf{e}}_R \xrightarrow{\tau}_R q'_R$ et $\forall \mathsf{e}' \in (\Sigma_M \setminus \mathsf{e})$, $q'^{\mathsf{e}}_R \xrightarrow{!\mathsf{e}'}_R q'^{\mathsf{e}}_R$
   si $q_{R_{?^*}} \xrightarrow{?^*\mathsf{e},\,\mathsf{e}}_{R_{?^*}} q'_{R_{?^*}}$,
   (d) $q_R \xrightarrow{\mathsf{e}}_R q'_R$ si $q_{R_{?^*}} \xrightarrow{\mathsf{e}}_{R_{?^*}} q'_{R_{?^*}}$ et $\mathsf{e} \in \Sigma_F$.

et chaque occurrence de $\mathsf{e} \in (\Sigma)_{R_{?^*}}$ dans l'environnement de R est suivie de $d^o(R_{?^*})$ occurrences de τ. ♦

Ainsi, toute transition $q \xrightarrow{\mathsf{e}}_{E_{?^*}} q'$ de $E_{?^*}$ est transcrite, dans E, par la séquence :

$$q \xrightarrow{\mathsf{e}}_E q^1 \xrightarrow{\tau}_E \ldots \xrightarrow{\tau}_E q^{d^o(R_{?^*})} \xrightarrow{\tau}_E q'$$

Notons $S_{?^*}||E_{?^*}$ le Reset SFR Embarqué obtenus depuis $R_{?^*}$ et $E_{?^*}$ (définition 2.28, page 55), et $S||E$ le SFR Embarqué construit à partir de R et E (définition 2.21, page 51). Soient $\langle q_{R_{?^*}}, w_{R_{?^*}}, q_{E_{?^*}}\rangle$ et $\langle q_R, w_R, q_E\rangle$ deux configurations de $S_{?^*}||E_{?^*}$ et $S||E$ respectivement. Soit $\sim_{?^*}$ la relation d'équivalence entre ces configurations définie par :

$$\langle q_{R_{?^*}}, w_{R_{?^*}}, q_{E_{?^*}}\rangle \sim_{?^*} \langle q_R, w_R, q_E\rangle \quad \Leftrightarrow \quad q_{R_{?^*}} = q_R,\ w_{R_{?^*}} = w_R \text{ et } \begin{cases} \text{soit} & q_E = q_{E_{?^*}} \\ \text{sinon} & q_E \xrightarrow{\tau}{}^*_E\, q_{E_{?^*}} \end{cases}$$

La simulation de $S_{?^*}||E_{?^*}$ par $S||E$ est définie par :

- Pour toutes configurations $\langle q_{R_{?^*}}, w_{R_{?^*}}, q_{E_{?^*}}\rangle, \langle q'_{R_{?^*}}, w'_{R_{?^*}}, q'_{E_{?^*}}\rangle \in RS(S_{?^*}||E_{?^*})$, ainsi que $\langle q_R, w_R, q_E\rangle \in RS(S||E)$ telles que :
  - $\langle q_{R_{?^*}}, w_{R_{?^*}}, q_{E_{?^*}}\rangle \sim_{?^*} \langle q_R, w_R, q_E\rangle$,
  - $\langle q_{R_{?^*}}, w_{R_{?^*}}, q_{E_{?^*}}\rangle \rightarrow_{S_{?^*}||E_{?^*}} \langle q'_{R_{?^*}}, w'_{R_{?^*}}, q'_{E_{?^*}}\rangle$,

  il existe une configuration $(q'_R, w'_R, q'_E) \in RS(S||E)$ telle que :

– $\langle q_R, w_R, q_E\rangle \xrightarrow{\sigma}{}^*_{S||E} \langle q'_R, w'_R, q'_E\rangle$ et pour toute configuration $\langle q''_R, w''_R, w''_E\rangle$ de σ, $q''_R \notin Q_{R_{?*}}$,
– $\langle q'_{R_{?*}}, w'_{R_{?*}}, q'_{E_{?*}}\rangle \sim_{?*} \langle q'_R, w'_R, q'_E\rangle$.

– Et réciproquement, pour toutes configurations $\langle q_R, w_R, q_E\rangle, \langle q'_R, w'_R, q'_E\rangle \in RS(S||E)$ et $\langle q_{R_{?*}}, w_{R_{?*}}, q_{E_{?*}}\rangle \in RS(S_{?*}||E_{?*})$ telles que :
– $\langle q_{R_{?*}}, w_{R_{?*}}, q_{E_{?*}}\rangle \sim_{?*} \langle q_R, w_R, q_E\rangle$,
– et $\langle q_R, w_R, q_E\rangle \xrightarrow{\sigma}_{S||E} \langle q'_R, w'_R, q'_E\rangle$ et pour toute configuration $\langle q''_R, w''_R, q''_E\rangle$ de σ, $q''_R \notin Q_{R_{?*}}$,

il existe une configuration $\langle q'_{R_{?*}}, w'_{R_{?*}}, q'_{E_{?*}}\rangle \in RS(S_{?*}||E_{?*})$ telle que :
– $\langle q_{R_{?*}}, w_{R_{?*}}, q_{E_{?*}}\rangle \rightarrow_{S_{?*}||E_{?*}} \langle q'_{R_{?*}}, w'_{R_{?*}}, q'_{E_{?*}}\rangle$,
– $\langle q'_{R_{?*}}, w'_{R_{?*}}, q'_{E_{?*}}\rangle \sim_{?*} \langle q'_R, w'_R, q'_E\rangle$.

Théorème 2.34

Pour tout Reset AFR $R_{?*}$ de sémantique $S_{?*}$, plongé dans un environnement $E_{?*}$, il existe un Reset-libre AFR R, de sémantique S, plongé dans une environnement E tel que $S||E$ simule $S_{?*}||E_{?*}$. ♦

Preuve. Remarquons que S est complet vis-à-vis des événements fugaces, donc les occurrences de τ reçues dans des configurations où elles ne sont pas attendues sont simplement ignorées. Nous prouvons maintenant le résultat de simulation par induction. Puisque $\rightarrow_{R_{?*}}$ et $\rightarrow_R$ ne diffèrent que pour les transitions reset, nous nous concentrons uniquement sur ce cas.

Notons tout d'abord que les configurations initiale de $S_{?*}||E_{?*}$ et $S||E$, $\langle q^0_{R_{?*}}, \varepsilon, q^0_{E_{?*}}\rangle$ et $\langle q^0_R, \varepsilon, q^0_E\rangle$ respectivement, sont accessibles et $\sim_{?*}$-équivalentes.

– Soient trois configurations $\langle q_{R_{?*}}, w_{R_{?*}}, q_{E_{?*}}\rangle, \langle q'_{R_{?*}}, w'_{R_{?*}}, q'_{E_{?*}}\rangle \in RS(S_{?*}||E_{?*})$, ainsi que $\langle q_R, w_R, q_E\rangle \in RS(S||E)$ telles que :
– $\langle q_{R_{?*}}, w_{R_{?*}}, q_{E_{?*}}\rangle \sim_{?*} \langle q_R, w_R, q_E\rangle$,
– $\langle q_{R_{?*}}, w_{R_{?*}}, q_{E_{?*}}\rangle \xrightarrow{?^*\mathsf{e}}_{S_{?*}||E_{?*}} \langle q'_{R_{?*}}, w'_{R_{?*}}, q'_{E_{?*}}\rangle$,

Soient $w_1 \in (\Sigma^!_{q_{R_{?*}}})^*$ et $w_2, \ldots, w_n \in (\Sigma_M \setminus \{\mathsf{e}\})^*$ tels que $w_{R_{?*}} = w_1\mathsf{e}w_2\mathsf{e}\ldots\mathsf{e}w_n$. Par la définition de l'opération reset, $w'_{R_{?*}} = w_1w_2\ldots w_n$. Par la définition de $\sim_{?*}$, $w_R = w_1\mathsf{e}w_2\mathsf{e}\ldots\mathsf{e}w_n$, et par construction, $S||E$ franchit la transition :

$$\langle q_R, w_1\mathsf{e}w_2\mathsf{e}\ldots\mathsf{e}w_n, q_E\rangle \xrightarrow{?\mathsf{e}}_{S||E} \langle q'^{\mathsf{e}}_R, w_1w_2\mathsf{e}\ldots\mathsf{e}w_n, q_E\rangle$$

Puis, par priorité au traitement des occurrences mémorisées, et tant qu'il y a des occurrences mémorisées de e, $S||E$ exécute la boucle de consommation de e sur q'^{e}_R, c'est à dire la séquence :

$$\langle q'^{\mathsf{e}}_R, w_1w_2\mathsf{e}\ldots\mathsf{e}w_n, q_E\rangle \xrightarrow{?\mathsf{e}}_{S||E} \langle q'^{\mathsf{e}}_R, w_1w_2\ldots\mathsf{e}w_n, q_E\rangle \xrightarrow{?\mathsf{e}}_{S||E} \langle q'^{\mathsf{e}}_R, w_1w_2\ldots w_n, q_E\rangle$$

Notons que nous avons nécessairement $q_E \xrightarrow{\tau}{}^*_E q_{E_{?*}}$ et $q_E \neq q_{E_{?*}}$. En effet, la transition reset que nous simulons appartient à une séquence de stabilisation[6] qui, par le lemme 2.14 (page 46) est précédée par une transition de traitement immédiat, contrôlée par l'environnement. Par construction de E, après l'émission de cette occurrence, E pro-

[6]Éventuellement limitée à cette seule transition.

pose consécutivement, $d^o(R_{?^*})$ occurrences de τ. Or, par la remarque 2.29 (page 55), une séquence de stabilisation contient au plus $d^o(R_{?^*})$ transitions reset, et par conséquent, E propose à nouveau une occurrence de τ.
$S||E$ franchit finalement la transition :

$$\langle q'^{\mathsf{e}}_R, w_1 w_2 \dots w_n, q_E\rangle \xrightarrow{\tau}_{S||E} \langle q'_R, w_1 w_2 \dots w_n, q'_E\rangle$$

Enfin, $q'^{\mathsf{e}}_R \notin Q_R$, donc la $\sim_{?^*}$-équivalence des configurations ainsi atteintes par $S_{?^*}||E_{?^*}$ et $S||E$ est assurée.

– Réciproquement, considérons trois configurations $\langle q_R, w_R, q_E\rangle, \langle q'_R, w'_R, q'_E\rangle \in RS(S||E)$ et $\langle q_{R_{?^*}}, w_{R_{?^*}}, q_{E_{?^*}}\rangle \in RS(S_{?^*}||E_{?^*})$ telles que :
– $\langle q_{R_{?^*}}, w_{R_{?^*}}, q_{E_{?^*}}\rangle \sim_{?^*} \langle q_R, w_R, q_E\rangle$,
– et $\langle q_R, w_R, q_E\rangle \xrightarrow{\sigma}_{S||E} \langle q'_R, w'_R, q'_E\rangle$ et pour toute configuration $\langle q''_R, w''_R, q''_E\rangle$ de σ, $q''_R \notin Q_{R_{?^*}}$,

Puisque nous ne nous intéressons qu'à la simulation des transitions reset, la séquence σ, correspond justement à la simulation d'une telle transition. Il existe donc $w_1 \in (\Sigma^!_{q_R})^*$ et $w_2, \dots, w_n \in (\Sigma_M \setminus \{\mathsf{e}\})^*$ tels que $w_R = w_1 \mathsf{e} w_2 \mathsf{e} \dots \mathsf{e} w_n$ et $w'_R = w_1 w_2 \dots w_n$, c'est à dire que la séquence σ correspond à :

$$\begin{array}{l}\langle q_R, w_1 \mathsf{e} w_2 \mathsf{e} \dots \mathsf{e} w_n, q_E\rangle \xrightarrow{?\mathsf{e}}_{S||E} \langle q'^{\mathsf{e}}_R, w_1 w_2 \mathsf{e}\mathsf{e} w_n, q_E\rangle \xrightarrow{?\mathsf{e}}_{S||E} \dots \\ \qquad \langle q'^{\mathsf{e}}_R, w_1 w_2 \dots \mathsf{e} w_n, q_E\rangle \xrightarrow{?\mathsf{e}}_{S||E} \langle q'^{\mathsf{e}}_R, w_1 w_2 \dots w_n, q_E\rangle \xrightarrow{\tau}_{S||E} \langle q'_R, w_1 w_2 \dots w_n, q'_E\rangle\end{array}$$

et $q'^{\mathsf{e}}_R \notin Q_R$. Par construction, dans $S_{?^*}||E_{?^*}$, la transition reset correspondante est franchie. Notons que par définition de $\sim_{?^*}$, $w_{R_{?^*}} = w_1 \mathsf{e} w_2 \mathsf{e} \dots \mathsf{e} w_n$, et il existe donc une configuration $\langle q'_{R_{?^*}}, w_1 w_2 \dots w_n, q_{E_{?^*}}\rangle$ telle que :

$$\langle q_{R_{?^*}}, w_1 \mathsf{e} w_2 \mathsf{e} \dots \mathsf{e} w_n, q_{E_{?^*}}\rangle \xrightarrow{?^*\mathsf{e}}_{S_{?^*}||E_{?^*}} \langle q'_{R_{?^*}}, w_1 w_2 \dots w_n, q_{E_{?^*}}\rangle$$

Enfin, $q'_E = q_{E_{?^*}}$ ou $q'_E \xrightarrow{\tau}{}^*_E\, q_{E_{?^*}}$ puisque par hypothèse $q_E \xrightarrow{\tau}{}^*_E\, q_{E_{?^*}}$ et $q_E \neq q_{E_{?^*}}$ car pour le franchissement de σ, E a nécessairement proposé une occurrence de τ. Il apparaît alors la relation de simulation puisque : $\langle q'_{R_{?^*}}, w_1 w_2 \dots w_n, q_{E_{?^*}}\rangle \sim_{?^*} \langle q'_R, w_1 w_2 \dots w_n, q'_E\rangle$

♦

Dans cette première partie, nous avons défini la problèmatique de notre thèse. Nous considérons des systèmes réactifs, avec le point de vue de la programmation asynchrone. Dans ce cadre, il est fait l'hypothèse qu'il existe toujours un grain de temps assez fin pour distinguer deux événements consécutifs. Nous nous intéressons particulièrement au langage de programmation ELECTRE au sein duquel, les hypothèses asynchrones s'expriment par :

- les modules (blocs élémentaires de code exécutable) ont une durée finie, et non nulle (il est possible d'en distinguer le début de la fin),
- deux occurrences successives d'un même événement ne peuvent pas être simultanées.

Puisque les modules ont une durée finie, une occurrence d'événement peut se produire alors qu'un module s'exécute, sans qu'il ne soit possible de l'interrompre pour traiter cette occurrence. Elle est alors mémorisée et traitée ultérieurement, en respect de la politique FIFFO (First In First Fireable Out). La sémantique des programmes ELECTRE est décrite par les automates à file réactifs, qui sont donc munis d'une file (à gestion FIFFO) pour la mémorisation des événements.

Une propriété fondamentale des automates à file réactifs est la bornitude de la file de mémorisation. En effet, toute application réelle ne peut mémoriser qu'un nombre borné d'éléments : un dispositif physique de mémorisation ne peut contenir qu'un nombre borné d'éléments. Nous choisissons donc l'étude de cette propriété comme point central de notre thèse. Cette propriété dépend fortement de la façon qu'ont les événements de survenir. Or, cette dernière spécification fait défaut aux programmes ELECTRE comme aux automates à file réactifs. Nous avons introduit la spécification de l'environnement des automates à file réactifs, donnant lieu aux systèmes à file réactifs.

Finalement, les événements mémorisables sont de deux types en ELECTRE (et donc, dans les automates à file réactifs) : à mémorisation unique, ou à mémorisation multiple. Nous avons montré que les événements à mémorisation multiple permettent de simuler ceux à mémorisation unique. Ceci nous permet de ne plus considérer que les premiers dans la suite de notre thèse.

Deuxième partie

Expressivité des systèmes à file réactifs embarqués

Nous étudions, dans les trois chapitres à venir, les *systèmes à file réactifs embarqués* introduits en fin du chapitre précédent (en section 2.3.2, page 51). La motivation première pour la spécification explicite de l'environnement est l'étude de la bornitude des systèmes à file réactifs. Ce problème, ainsi que l'ensemble des problèmes d'interêt (présentés en section 1.3.3, page 30) sont examinés dans le chapitre 3. Nous montrons que les systèmes à file réactifs ont la puissance des machines de Turing dès lors qu'ils disposent d'au moins deux événements mémorisables et d'un environnement périodique (théorème 3.6, page 76), et qu'à l'inverse tous les problèmes soulevés sont décidables lorsqu'il n'y a pas plus d'un événement mémorisable et que l'environnement est rationnel (théorème 3.7, page 79). Les environnement périodiques forment une classe particulièrement intéressante pour l'étude des systèmes temps-réels qui sont souvent conçus pour évoluer de façon périodique dans de tels contextes. Le premier chapitre se termine donc par l'étude de la périodicité de l'exécution des systèmes à file réactifs plongés en environnement périodique.

Les résultats d'indécidabilité du premier chapitre nous incite à examiner, dans le chapitre 4 (page 89) l'effet d'une file de mémorisation non fiable sur les résultats d'indécidabilité. Suivant une méthode désormais classique [1, 2], nous introduisons les *Lossy SFR Embarqués*, pour lesquels certains problèmes, dont l'accessibilité, deviennent décidables. Ceci permet de vérifier les propriétés de sûreté de la forme «une mauvaise configuration n'est jamais accessible». De plus, l'introduction des pertes, non déterministes et incontrollées, d'occurrences mémorisées ajoute des comportements au système. Dans le cas où une telle propriété de «non-accessibilité» est vérifiée pour un système avec perte, elle l'est aussi pour le système sans perte.

Cependant, le problème de la bornitude reste indécidable pour les Lossy SFR Embarqués. Pourtant, cette propriété est particulièrement importante, que ce soit dans un but de vérification : les techniques utilisées pour les systèmes finis et infinis diffèrent fortement, ou dans un but d'implantation du système, puisqu'aucune mémoire physique ne peut contenir un nombre non borné d'éléments. Nous introduisons donc dans le chapitre 5, une méthode de test de la non-bornitude pour les systèmes à file réactifs embarqués.

Chapitre 3

Systèmes à file réactifs embarqués

Dans [91, 92], les auteurs considèrent la vérification des systèmes à file réactifs. Rappelons qu'un SFR S peut être vu comment un SFR Embarqué $S||E$ dont l'environnement est non-contraignant (remarque 2.25, page 53). Ici, nous considérons la vérification des systèmes à file réactifs embarqués munis d'environnements quelconques. Notre but est de nous intéresser uniquement aux comportements *réels* des SFR, c'est à dire, les comportements de l'application modélisée. Pour cela, il est nécessaire de clore la spécification en explicitant le mode d'occurrence des événements pour le système modélisé. Les problèmes de vérification, tels que l'accessibilité et la bornitude en dépendent, puisque l'ensemble des configurations accessibles de $S||E$ est *a priori* inclus (i.e. plus petit) dans celui de S. Particulièrement, le problème de la bornitude prend vraiment son sens lorsque le mode d'occurrence des événements est connu.

Rappelons que les résultats positifs de [91, 92] pour les SFR sont liés au fait qu'à toute exécution de la forme :

$$(q_0, \varepsilon) \xrightarrow{\sigma_1} (q, w) \xrightarrow{!\mathtt{e}} (q, w\mathtt{e}) \xrightarrow{\sigma_2} (q', w'_1\mathtt{e}w'_2) \tag{3.1}$$

correspond une exécution :

$$(q_0, \varepsilon) \xrightarrow{\sigma_1} (q, w) \xrightarrow{\sigma_2} (q', w'_1w'_2) \tag{3.2}$$

où $\mathtt{e}$ n'est pas mémorisé[1]. Cependant, lorsque l'environnement est explicitement spécifié, le choix d'exécuter ou non la transition de mémorisation de $\mathtt{e}$ n'est plus du ressort du système à file réactif. En effet, lorsque l'environnement propose une occurrence de $\mathtt{e}$, cette transition peut être exécutée[2] alors que dans le cas contraire, elle n'est pas franchissable.

Ainsi, nous montrons en section 3.1 que lorsque le système dispose d'au moins deux événements mémorisables, il est capable de simuler une machine à compteurs [80], et donc une machine de Turing. Puis nous considérons, en section 3.2, les systèmes qui ne disposent que d'un événement mémorisable. Dans ce cas, cet événement peut être représenté par un compteur non borné avec test à zéro, et la puissance du modèle est donc moindre : les machines à un compteur sont des automates à pile dont l'ensemble des configurations accessibles est effectivement reconnaissable [17, 42]. Finalement, nous considérons en section 3.3 la classe des environnements périodiques (courants dans le domaine temps-réel) pour lesquels nous cherchons à savoir

[1] Ceci vient du fait que les transitions de mémorisations forment des boucles sur les états de l'AFR.

[2] Elle *doit* être exécutée si $\mathtt{e}$ est le seul événement proposé.

si l'exécution du système à file réactif embarqué est périodique, ce que nous considérons comme un critère de bon fonctionnement de l'application.

3.1 SFR Embarqué avec 2 événements mémorisables

En section 3.1.2, nous montrons comment construire un AFR R (qui génère le SFR S) et un environnement périodique E tels que $S||E$ simule une machine à compteurs. Comme il suffit de deux compteurs pour simuler une machine de Turing, nous en déduisons que tout SFR Embarqué possédant au moins deux événements mémorisables et un environnement périodique peut simuler une machine de Turing. Au préalable, nous rappelons les résultats qui nous importent sur les machines à compteurs en section 3.1.1.

3.1.1 Machines à compteurs

Définition 3.1 (Machine à compteurs [80])
Une *machine à n compteurs (MC(n))* [80], $M = (Q, C, I)$, est constituée :

1. d'un ensemble fini d'états Q où l'on distingue l'état initial q_0 et l'état final q_F,
2. d'un ensemble de n compteurs $C = \{c_1, \ldots, c_n\}$ à valeurs dans $\mathbb{N}$,
3. d'un ensemble fini d'instructions $I \subseteq Q \times C \times \{{+}{+}, {-}{-}, {=}0?\} \times Q$ vérifiant $I_{q_F} = \emptyset$ et $\forall q \in Q \setminus \{q_F\},\ I_q \neq \emptyset$ où :
$$I_q = \{i \in I \mid i = q \xrightarrow{c\ \mathrm{op}} q'\} \tag{3.3}$$
est l'ensemble des instructions définies en q.

♦

Une machine à compteurs est *déterministe* s'il existe exactement en tout état $q \neq q_F$:
– soit une instruction d'incrémentation : $\exists q' \in Q, c_k \in C$ t.q. $I_q = \{q \xrightarrow{c_k{+}{+}} q'\}$,
– soit une instruction de décrémentation et une instruction de test à zéro appliquées au même compteur : $\exists q', q'' \in Q, c_k \in C$ t.q. $I_q = \{q \xrightarrow{c_k=0?} q', q \xrightarrow{c_k{-}{-}} q''\}$.

Remarque 3.2
Les machines à compteurs définies par Minsky [80] sont déterministes. Dans la suite, sauf mention explicite, les machines à compteurs considérées sont déterministes. ♦

Sémantique

La sémantique d'une machine à n compteurs M est définie par le système de transitions dont les états sont les configurations de M : les $n+1$-uplets $\langle q, m_1, \ldots, m_n\rangle$ constitués de l'état courant $q \in Q$ de la machine, et des valeurs respectives $m_1, \ldots, m_n \in \mathbb{N}$ de chacun des compteurs $c_1, \ldots, c_n$, et dont la relation de transition est définie par :

1. $\langle q, m_1, \ldots, m_k, \ldots, m_n\rangle \rightarrow \langle q', m_1, \ldots, m_k + 1, \ldots, m_n\rangle$ si $q \xrightarrow{c_k{+}{+}} q' \in I$ (incrémentation du $k^{i\grave{e}me}$ compteur),

2. $\langle q, m_1, \ldots, m_k, \ldots, m_n \rangle \rightarrow \langle q', m_1, \ldots, m_k, \ldots, m_n \rangle$ si $q \xrightarrow{c_k=0?} q' \in I$ et $m_k = 0$ (test à zéro de c_k),
3. $\langle q, m_1, \ldots, m_k, \ldots, m_n \rangle \rightarrow \langle q'', m_1, \ldots, m_k - 1, \ldots, m_n \rangle$ si $q \xrightarrow{c_k--} q' \in I$ et $m_k > 0$ (décrémentation du $k^{ième}$ compteur).

La configuration *initiale* est le $n+1$-uplet $\langle q_0, 0, \ldots, 0 \rangle$; une configuration $\langle q, m_1, \ldots, m_n \rangle$ est *finale* si $q = q_F$.

Remarque 3.3
Nous notons les instructions des machines à compteurs sous la forme de transitions par souci de concision. De même, afin de faciliter la lecture de ce document, les machines à compteurs sont par la suite représentées sous la forme d'un système de transitions. Bien entendu, les notations de [80] trouvent leur équivalent :

(q : c_k++; goto q')	$q \xrightarrow{c_k++} q'$
(q : if c_k=0? then goto q'	$q \xrightarrow{c_k=0?} q'$
else c_k−−; goto q'')	$q \xrightarrow{c_k--} q''$

♦

Une *exécution* d'une machine à compteurs est une séquence finie ou infinie de configurations ayant pour origine la configuration initiale :

$$\langle q_0, 0, \ldots, 0 \rangle \rightarrow \langle q, m_1, \ldots, m_n \rangle \rightarrow \ \ldots \rightarrow \langle q', m'_1, \ldots, m'_n \rangle \rightarrow \ \ldots$$

L'ensemble des exécutions d'une machine à compteurs M est notée $[\![M]\!]$.

Une configuration $\langle q, m_1, \ldots, m_n \rangle$ est *accessible* s'il existe une exécution de M qui la visite :

$$\exists \sigma \in [\![M]\!], \text{ t.q. } \sigma = \langle q_0, 0, \ldots, 0 \rangle \rightarrow \ \ldots \rightarrow \langle q, m_1, \ldots, m_n \rangle \ldots$$

Nous notons $\langle q, m_1, \ldots, m_n \rangle \in \sigma$ si $\langle q, m_1, \ldots, m_n \rangle$ est accessible sur l'exécution $\sigma \in [\![M]\!]$.

Propriétés

Minsky [80] a prouvé qu'il est possible de simuler une machine de Turing par une machine à 5 compteurs déterministe, et que toute machine à n compteurs déterministe peut être simulée par une machine à deux compteurs déterministe. Par conséquent :

Théorème 3.4 (Minsky [80])
Pour toute machine de Turing T, il existe une machine à 2 compteurs déterministe qui simule T. ♦

3.1.2 Capacité calculatoire des systèmes à file réactifs embarqués

Revenons à nos SFR Embarqués. L'environnement d'un automate à file réactif le contraint à exécuter certaines transitions et l'empêche d'en franchir d'autres. En particulier, il est désormais possible de choisir lorsqu'une transition de mémorisation est franchie, et le nombre de fois qu'elle est franchie. En nous appuyant sur cette capacité, nous montrons que les automates à file réactifs embarqués ont la possibilité de compter le nombre d'occurrences mémorisées de chaque événement, et même de tester la nullité de ce nombre. Ainsi nous construisons un système à file réactif dont l'environnement est périodique, qui simule une machine à compteurs déterministe donnée. Notre méthode est expliquée dans la première section et la preuve est apportée dans la seconde section.

Simulation des MC(2) déterministes par les SFR Embarqués

Considérons une machine à 2 compteurs déterministe $M = (Q, C, I)$ avec $C = \{c_1, c_2\}$ à laquelle nous associons un système à file réactif embarqué $S||E$ muni de deux événements mémorisables, $\mathtt{e}_1$ et $\mathtt{e}_2$, et d'un environnement périodique. Dans une configuration du système, les nombres d'occurrences mémorisées de $\mathtt{e}_1$ et de $\mathtt{e}_2$ modélisent respectivement les valeurs courantes de c_1 et c_2.

Afin de transcrire les instructions de M dans $S||E$:

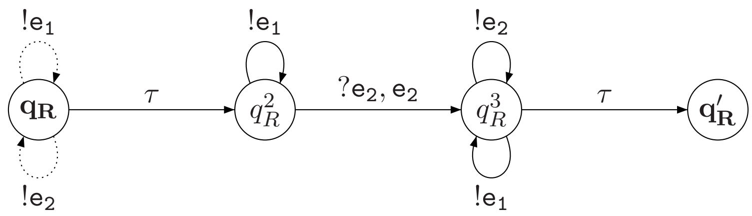
- nous définissons une traduction en une structure d'AFR pour chaque type d'instruction d'une machine à compteurs déterministe : l'instruction d'incrémentation ($q \xrightarrow{c_k++} q'$) et celle de test à zéro et décrémentation ($q \xrightarrow{c_k=0?} q'$ et $q \xrightarrow{c_k--} q''$).
- L'environnement périodique du système est défini sous la forme d'un automate cyclique, d'état initial q_E^0, de telle sorte que lors de la simulation de l'une des instructions de M, partant de q_E^0, il se retrouve en q_E^0.

Pour clarifier, nous supposons que les instructions de M sont adressées sur le compteur c_1. La figure 3.1 représente les structures d'AFR qui simulent les deux types d'instruction d'une machine à compteurs déterministe, et l'environnement périodique utilisé pour la simulation est décrit en figure 3.2. Nous détaillons maintenant le fonctionnement de ces structures. Les transitions dessinées en pointillés sont nécessaires pour satisfaire la condition (4) de la définition 2.5, page 42, mais elles ne sont d'aucune utilité pour la simulation.

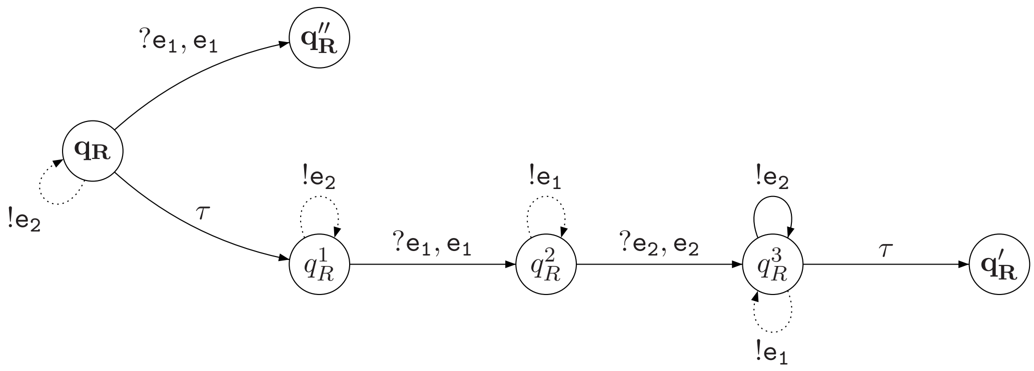
Afin de tester si la valeur courante de c_1 est nulle ($q \xrightarrow{c_1=0?} q'$), il suffit de vérifier que dans la configuration actuelle de $S||E$, $\langle q_R, w, q_E^0 \rangle$, il n'y a pas d'occurrence de $\mathtt{e}_1$ mémorisée. La structure en figure 3.1(b) débute donc en q_R par tenter de franchir la transition $q_R \xrightarrow{?\mathtt{e}_1}_R q_R''$.

- S'il y a une occurrence mémorisée de $\mathtt{e}_1$, cette transition est franchie, une occurrence de $\mathtt{e}_1$ est consommée et le système atteint l'état q_R''. Ceci correspond donc à l'exécution l'instruction $q \xrightarrow{c_1--} q''$: la décrémentation de c_1.
- Lorsque la valeur de c_1 est nulle, il n'y aucune occurrence mémorisée de $\mathtt{e}_1$, donc cette transition n'est pas franchissable. Il faut cependant empêcher $S||E$ d'exécuter la transition duale $q_R \xrightarrow{\mathtt{e}_1}_R q_R''$ car le système doit atteindre q_R' (et non pas q_R'') quand c_1 est nul en q. Pour cela, l'environnement ne doit pas proposer d'occurrence de $\mathtt{e}_1$, ni d'ailleurs d'occurrence de $\mathtt{e}_2$ puisque dans le cas symétrique où $\mathtt{e}_2$ est considéré en lieu et place de $\mathtt{e}_1$, le même problème se pose. Nous introduisons donc un événement *fugace*, nommé τ, que l'environnement propose en premier lieu : lors de la transition $q_E^0 \xrightarrow{\tau}_E q_E^1$ (voir figure 3.2). Ainsi $S||E$ est contraint de franchir la transition $q_R \xrightarrow{\tau}_R q_R^1$ et il atteint la configuration $\langle q_R^1, w, q_E^1 \rangle$.

Laissons pour le moment la traduction de l'opération de test à zéro/décrémentation, et



(a) Simulation de l'instruction : $q \xrightarrow{c_1{+}{+}} q'$.



(b) Simulation des instructions : $q \xrightarrow{c_1=0?} q'$ et $q \xrightarrow{c_1--} q''$.

FIG. 3.1 – Structures d'AFR pour la simulation d'une machine à 2 compteurs.

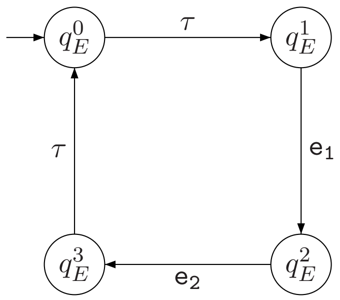


FIG. 3.2 – Environnement périodique pour la simulation d'une machine à 2 compteurs.

passons à la structure correspondant à l'instruction d'incrémentation en figure 3.1(a). Depuis $\langle q_R^0, w, q_E^0\rangle$, la toute première transition $q_R \xrightarrow{\tau}_R q_R^2$ est nécessaire pour consommer l'événement τ proposé par l'environnement. Elle amène le système en $\langle q_R^2, w, q_E^1\rangle$. L'incrémentation de c_1 est naturellement modélisée par la mémorisation d'une occurrence de $\mathtt{e_1}$. Il faut donc que l'environnement émette, cette fois-ci, une occurrence de cet événement. Cependant, lorsque l'opération symétrique, l'incrémentation de c_2, est considérée, c'est d'une occurrence de $\mathtt{e_2}$ dont le système a besoin. Il faudrait donc que E propose $\mathtt{e_1}$ ou $\mathtt{e_2}$, ce qui n'est pas possible puisque l'environnement est périodique. La solution consiste alors à proposer ces événements en séquence : $\mathtt{e_1}$ sur la transition $q_E^1 \xrightarrow{\mathtt{e_1}}_E q_E^2$ puis $\mathtt{e_2}$ sur la transition suivante $q_E^2 \xrightarrow{\mathtt{e_2}}_E q_E^3$ (figure 3.2).

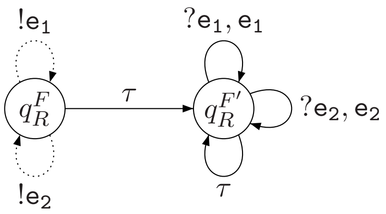
Il reste à traiter ces occurrences correctement dans la structure d'incrémentation. Il faut qu'à l'arrivée en q'_R le nombre d'occurrences mémorisées de $\mathtt{e_1}$ ait augmenté d'une unité alors que pour $\mathtt{e_2}$, ce nombre doit rester globalement stable. L'occurrence de $\mathtt{e_1}$ est donc mémorisée en franchissant l'une des deux transitions : $q_R^2 \xrightarrow{!\mathtt{e_1}}_R q_R^2$ ou $q_R^3 \xrightarrow{!\mathtt{e_1}}_R q_R^3$. Ce choix est résolu par la présence ou l'absence d'une occurrence mémorisée de $\mathtt{e_2}$ lorsque $S||E$ atteint la configuration $\langle q_R^2, w, q_E^1\rangle$. En effet, pour traiter l'occurrence de $\mathtt{e_2}$ proposée par l'environnement, la structure dispose de la transition $q_R^2 \xrightarrow{\mathtt{e_2}}_R q_R^3$ (voir figure 3.1(a)). Or cette transition est couplée à la transition de traitement différé de $\mathtt{e_2}$: $q_R^2 \xrightarrow{?\mathtt{e_2}}_R q_R^3$ conformément au point (4b) de la définition 2.5.

- Par conséquent, lorsque la file w contient une occurrence de $\mathtt{e_2}$ en $\langle q_R^2, w, q_E^1\rangle$, $S||E$ exécute les transitions suivantes : $q_R^2 \xrightarrow{?\mathtt{e_2}}_R q_R^3$ puis $q_R^3 \xrightarrow{!\mathtt{e_1}}_R q_R^3$ et $q_R^3 \xrightarrow{!\mathtt{e_2}}_R q_R^3$ atteignant $\langle q_R^3, \mathtt{e_2}^{\ominus 1} w\mathtt{e_1}\mathtt{e_2}, q_E^3\rangle$.
- Dans le cas contraire, le système franchit les transitions : $q_R^2 \xrightarrow{!\mathtt{e_1}}_R q_R^2$ et $q_R^2 \xrightarrow{\mathtt{e_2}}_R q_R^3$ et atteint la configuration $\langle q_R^3, w\mathtt{e_1}, q_E^3\rangle$.

Dans ces deux cas, la file contient désormais une occurrence supplémentaire de $\mathtt{e_1}$ alors que le nombre d'occurrences mémorisées de $\mathtt{e_2}$ est resté globalement stable.

Revenons maintenant à la structure de la figure 3.1(b) et plus particulièrement, à son état q_R^1. Depuis $\langle q_R^1, w, q_E^1\rangle$, nous devons encore consommer les événements $\mathtt{e_1}$ et $\mathtt{e_2}$ proposés par E pour ramener celui-ci dans son état initial, en vue de l'exécution d'une éventuelle structure suivante. Le traitement de l'occurrence de $\mathtt{e_1}$ peut simplement s'effectuer par une transition de traitement immédiat $q_R^1 \xrightarrow{\mathtt{e_1}}_R q_R^2$ (voir figure 3.1(b)) puisque lorsque q_R^1 est atteint, nous avons l'assurance que la file ne contient pas d'occurrence de $\mathtt{e_1}$. Pour $\mathtt{e_2}$, le traitement est légèrement plus compliqué puisque nous ne disposons d'aucune information sur la présence ou l'absence d'occurrences mémorisées. Nous utilisons donc la même stratégie que précédemment avec les transitions : $q_R^2 \xrightarrow{?\mathtt{e_2},\mathtt{e_2}}_R q_R^3$ et $q_R^3 \xrightarrow{!\mathtt{e_2}}_R q_R^3$ (voir figure 3.1(b)) qui consomment l'occurrence de $\mathtt{e_2}$ tout en laissant le nombre d'occurrences mémorisées globalement invariant. Donc $S||E$ atteint la configuration $\langle q_R^3, w, q_E^3\rangle$ si $(\Psi w)_2 = 0$ et $\langle q_R^3, \mathtt{e_2}^{\ominus 1} w\mathtt{e_2}, q_E^3\rangle$ sinon.

Finalement, nous ajoutons à chacune des structures une dernière transition : $q_R^3 \xrightarrow{\tau}_R q'_R$ pour permettre leur assemblage sans aucun conflit (comme, par exemple, entre les états q_R, figure 3.1(b) et q_R^3, figure 3.1(a) où les traitements de $\mathtt{e_1}$ sont antagonistes). Bien entendu, nous ajoutons une transition d'émission de τ à l'environnement : $q_E^3 \xrightarrow{\tau}_E q_E^0$, et celui-ci retrouve bien son état initial à la fin de l'exécution de chacune des structures. De même, nous ajoutons les transitions nécessaires pour que l'AFR obtenu soit bien complet vis-à-vis des événements mémorisables (point (4) de la définition 2.5, page 42) : les transitions représentées en pointillés en figures 3.1(a) et 3.1(b). Finalement, nous terminons notre construction par l'ajout de la structure de terminaison en q_R^F :



où q_R^F est l'état de l'AFR correspondant à q_F, l'état final de M. Nous choisissons d'introduire un nouvel état $q_R^{F'}$ plutôt que de définir ces transitions sur q_R^F afin de simplifier la preuve du théorème 3.6 (page 76).

Preuve de la simulation

Soit M une machine à 2 compteurs, c_1 et c_2, déterministe. Nous notons $\overline{k}$ le nombre dual de k dans $\{1, 2\}$: $\overline{k} = 3 - k$. À partir de M, et en suivant la construction décrite par les structures d'AFR de la figure 3.1 (page 73), nous définissons un AFR à 2 compteurs.

Définition 3.5 (AFR à 2 compteurs)
L'*automate à file réactif à 2 compteurs (AFR à 2 compteurs)* $R = (Q_R, q_R^0, A_R, \rightarrow_R)$ associé à une machine à 2 compteurs déterministe est défini par :

1. $Q_R = \bigcup_{q \xrightarrow{c_k++} q' \in I} \left\{ q_R, q_R^{\overline{k}}, q_R^3, q_R' \right\} \cup \bigcup_{\left\{ q \xrightarrow{c_k=0?} q',\ q \xrightarrow{c_k--} q'' \right\} \subseteq I} \left\{ q_R, q_R^1, q_R^2, q_R^3, q_R', q_R'' \right\} \cup \{q_R^{F'}\}$.
2. q_R^0 correspond à l'état initial q_0 de M.
3. $A_R = (\{!, ?\} \times \{\mathsf{e}_1, \mathsf{e}_2\}) \cup \{\mathsf{e}_1, \mathsf{e}_2, \tau\}$, i.e. $\Sigma_M = \{\mathsf{e}_1, \mathsf{e}_2\}$ et $\Sigma_F = \{\tau\}$.
4. La relation de transition $\rightarrow_R$ de R est définie par :

$$\begin{aligned}
&\bigcup_{\left(q \xrightarrow{c_k++} q'\right)} \left\{ \begin{array}{c} q_R \xrightarrow{!\mathsf{e}_1}_R q_R,\ q_R \xrightarrow{!\mathsf{e}_2}_R q_R,\ q_R \xrightarrow{\tau}_R q_R^{\overline{k}},\ q_R^{\overline{k}} \xrightarrow{!\mathsf{e}_\mathsf{k}}_R q_R^{\overline{k}},\ q_R^{\overline{k}} \xrightarrow{?\mathsf{e}_{\overline{\mathsf{k}}},\, \mathsf{e}_{\overline{\mathsf{k}}}}_R q_R^3,\ q_R^3 \xrightarrow{!\mathsf{e}_1}_R q_R^3, \\ q_R^3 \xrightarrow{!\mathsf{e}_2}_R q_R^3,\ q_R^3 \xrightarrow{\tau}_R q_R' \end{array} \right\} \\
\cup &\bigcup_{\left\{ \begin{array}{l} q \xrightarrow{c_k=0?} q', \\ q \xrightarrow{c_k--} q'' \end{array} \right\} \subseteq I} \left\{ \begin{array}{l} q_R \xrightarrow{!\mathsf{e}_{\overline{\mathsf{k}}}}_R q_R,\ q_R \xrightarrow{?\mathsf{e}_\mathsf{k},\, \mathsf{e}_\mathsf{k}}_R q_R'',\ q_R \xrightarrow{\tau}_R q_R^1,\ q_R^1 \xrightarrow{!\mathsf{e}_2}_R q_R^1,\ q_R^1 \xrightarrow{?\mathsf{e}_1,\, \mathsf{e}_1}_R q_R^2, \\ q_R^2 \xrightarrow{!\mathsf{e}_1}_R q_R^2,\ q_R^2 \xrightarrow{?\mathsf{e}_2,\, \mathsf{e}_2}_R q_R^3,\ q_R^3 \xrightarrow{!\mathsf{e}_1}_R q_R^3,\ q_R^3 \xrightarrow{!\mathsf{e}_2}_R q_R^3,\ q_R^3 \xrightarrow{\tau}_R q_R' \end{array} \right\} \\
\cup &\left\{ q_R^F \xrightarrow{!\mathsf{e}_1}_R q_R^F,\ q_R^F \xrightarrow{!\mathsf{e}_2}_R q_R^F,\ q_R^F \xrightarrow{\tau}_R q_R^{F'},\ q_R^{F'} \xrightarrow{?\mathsf{e}_1,\, \mathsf{e}_1}_R q_R^{F'},\ q_R^{F'} \xrightarrow{?\mathsf{e}_2,\, \mathsf{e}_2}_R q_R^{F'},\ q_R^{F'} \xrightarrow{\tau}_R q_R^{F'} \right\}
\end{aligned}$$

♦

Soit $E = (Q_E, q_E^0, A_E, \rightarrow_E)$ le système de transitions correspondant à la figure 3.2 (page 73). Enfin, soient S le SFR qui donne la sémantique de R (conformément à la définition 2.11, page 44), et $S||E$ le SFR Embarqué définit depuis S et E en respect de la définition 2.21 (page 51).

La relation d'équivalence $\sim_\Psi$ sur les configurations de M et $S||E$ est définie par :

$$\langle q, m_1, m_2 \rangle \sim_\Psi \langle q_R, w, q_E \rangle \Leftrightarrow q = q_R \text{ et } \Psi(w) = \begin{pmatrix} m_1 & m_2 \end{pmatrix} \text{ et } q_E = q_E^0$$

Nous définissons la simulation d'une machine à compteurs déterministe M par un SFR Embarqué $S||E$, par :

- Pour toutes configurations $\langle q, m_1, m_2\rangle, \langle q', m'_1, m'_2\rangle \in RS(M)$ et $\langle q_R, w, q_E\rangle \in RS(S||E)$ telles que :
  - $\langle q, m_1, m_2\rangle \sim_\Psi \langle q_R, w, q_E\rangle$,
  - et $\langle q, m_1, m_2\rangle \rightarrow \langle q', m'_1, m'_2\rangle$,

  il existe une configuration $\langle q'_R, w', q'_E\rangle \in RS(S||E)$ telle que :
  - $\langle q_R, w, q_E\rangle \xrightarrow{\sigma}{}^*_{S||E} \langle q'_R, w', q'_E\rangle$ et pour toute configuration $\langle q''_R, w'', q''_E\rangle$ de σ, $q''_R \notin Q$,
  - et $\langle q', m'_1, m'_2\rangle \sim_\Psi \langle q'_R, w', q'_E\rangle$.
- Et réciproquement, pour toutes configurations $\langle q_R, w, q_E\rangle, \langle q'_R, w', q'_E\rangle \in RS(S||E)$ et $\langle q, m_1, m_2\rangle \in RS(M)$ telles que :
  - $\langle q, m_1, m_2\rangle \sim_\Psi \langle q_R, w, q_E\rangle$,
  - et $\langle q_R, w, q_E\rangle \xrightarrow{\sigma}{}^*_{S||E} \langle q'_R, w', q'_E\rangle$ et pour toute configuration $\langle q''_R, w'', q''_E\rangle$ de σ, $q''_R \notin Q$,

  il existe une configuration $\langle q', m'_1, m'_2\rangle \in RS(M)$ telle que :
  - $\langle q, m_1, m_2\rangle \rightarrow \langle q', m'_1, m'_2\rangle$,
  - et $\langle q', m'_1, m'_2\rangle \sim_\Psi \langle q'_R, w', q'_E\rangle$.

Théorème 3.6

Pour toute machine à 2 compteurs déterministe M, il existe un SFR Embarqué $S||E$ avec 2 événements mémorisables et un environnement périodique qui simule M. ♦

Preuve. Considérons une machine à 2 compteurs déterministe M, et l'AFR à 2 compteurs R obtenu par la définition 3.5 (page 75). Soient S le SFR qui donne la sémantique de R, E le système de transitions étiqueté de la figure 3.2 (page 73), et $S||E$ le SFR Embarqué obtenu en plongeant S dans E, conformément à la définition 2.21 (page 51). Nous prouvons la simulation de M par $S||E$ par induction.

Les deux configurations initiales de M et $S||E$: $\langle q_0, 0, 0\rangle$ et $\langle q^0_R, \varepsilon, q^0_E\rangle$ sont accessibles et $\sim_\Psi$-équivalentes.

- Soient $\langle q, m_1, m_2\rangle, \langle q', m'_1, m'_2\rangle \in RS(M)$ et $\langle q_R, w, q_E\rangle \in RS(S||E)$ trois configurations telles que :
  - $\langle q, m_1, m_2\rangle \sim_\Psi \langle q_R, w, q_E\rangle$,
  - et $\langle q, m_1, m_2\rangle \rightarrow \langle q', m'_1, m'_2\rangle$,

  Nous ne considérons que les instructions définies sur le compteur c_1 de M, celles définies sur c_2 sont symétriques. Depuis cette configuration, M exécute l'instruction :
  - $\mathbf{q} \xrightarrow{c_1++} \mathbf{q'}$. Alors, M atteint la configuration $\langle q', m_1+1, m_2\rangle$. Par la définition 3.5, $S||E$ franchit l'une des séquences de transitions suivantes :
    - **Si $|\mathbf{w}|_{\mathsf{e}_2} > \mathbf{0}$.** Alors, :

$$\langle q_R, w, q^0_E\rangle \xrightarrow{\tau} \langle q^2_R, w, q^1_E\rangle \xrightarrow{?\mathsf{e}_2} \langle q^3_R, \mathsf{e}_2{}^{\ominus 1}w, q^1_E\rangle \xrightarrow{!\mathsf{e}_1} \langle q^3_R, (\mathsf{e}_2{}^{\ominus 1}w).\mathsf{e}_1, q^2_E\rangle \xrightarrow{!\mathsf{e}_2} \langle q^3_R, (\mathsf{e}_2{}^{\ominus 1}w).\mathsf{e}_1.\mathsf{e}_2, q^3_E\rangle \xrightarrow{\tau} \langle q'_R, (\mathsf{e}_2{}^{\ominus 1}w).\mathsf{e}_1.\mathsf{e}_2, q^0_E\rangle$$

puisque $\Psi((\mathsf{e}_2{}^{\ominus 1}w).\mathsf{e}_1.\mathsf{e}_2) = \begin{pmatrix}|w|_{\mathsf{e}_1}+1 & |w|_{\mathsf{e}_2}\end{pmatrix}$ nous en déduisons l'équivalence des deux configurations atteintes par M et $S||E$: $\langle q', m_1+1, m_2\rangle$ et $\langle q'_R, (\mathsf{e}_2{}^{\ominus 1}w).\mathsf{e}_1.\mathsf{e}_2, q^0_E\rangle$

respectivement.

– **Sinon $|\mathbf{w}|_{\mathsf{e}_2} = 0$.** Par conséquent $S||E$ exécute la séquence :

$$\langle q_R, w, q_E^0\rangle \xrightarrow{\tau} \langle q_R^2, w, q_E^1\rangle \xrightarrow{!\mathsf{e}_1} \langle q_R^2, w.\mathsf{e}_1, q_E^2\rangle \xrightarrow{\mathsf{e}_2} \langle q_R^3, w.\mathsf{e}_1, q_E^3\rangle \xrightarrow{\tau} \langle q'_R, w.\mathsf{e}_1, q_E^0\rangle$$

et il est clair que $\langle q, m_1+1, m_2\rangle \sim_\Psi \langle q'_R, w.\mathsf{e}_1, q_E^0\rangle$.

– $\mathbf{q} \xrightarrow{c_1--} \mathbf{q''}$. Dans ce cas, $m_1 > 0$ et M atteint la configuration $\langle q'', m_1-1, m_2\rangle$. De son côté, puisque $|w|_{\mathsf{e}_1} > 0$, $S||E$ exécute la transition :

$$\langle q_R, w, q_E^0\rangle \xrightarrow{?\mathsf{e}_1} \langle q''_R, {\mathsf{e}_1}^{\ominus 1}w, q_E^0\rangle$$

et l'on a finalement $\langle q'', m_1-1, m_2\rangle \sim_\Psi \langle q''_R, {\mathsf{e}_1}^{\ominus 1}w, q_E^0\rangle$.

– $\mathbf{q} \xrightarrow{c_1=0?} \mathbf{q'}$. Donc, $m_1 = 0$, ce qui entraine M dans la configuration : $\langle q', m_1, m_2\rangle$. Puisque $|w|_{\mathsf{e}_1} = 0$, suivant $|w|_{\mathsf{e}_2}$, $S||E$ franchit une des séquences suivantes :

– **Soit $|\mathbf{w}|_{\mathsf{e}_2} > 0$.** Alors :

$$\langle q_R, w, q_E^0\rangle \xrightarrow{\tau} \langle q_R^1, w, q_E^1\rangle \xrightarrow{\mathsf{e}_1} \langle q_R^2, w, q_E^2\rangle \xrightarrow{?\mathsf{e}_2} \langle q_R^3, {\mathsf{e}_2}^{\ominus 1}w, q_E^2\rangle \xrightarrow{!\mathsf{e}_2}$$
$$\langle q_R^3, ({\mathsf{e}_2}^{\ominus 1}w).\mathsf{e}_2, q_E^3\rangle \xrightarrow{\tau} \langle q'_R, ({\mathsf{e}_2}^{\ominus 1}w).\mathsf{e}_2, q_E^0\rangle$$

Puisque $|({\mathsf{e}_2}^{\ominus 1}w).\mathsf{e}_2|_{\mathsf{e}_2} = |w|_{\mathsf{e}_2}$, les configurations $\langle q', m_1, m_2\rangle$ et $\langle q'_R, ({\mathsf{e}_2}^{\ominus 1}w).\mathsf{e}_2, q_E^0\rangle$ atteintes par M et $S||E$ respectivement, sont $\sim_\Psi$-équivalentes.

– **Sinon, $|\mathbf{w}|_{\mathsf{e}_2} = 0$.** Par conséquent, la séquence exécutée par $S||E$ est :

$$\langle q_R, w, q_E^0\rangle \xrightarrow{\tau} \langle q_R^1, w, q_E^1\rangle \xrightarrow{\mathsf{e}_1} \langle q_R^2, w, q_E^2\rangle \xrightarrow{\mathsf{e}_2} \langle q_R^3, w, q_E^3\rangle \xrightarrow{\tau} \langle q'_R, w, q_E^0\rangle$$

et $\langle q', m_1, m_2\rangle \sim_\Psi \langle q'_R, w, q_E^0\rangle$.

– Réciproquement, soient trois configurations $\langle q_R, w, q_E\rangle, \langle q'_R, w', q'_E\rangle \in RS(S||E)$ et $\langle q, m_1, m_2\rangle \in RS(M)$ telles que :

– $\langle q, m_1, m_2\rangle \sim_\Psi \langle q_R, w, q_E\rangle$,

– et $\langle q_R, w, q_E\rangle \xrightarrow{\sigma}{}^*_{S||E} \langle q'_R, w', q'_E\rangle$ et pour toute configuration $\langle q''_R, w'', q''_E\rangle$ de σ, $q''_R \notin Q$,

Nous considérons uniquement les séquences issues de $\langle q_R, w, q_E\rangle$ qui correspondent à des opérations sur c_1 (donc sur e_1). Celles concernant c_2 se déduisent par symétrie. Notons que nous avons $q_R \in Q$ et $q_E = q_E^0$ par hypothèse.

– **Soit la séquence σ est :**

$$\langle q_R, w, q_E^0\rangle \xrightarrow{\tau}_{S||E} \langle q_R^2, w, q_E^1\rangle \xrightarrow{!\mathsf{e}_1}_{S||E} \langle q_R^2, w.\mathsf{e}_1, q_E^2\rangle \xrightarrow{\mathsf{e}_2}_{S||E}$$
$$\langle q_R^3, w.\mathsf{e}_1, q_E^3\rangle \xrightarrow{\tau}_{S||E} \langle q'_R, w', q_E^0\rangle$$

avec $w' = w.\mathsf{e}_1$, ou encore :

$$\langle q_R, w, q_E^0\rangle \xrightarrow{\tau}_{S||E} \langle q_R^2, w, q_E^1\rangle \xrightarrow{?\mathsf{e}_2} \langle q_R^3, {\mathsf{e}_2}^{\ominus 1}w, q_E^1\rangle \xrightarrow{!\mathsf{e}_1}_{S||E} \langle q_R^3, {\mathsf{e}_2}^{\ominus 1}w.\mathsf{e}_1, q_E^2\rangle$$
$$\xrightarrow{!\mathsf{e}_2}_{S||E} \langle q_R^3, {\mathsf{e}_2}^{\ominus 1}w.\mathsf{e}_1.\mathsf{e}_2, q_E^3\rangle \xrightarrow{\tau}_{S||E} \langle q'_R, w', q_E^0\rangle$$

avec $w' = {\mathsf{e}_2}^{\ominus 1}w.\mathsf{e}_1.\mathsf{e}_2$. Dans ces deux cas, $\Psi(w') = \Big(|w|_{\mathsf{e}_1}+1 \quad |w|_{\mathsf{e}_2}\Big)$. Par construction, il existe dans M une instruction d'incrémentation, et M franchit la transition :

$$\langle q, m_1, m_2 \rangle \xrightarrow{c_1++} \langle q', m_1+1, m_2 \rangle$$

Nous avons $q' = q'_R$ (par construction), et puisque $\Psi(w) = \begin{pmatrix} m_1 & m_2 \end{pmatrix}$ par hypothèse d'induction, nous en déduisons : $\langle q', m_1+1, m_2 \rangle \sim_\Psi \langle q'_R, w', q^0_E \rangle$.

– **Soit la séquence σ est :**

$$\langle q_R, w, q^0_E \rangle \xrightarrow{?\mathsf{e}_1}_{S||E} \langle q''_R, w', q^0_E \rangle$$

avec $w' = \mathsf{e_1}^{\ominus 1} w$. Par construction, il existe une instruction de décrémentation issue de q dans M, et par hypothèse d'induction, $m_1 > 0$. Donc, M exécute l'instruction :

$$\langle q, m_1, m_2 \rangle \xrightarrow{c_1--} \langle q'', m_1-1, m_2 \rangle$$

et nous avons : $\langle q'', m_1-1, m_2 \rangle \sim_\Psi \langle q''_R, w', q^0_E \rangle$

– **Sinon, σ est l'une des séquences :**

$$\langle q_R, w, q^0_E \rangle \xrightarrow{\tau}_{S||E} \langle q^1_R, w, q^1_E \rangle \xrightarrow{\mathsf{e}_1}_{S||E} \langle q^2_R, w, q^2_E \rangle \xrightarrow{\mathsf{e}_2}_{S||E} \langle q^3_R, w, q^3_E \rangle \xrightarrow{\tau}_{S||E} \langle q'_R, w', q^0_E \rangle$$

avec $w' = w$, ou :

$$\langle q_R, w, q^0_E \rangle \xrightarrow{\tau}_{S||E} \langle q^1_R, w, q^1_E \rangle \xrightarrow{\mathsf{e}_1}_{S||E} \langle q^2_R, w, q^2_E \rangle \xrightarrow{?\mathsf{e}_2}_{S||E} \langle q^3_R, \mathsf{e_2}^{\ominus 1} w, q^2_E \rangle \xrightarrow{!\mathsf{e}_2}_{S||E}$$
$$\langle q^3_R, \mathsf{e_2}^{\ominus 1} w.\mathsf{e}_2, q^3_E \rangle \xrightarrow{\tau}_{S||E} \langle q'_R, w', q^0_E \rangle$$

avec $w' = \mathsf{e_2}^{\ominus 1} w.\mathsf{e}_2$. Dans les deux cas, $\Psi(w') = \Psi(w)$. Par construction, il existe dans M une instruction de test à zéro en q, et $m_1 = 0$ par hypothèse d'induction. Alors, M exécute l'instruction :

$$\langle q, m_1, m_2 \rangle \xrightarrow{c_1=0?} \langle q', m_1, m_2 \rangle$$

et : $\langle q', m_1, m_2 \rangle \sim_\Psi \langle q'_R, w', q^0_E \rangle$.

♦

3.2 SFR Embarqué avec 1 événement mémorisable

Lorsqu'un système à file réactif ne dispose que d'un seul événement mémorisable e, la gestion de la file se résume à compter le nombre d'occurrences mémorisées de e, et à augmenter et diminuer celui-ci au gré des mémorisations et des consommations de e. L'ordre induit par la gestion FIFFO de la file disparaît.

Donc, tout système à file réactif avec 1 événement mémorisable est simulable par une machine disposant d'un compteur :

– chaque mémorisation de e peut être modélisée par l'incrémentation du compteur,
– et chaque traitement différé de e est simulable par la décrémentation de ce compteur.

– Le traitement immédiat d'une occurrence de $\mathtt{e}$ n'est possible que si la file est vide, donc lorsque le compteur vaut zéro.

La classe des machines à 1 compteur est close par synchronisation avec un automate fini (langage rationnel), et elle forme une sous classe des automates à pile. Par conséquent, tout système à file réactif avec un événement mémorisable et un environnement rationnel peut être vu comme un automate à pile.

Rappelons que l'espace des configurations accessibles d'un automate à pile est effectivement reconnaissable, que le model-checking de Ltl et de Ctl est décidable pour ces automates [17, 42]. Nous obtenons donc :

Théorème 3.7

Tous les problèmes d'interêt sont décidables pour les systèmes à file réactifs embarqués avec 1 événement mémorisable et un environnement rationnel. ♦

Preuve. Comme nous l'avons indiqué précédemment, l'espace des configurations accessibles est effectivement reconnaissable (Erp) par [17, 42], ce qui implique que les problèmes d'accessibilité (R), donc de couverture (C), et de bornitude (B) sont aussi décidables. Les problèmes de model-checking de Ltl (Mc-LTL) et de Ctl(Mc-CTL) sont décidables par [17, 42]. ♦

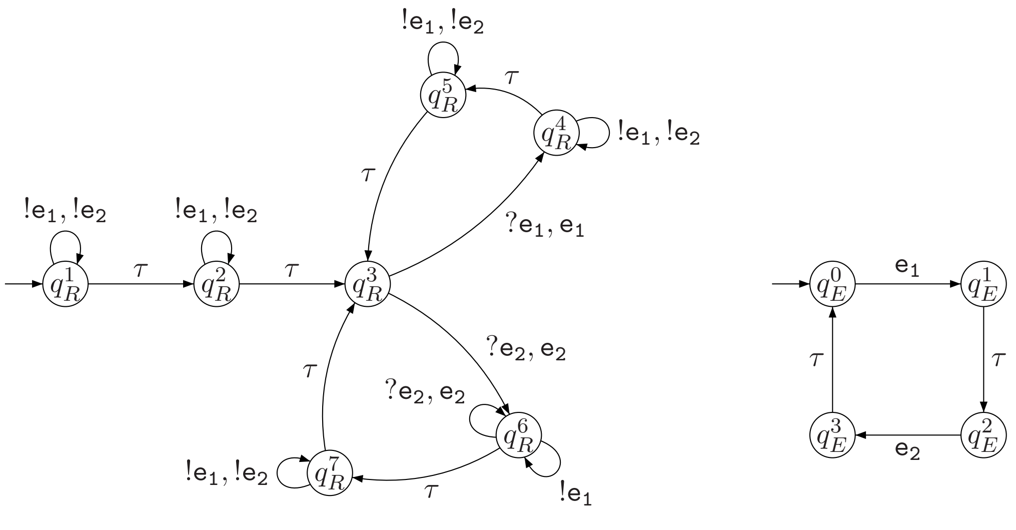
3.3 Comportement des SFR en environnement périodique

Les applications temps-réel sont souvent conçues pour exécuter des tâches périodiques, comme par exemple l'ordonnancement de processus. Ces applications sont donc spécifiées de façon à calquer leur comportement sur le rythme des sollicitations de leur environnement : si les demandes d'exécution de tâches arrivent périodiquement, l'ordonnanceur doit faire en sorte que l'exécution de ces tâches soit elle aussi périodique. Un comportement non-périodique est alors vu comme un problème dans la spécification.

Soit S le SFR qui donne la sémantique de l'AFR R de la figure La figure 3.3(a) (page 80). La figure 3.3, montre un exemple de SFR Embarqué $S||E$, avec un environnement périodique, dont l'exécution est apériodique.

On démontre en effet que depuis toute configuration de la forme $\langle q_R^3, \mathtt{e_1}^n\mathtt{e_2}, q_E^0\rangle$, $S||E$ atteint la configuration $\langle q_R^3, \mathtt{e_1}^{n+1}\mathtt{e_2}, q_E^0\rangle$. Supposons donc que $S||E$ est dans la configuration $\langle q_R^3, \mathtt{e_1}^n\mathtt{e_2}, q_E^0\rangle$, alors, les occurrences mémorisées sont traitées par ordre d'ancienneté. Le cycle sur q_R^3, passant par les états q_R^4 et q_R^5, est donc exécuté n fois, et $S||E$ atteint la configuration $\langle q_R^3, \mathtt{e_2}(\mathtt{e_1}\mathtt{e_2})^n, q_E^0\rangle$. Ensuite, c'est le cycle sur q_R^3, passant par les états q_R^5 et q_R^6 qui est exécuté. La boucle de consommation $?\mathtt{e_2}$ sur q_R^6 consomme toutes les occurrences mémorisées de $\mathtt{e_2}$. Par conséquent, $S||E$ atteint la configuration $\langle q_R^3, \mathtt{e_1}^n\mathtt{e_2}, q_E^0\rangle$. Pour être complet, remarquons que depuis sa configuration initiale $\langle q_R^1, \varepsilon, q_E^0\rangle$, $S||E$ atteint la configuration $\langle q_R^3, \mathtt{e_1}\mathtt{e_2}, q_E^0\rangle$, ce qui amorce le processus. Alors, puisque depuis $\langle q_R^3, \mathtt{e_1}^n\mathtt{e_2}, q_E^0\rangle$ le cycle sur q_R^3 passant par q_R^4 et q_R^5 est exécuté n fois, et la boucle $?\mathtt{e_2}$ sur q_R^6 l'est aussi n fois, nous en déduisons l'apériodicité.

Considérons à nouveau l'AFR R de la figure 3.3(a) (page 80), mais nous plongons maintenant sa sémantique S dans l'environnement représenté en figure 3.4. L'exécution du SFR Embarqué



(a) AFR R.

(b) Environnement E.

FIG. 3.3 – Exemple d'un SFR Embarqué, avec un environnement périodique, dont l'exécution est apériodique.

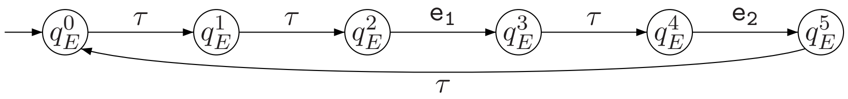


FIG. 3.4 – Environnement périodique dans lequel l'AFR de la figure 3.3(a) se comporte périodiquement.

$S||E$ ainsi obtenu est ultimement périodique, car $S||E$ est borné. En effet, en développant son exécution, on découvre que la configuration $(q_R^3, \mathsf{e}_1\mathsf{e}_2, q_E^1)$ est récurrente, ce qui suffit pour que l'exécution soit ultimement périodique (voir remarque 3.8).

Remarque 3.8

Lorsque le système à file réactif embarqué, en environnement périodique, est borné, son exécution est ultimement périodique. En effet, les configurations du système sont alors en nombre fini, donc sur l'exécution du système, il en existe au moins une, $\langle q_R, w, q_E\rangle$, qui est récurrente :

$$\langle q_R^0, \varepsilon, q_E^0\rangle \xrightarrow{a_0}_{S||E} \cdots \xrightarrow{a_{n-1}}_{S||E} \langle q_R, w, q_E\rangle \xrightarrow{a_n}_{S||E} \cdots \xrightarrow{a_{n+p}}_{S||E} \langle q_R, w, q_E\rangle$$

avec $\forall i,\ a_i \in A_{S||E}$ et $p \geq 0$. Comme $S||E$ est déterministe, à partir de la deuxième occurrence de $\langle q_R, w, q_E\rangle$, le système exécute à nouveau a_n, puis a_{n+1}, et ainsi de suite jusqu'à a_{n+p}. Ceci l'amène à nouveau dans la configuration $\langle q_R, w, q_E\rangle$, où il reproduit le même comportement infiniment. ♦

Remarque 3.9

Une conséquence immédiate de la remarque 3.8 est que tout système dont l'exécution est apériodique est non borné. ♦

Par conséquent, le problème de l'apériodicité de l'exécution d'un SFR Embarqué est fortement lié au problème de la bornitude. Dans cette section, nous examinons donc la périodicité asymptotique de l'exécution des systèmes à file réactifs embarqués dans des environnements périodiques. Cette propriété est définie de la façon suivante :

(Pe) Pour un SFR Embarqué avec un environnement périodique $S||E$, $[\![S||E]\!]$ est-elle ultimement périodique ?

Nous cherchons à nous assurer que les SFR Embarqués à environnement périodique s'adaptent au rythme de leur environnement. Nous considérons alors qu'un système qui s'exécute périodiquement (asymptotiquement) dans un environnement périodique est bien spécifié, alors que dans le cas inverse, la spécification est erronée. Nous montrons maintenant que, dès lors que le système dispose de deux événements mémorisables, il peut se comporter de façon totalement apériodique malgré un environnement périodique. Par contre, avec un seul événement mémorisable, un système à file réactif plongé en environnement périodique se comporte toujours périodiquement (asymptotiquement).

3.3.1 SFR Embarqués avec 2 événements mémorisables

Il est aisé de construire un AFR dont le comportement est asymptotiquement périodique en environnement périodique : il suffit que le système à file réactif embarqué correspondant soit borné (par la remarque 3.8). Mais cette condition n'est absolument pas nécessaire : la figure 3.5 montre l'exemple d'un système à file réactif embarqué non borné dont l'exécution est périodique.

Nous montrons en effet que la séquence d'actions exécutées par ce système à file réactif embarqué est :

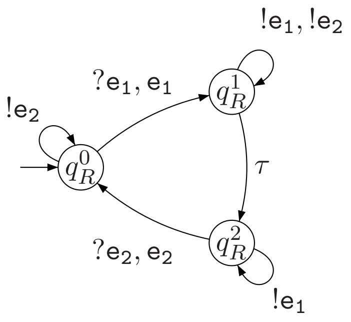
$$\mathtt{e}_1!\mathtt{e}_2!\mathtt{e}_1\tau\,(?\mathtt{e}_2?\mathtt{e}_1!\mathtt{e}_2!\mathtt{e}_1!\mathtt{e}_2!\mathtt{e}_1\tau)^\omega$$

Depuis la configuration initiale $\langle q_R^0, \varepsilon, q_E^0\rangle$, le système exécute la séquence d'actions suivante : $\mathtt{e}_1!\mathtt{e}_2!\mathtt{e}_1\tau$ et arrive dans la configuration $\langle q_R^2, \mathtt{e}_2\mathtt{e}_1, q_E^4\rangle$. Il suffit ensuite de noter que depuis la configuration $\langle q_R^2, (\mathtt{e}_2\mathtt{e}_1)^n, q_E^4\rangle$, $S||E$ exécute les actions $?\mathtt{e}_2?\mathtt{e}_1!\mathtt{e}_2!\mathtt{e}_1!\mathtt{e}_2!\mathtt{e}_1\tau$ pour atteindre la configuration $\langle q_R^2, (\mathtt{e}_2\mathtt{e}_1)^{n+1}, q_E^4\rangle$, ce qui achève la preuve.

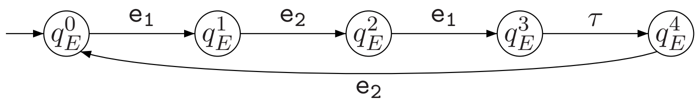
Cependant, il est tout aussi facile de construire un système à file réactif dont l'exécution en environnement périodique est apériodique.

Nous avons montré au début de cette section, un exemple de SFR qui se comporte de façon apériodique ou périodique suivant l'environnement dans lequel il est plongé.

- Cependant, il est légitime de s'interroger sur la possibilité de produire, de façon systématique, des SFR Embarqués dont l'exécution est apériodique.
- L'apériodicité de l'exécution du système représenté en figure 3.3 (page 80) est obtenue grace à la transition de consommation formant une boucle sur l'état q_R^6. Est-il possible qu'un SFR Embarqués sans cycle de consommation, se comporte de façon apériodique en environnement périodique ?



(a) Automate à file réactif R.



(b) Environnement périodique E.

FIG. 3.5 – SFR Embarqué non borné $S||E$ dont l'exécution est périodique.

Un exemple reproductible de comportement apériodique

Nous avons prouvé en section 3.1 que lorsqu'un système à file réactif avec (au moins) deux événements mémorisables est plongé dans un environnement périodique, il est capable de simuler une machine à compteurs. En utilisant ce résultat, nous répondons maintenant aux deux questions soulevées. Il est très simple de construire une machine à compteurs dont l'exécution est apériodique, comme le montre la figure 3.6 (page 83). Cette machine exécute le programme suivant :

tant que *true* **faire**
  $c_2 := c_1;\ c_2{+}{+}$
  $c_1 := c_2;\ c_1{+}{+}$
fin tant que

Il s'agit des instructions $c_2 := c_1$, $c_2{+}{+}$, $c_1 := c_2$ et $c_1{+}{+}$ exécutées en séquence et dans cet ordre, une infinité de fois. L'opération d'affectation $c_i := c_j$ est modélisée à partir des opérations $++$, $--$ et $=0?$ en incrémentant c_i et décrémentant c_j jusquà ce que la valeur de c_j soit nulle. Les valeurs des compteurs n'étant à priori pas bornées, cette opération est modélisée par une boucle qui doit donc être itérée c_j fois. Or, il est clair que les compteurs c_1 et c_2 sont non bornés, et donc l'exécution de cette machine est apériodique.

En utilisant la construction de la définition 3.5 (page 75), nous obtenons le système à file réactif embarqué avec environnement périodique (représenté en figure 3.7) qui simule cette ma-

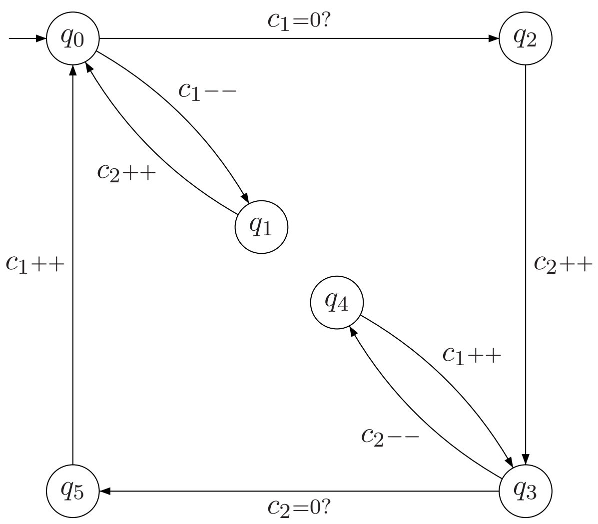
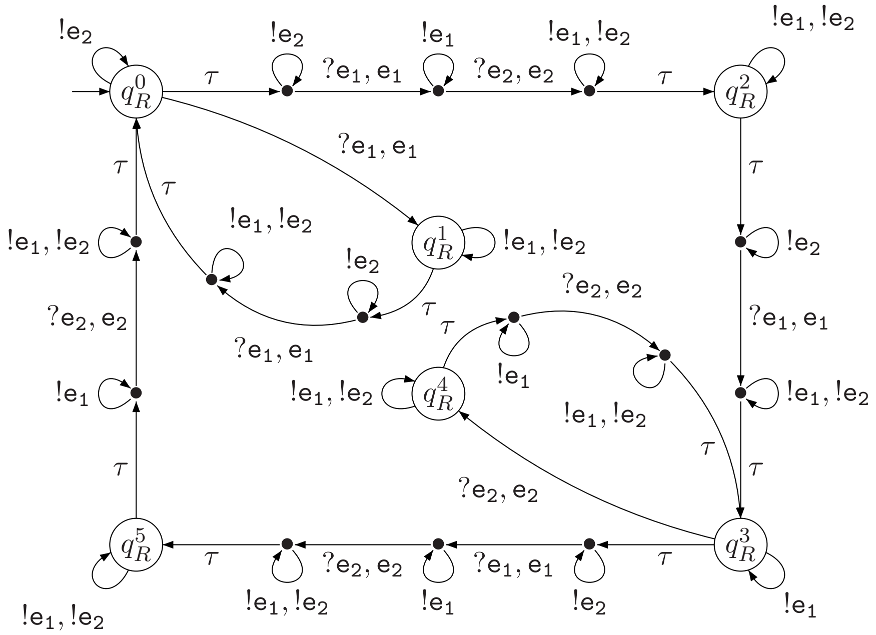
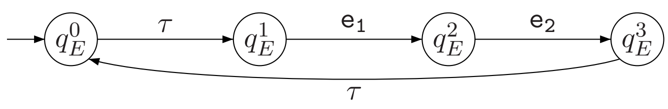


FIG. 3.6 – Machine à 2 compteurs dont l'exécution est apériodique.



(a) Automate à file réactif R.



(b) Environnement périodique E.

FIG. 3.7 – SFR Embarqué $S||E$ dont l'exécution est apériodique.

chine (par le théorème 3.6, page 76). Il reste à prouver que[3] :

$$[\![S||E]\!] = \tau \mathtt{e}_1\mathtt{e}_2\tau\tau\mathtt{e}_1!\mathtt{e}_2\tau \bigodot_{n=1}^{\infty} \Big[(?\mathtt{e}_1\tau?\mathtt{e}_2!\mathtt{e}_1!\mathtt{e}_2\tau)^{2(n-1)} (?\mathtt{e}_2\tau!\mathtt{e}_1\mathtt{e}_2\tau)\, \tau?\mathtt{e}_1!\mathtt{e}_1\mathtt{e}_2\tau\tau!\mathtt{e}_1\mathtt{e}_2\tau$$
$$(?\mathtt{e}_1\tau?\mathtt{e}_1!\mathtt{e}_1!\mathtt{e}_2\tau)^{2n-1} (?\mathtt{e}_1\tau\mathtt{e}_1!\mathtt{e}_2\tau)\, \tau\mathtt{e}_1?\mathtt{e}_2!\mathtt{e}_2\tau\tau\mathtt{e}_1!\mathtt{e}_2\tau \Big]$$

Depuis sa configuration initiale $\langle q_0, \varepsilon, q_E^0 \rangle$, le système exécute la séquence de transitions suivante :

$$\langle q_R^0, \varepsilon, q_E^0 \rangle \xrightarrow{\tau \mathtt{e}_1\mathtt{e}_2\tau}_{S||E} \langle q_R^2, \varepsilon, q_E^0 \rangle \xrightarrow{\tau \mathtt{e}_1!\mathtt{e}_2\tau}_{S||E} \langle q_R^3, \mathtt{e}_2, q_E^0 \rangle$$

À partir de ce point, il suffit de montrer qu'à partir de la configuration $\langle q_R^3, \mathtt{e_2}^n, q_E^0 \rangle$ le système exécute la séquence d'actions :

$$(?\mathtt{e}_1\tau?\mathtt{e}_2!\mathtt{e}_1!\mathtt{e}_2\tau)^{2(n-1)} (?\mathtt{e}_2\tau!\mathtt{e}_1\mathtt{e}_2\tau)\, \tau?\mathtt{e}_1!\mathtt{e}_1\mathtt{e}_2\tau\tau!\mathtt{e}_1\mathtt{e}_2\tau$$
$$(?\mathtt{e}_1\tau?\mathtt{e}_1!\mathtt{e}_1!\mathtt{e}_2\tau)^{2n-1} (?\mathtt{e}_1\tau\mathtt{e}_1!\mathtt{e}_2\tau)\, \tau\mathtt{e}_1?\mathtt{e}_2!\mathtt{e}_2\tau\tau\mathtt{e}_1!\mathtt{e}_2\tau$$

pour atteindre finalement la configuration : $\langle q_R^3, \mathtt{e_2}^{n+2}, q_E^0 \rangle$.

À partir de $\langle q_R^3, \mathtt{e_2}^n, q_E^0 \rangle$, le système exécute $n-1$ fois la séquence d'actions $?\mathtt{e}_2\tau?\mathtt{e}_2!\mathtt{e}_1!\mathtt{e}_2\tau$ jusqu'à ce qu'il n'y ait plus qu'une occurrence mémorisée de $\mathtt{e}_2$, menant à la configuration $\langle q_R^3, \mathtt{e}_2(\mathtt{e}_1)^{n-1}, q_E^0 \rangle$. Puisqu'il n'y a plus qu'une seule occurrence mémorisée de $\mathtt{e}_2$, le système ne peut plus exécuter la deuxième action $?\mathtt{e}_2$ de la séquence d'actions précédente. Elle se mue alors en un traitement immédiat, par conséquent le système exécute cette fois-ci la séquence d'actions $?\mathtt{e}_2\tau!\mathtt{e}_1\mathtt{e}_2\tau$, menant à la configuration $\langle q_R^3, \mathtt{e_1}^n, q_E^0 \rangle$.

Maintenant qu'il n'y a plus d'occurrence mémorisée de $\mathtt{e}_2$, $S||E$ franchit les transitions menant à q_R^5, puis q_R^0. Il exécute donc la séquence d'actions : $\tau?\mathtt{e}_1!\mathtt{e}_1\mathtt{e}_2\tau\tau!\mathtt{e}_1\mathtt{e}_2\tau$ atteignant finalement la configuration $\langle q_R^0, \mathtt{e_1}^{n+1}, q_E^0 \rangle$.

Ensuite, la séquence $?\mathtt{e}_1\tau?\mathtt{e}_1!\mathtt{e}_1!\mathtt{e}_2\tau$ correspondant au transfert de c_1 dans c_2 est itérée n fois, jusqu'à ce que la file ne contienne plus qu'une seule occurrence de $\mathtt{e}_1$. La configuration atteinte par le système est alors $\langle q_R^0, e_1(e_2)^n, q_E^0 \rangle$. De même que pour $\mathtt{e}_2$ précédemment, puisque la file ne contient plus qu'une occurrence de $\mathtt{e}_1$, la deuxième action $?\mathtt{e}_1$ n'est plus exécutable : la consommation de $\mathtt{e}_1$ se transforme en traitement immédiat. La dernière itération de la boucle qui achève le transfert correspond à la séquence $?\mathtt{e}_1\tau\mathtt{e}_1!\mathtt{e}_2\tau$ qui conduit le système dans la configuration $\langle q_R^0, \mathtt{e_2}^{n+1}, q_E^0 \rangle$.

Le retour dans l'état q_R^3 s'opère par le passage via q_R^2, depuis q_R^0. Ceci correspond à la séquence d'actions : $\tau\mathtt{e}_1?\mathtt{e}_2!\mathtt{e}_2\tau\tau\mathtt{e}_1!\mathtt{e}_2\tau$ sur laquelle une occurrence supplémentaire de $\mathtt{e}_2$ est mémorisée, menant alors en $\langle q_R^3, \mathtt{e_2}^{n+2}, q_E^0 \rangle$.

Il est désormais clair :

– qu'il est possible de produire systématiquement des SFR Embarqués au comportement apériodique en environnement périodique,
– qu'il n'est pas nécessaire d'utiliser des cycles de consommation pour parvenir à ce résultat.

[3]Dans cette équation, $\bigodot$ désigne l'opérateur de concaténation.

Indécidabilité de la périodicité

Nous avons montré un exemple de système à file réactif embarqué dans un environnement périodique qui s'exécute, périodiquement, ou apériodiquement suivant son environnement.

Nous considérons la propriété (PE) (définie en section 3.3, page 80) comme un critère de bonne spécification des systèmes à file réactifs embarqués à environnement périodique, en particulier, à cause de sa relation au problème de la bornitude. Il serait donc interessant de pouvoir décider si un système donné respecte cette propriété.

Lemme 3.10

L'itérabilité infinie d'une séquence :

$$\sigma = q_1 \xrightarrow{c_{i_1}\,\mathtt{op}_1} q_2, q_2 \xrightarrow{c_{i_2}\,\mathtt{op}_2} q_3, \ldots, q_n \xrightarrow{c_{i_n}\,\mathtt{op}_n} q_1$$

d'une machine à compteurs déterministe est décidable. ♦

Preuve. Notons tout d'abord qu'il faut que σ contienne au moins autant d'instructions d'incrémentation que de décrémentation pour chacun des compteurs. En effet, si σ ne respecte pas cette contrainte pour un compteur c_k, la valeur de c_k est strictement décroissante lors des itérations successives de σ. Après un nombre fini d'itérations, la valeur de c_k devient nulle, donc plus aucune opération de décrémentation de c_k n'est possible, et par conséquent σ n'est pas itérable.

Une instruction $\mathtt{op}$ d'une machine à compteurs déterministe est franchissable si les conditions suivantes sont respectées :

- $\mathtt{op} = ++$: une instruction d'incrémentation est toujours exécutable,
- $\mathtt{op} = --$: cette instruction est franchissable ssi le compteur concerné est non nul,
- $\mathtt{op} = =\!0?$: l'exécution de cette instruction est soumise à la nullité du compteur considéré.

L'itérabilité de σ est donc conditionnée par l'invariance des conditions d'exécution de $\mathtt{op}_1, \ldots, \mathtt{op}_n$ lors des franchissements successifs de σ. Pour $\mathtt{op} = ++$, l'invariance est triviale puisque l'exécutabilité de cette instruction n'est pas conditionnée, et pour les autres instructions, elle est caractérisée par :

- $\mathtt{op} = --$: la configuration atteinte avant la première exécution de $--$ doit permettre son exécution : le compteur concerné c_k ne doit pas être nul. Puis il faut et il suffit que la valeur de c_k obtenue par le cumul des opérations sur c_k lors du franchissement de σ soit croissante (stable ou strictement croissante). C'est à dire qu'il y a au moins autant d'instructions d'incrémentation de c_k que d'instructions de décrémentation de c_k sur σ,
- $\mathtt{op} = =\!0?$: pour exécuter une première fois cette transition, il faut que la configuration atteinte immédiatement avant le franchissement soit telle que le compteur c_k concerné par cette opération soit nul. Puis, la nullité de ce compteur doit être préservée par les exécutions successives de σ, donc il faut et il suffit que σ contienne exactement autant d'opérations d'incrémentation et de décrémentation de σ.

Ces propriétés sont décidables, donc l'itérabilité de σ est décidable. ♦

Théorème 3.11
La propriété (PE) est indécidable pour les machines à 2 compteurs déterministes. ♦

Preuve. On prouve ce résultat par réduction de (B) à (PE). Supposons que (PE) est décidable, nous montrons qu'alors (B) est décidable. Lorsqu'une machine à compteur déterministe M est bornée (il existe un entier $b \in \mathbb{N}$ tel qu'en toute configuration accessible $\langle q, m_1, \ldots, m_n \rangle$ de M, $\sum_{i=1}^{n} m_i \leq b$) il est clair que son exécution est périodique puisqu'elle admet un nombre fini de configurations qui admettent chacune au plus une transition sortante. Par conséquent, si l'exécution de M est apériodique, M est non-bornée (de même que pour les systèmes à file réactifs embarqués, voir remarque 3.9).

Si l'exécution de M est ultimement périodique, il existe une séquence cyclique d'instructions :

$$\sigma = q_1 \xrightarrow{c_{i_1}\mathrm{op}_1} q_2, q_2 \xrightarrow{c_{i_2}\mathrm{op}_2} q_3, \ldots, q_n \xrightarrow{c_{i_n}\mathrm{op}_n} q_1$$

dont les itérations répétées forment un suffixe de l'exécution de M. Il est possible de trouver cette séquence en développant l'exécution de M et en appliquant la procédure décrite dans le lemme 3.10 (page 85). Lorsque σ est connue, il est possible de décider si M est bornée : il faut et il suffit que pour tous les compteurs, il y ait exactement autant[4] d'instructions d'incrémentation que d'instructions de décrémentation. À l'inverse, si pour un compteur donné il y a plus d'instructions d'incrémentation que de décrémentation, M est non-bornée.

Or, le problème de la bornitude (B) est indécidable pour les machines à compteurs (déterministes) [80], donc (PE) est indécidable elle aussi. ♦

Par le résultat de simulation du théorème 3.6 (page 76), et par le théorème 3.11 ci-dessus, nous avons :

Théorème 3.12
Le problème (PE) est indécidable pour les SFR Embarqués avec 2 événements mémorisables et un environnement périodique. ♦

3.3.2 SFR Embarqué avec 1 événement mémorisable

Nous prouvons maintenant que lorsqu'un SFR Embarqué est non borné, le nombre de configurations avec une file vide qui apparaîssent sur $[\![S||E]\!]$ est fini.

Lemme 3.13
Soit $S||E$ un système à file réactif embarqué avec un environnement périodique. S'il apparaît infiniment souvent une configuration avec une file vide sur $[\![S||E]\!]$, alors l'ensemble des configurations accessibles de $S||E$ est fini. ♦

[4] Rappelons que pour qu'une séquence σ soit itérable, il faut que σ contienne au moins autant d'instructions d'incrémentation que d'instructions de décrémentation pour chaque compteur.

Preuve. Supposons qu'infiniment souvent il apparaît une configuration ayant une file vide sur $[\![S||E]\!]$. Alors, puisque Q_R (l'ensemble des états de l'AFR qui engendre S) et Q_E sont finis, il existe (au moins) une configuration avec une file vide $\langle q_R, \varepsilon, q_E \rangle$ qui apparaît infiniment souvent sur $[\![S||E]\!]$:

$$[\![S||E]\!] = \langle q_R^0, \varepsilon, q_E^0 \rangle \xrightarrow{\sigma_0}_{S||E} \langle q_R, \varepsilon, q_E \rangle \xrightarrow{\sigma_1}_{S||E} \langle q_R, \varepsilon, q_E \rangle \ldots \langle q_R, \varepsilon, q_E \rangle \xrightarrow{\sigma_n}_{S||E} \cdots$$

où $\sigma_0, \sigma_1, \ldots, \sigma_n, \ldots$ sont des séquences d'actions du système : $\sigma_0 \in \left(A_{S||E}\right)^*$ et $\forall i \geq 1,\ \sigma_i \in \left(A_{S||E}\right)^+$.

Un environnement périodique propose dans chacun de ses états un seul événement au système à file réactif. Donc $S||E$ est déterministe, et par conséquent $\sigma_1 = \cdots = \sigma_n = \sigma_{n+1} = \ldots$. Nous en déduisons que l'ensemble des configurations accessibles est fini. ♦

Nous faisons maintenant l'hypothèse que $S||E$ n'est pas borné. À partir d'un certain point de son exécution, aucune des configurations rencontrées n'a une file vide (lemme 3.13, page 86). Par conséquent, à chaque fois que le système a la possibilité de franchir une transition de traitement différé de e, il prend cette possibilité. En effet, la priorité est donnée au traitement des occurrences mémorisées (voir définition 2.11, page 44). Comme nous le montrons maintenant, cette propriété de *saturation* de la file conduit $S||E$ à se comporter asymptotiquement de façon périodique.

Théorème 3.14

L'exécution d'un SFR Embarqué avec 1 événement mémorisable et un environnement périodique est ultimement périodique. ♦

Preuve. Soit e l'événement mémorisable de $S||E$. D'après le lemme 3.13 (page 86), puisque le système n'est pas borné (par hypothèse), il existe une configuration de $[\![S||E]\!]$ à partir de laquelle toutes les configurations rencontrées sur cette exécution ont une file non vide. Il existe donc $q_R \in Q_R$ (l'ensemble des états de l'AFR R qui engendre S) et $q_E \in Q_E$ tels que nous pouvons extraire de $[\![S||E]\!]$ une sous-séquence infinie de configurations de la forme $\langle q_R, \mathsf{e}^{u_n}, q_E \rangle$ avec :

1. (u_n) strictement croissante,
2. depuis $\langle q_R, \mathsf{e}^{u_n}, q_E \rangle$, $S||E$ exécute une transition de mémorisation de e : $!\mathsf{e}$.

Comme le système est non borné (par hypothèse), les points (1) et (2) sont vérifiés.

Il existe donc une séquence d'actions $\sigma \in \left(A_{S||E}\right)^*$ telle que :

$$\ldots \langle q_R, \mathsf{e}^{u_n}, q_E \rangle \xrightarrow{!\mathsf{e}.\sigma}_{S||E} \langle q_R, \mathsf{e}^{u_{n+1}}, q_E \rangle \xrightarrow{!\mathsf{e}}_{S||E} \cdots$$

Puisque $S||E$ est déterministe, à partir de $\langle q_R, \mathsf{e}^{u_{n+1}}, q_E \rangle$ il répète le même comportement illustré par la séquence d'actions $!\mathsf{e}.\sigma$: les transitions de consommation $?\mathsf{e}$ sont saturées sur σ. En effet, puisque $u_{n+1} > u_n$, la seule différence d'exécution peut provenir d'une transition de traitement différé $?\mathsf{e}$ qui serait désormais franchissable grâce à la présence d'occurrences mémorisées de e supplémentaires. Mais cela est impossible puisqu'aucune des configurations qui apparaîssent sur $[\![\ S||E\ ]\!]$ entre $\langle q_R, \mathtt{e}^{u_n}, q_E\rangle$ et $\langle q_R, \mathtt{e}^{u_{n+1}}, q_E\rangle$ n'a une file vide. Cette transition aurait donc déjà été franchie auparavant sur le chemin de $\langle q_R, \mathtt{e}^{u_n}, q_E\rangle$ à $\langle q_R, \mathtt{e}^{u_{n+1}}, q_E\rangle$. Par conséquent $[\![\ S||E\ ]\!]$ est ultimement périodique. ♦

Chapitre 4

Reset/Lossy SFR Embarqués

L'étude des systèmes à file réactifs embarqués présentée dans le chapitre précédent met en valeur l'influence du nombre d'événements mémorisables sur la puissance du modèle. En effet, alors que les systèmes avec un événement mémorisable sont aisément analysables (au sens où les propriétés d'intérêt sont décidables), ceux qui disposent d'au moins deux événements mémorisables ont l'expressivité des machines de Turing.

Dans ce chapitre, nous utilisons l'approche classique présentée dans [1] pour les machines communicantes ou dans [77] pour les machines à compteurs, qui consiste à considérer que la file de mémorisation n'est pas fiable. Ainsi, en lui permettant de perdre des occurrences d'événements mémorisées de façon non déterministe, nous obtenons la décidabilité de certains des problèmes d'interêt, dont l'accessibilité. Une propriété de sûreté exprime qu'une «mauvaise situation» ne se produit jamais. Puisque le modèle avec perte contient tout les comportements du modèle sans perte, il est alors possible de vérifier des propriétés de sûreté du modèle sans perte (où elles sont indécidables) sur le modèle avec perte (où elles sont donc décidables), pourvu que cette surapproximation soit assez fine.

Dans la première section, nous considérons les SFR Embarqués étendus avec l'opération reset (ou *Reset SFR Embarqués*), dans la continuité des Reset AFR étudiés en section 2.4 (page 54). Cette extension est bien connue pour augmenter la puissance des modèles considérés, comme dans le cas des réseaux de Petri [28], puisque l'opération reset peut être vue comme un «test à zéro faible[1]». Comme nous l'avons montré en section 2.4, dans le cas des systèmes à file réactifs embarqués, l'opération reset peut être simulée par un système sans reset : le modèle dispose intrinsèquement de cette capacité. Nous rappelons donc dans cette section les résultats des sections 3.1 (page 70) et 3.2 (page 78) pour les SFR Embarqués. Puis, dans la section 4.2, nous étudions les *Lossy SFR Embarqués*, qui peuvent perdre leurs occurrences d'événements mémorisées de façon non déterministe et incontrôlée (à l'inverse de l'opération *reset*). Nous montrons, pour ces systèmes, des résultats de décidabilité en section 4.2.1 (page 93), puis des résultats d'indécidabilité en section 4.2.2 (page 103).

---

[1] Au sens où il n'est pas possible de tester si une file vide, mais lorsque la transition est franchie, la file est assurément vide ensuite.

4.1 Reset SFR Embarqués

Nous avons prouvé que dans le cadre des systèmes à file réactifs, l'opération *reset*, qui consiste à consommer toutes les occurrences mémorisées d'un événement lors d'une seule transition, peut être simulée par l'opération de consommation standard (voir le théorème 2.34, page 61). La preuve constructive de cette simulation fait l'hypothèse que chaque occurrence d'événement provenant de l'environnement E est suivie de n occurrences d'un nouvel événement fugace τ, n étant fonction de R. Notons que tout environnement E peut être transformé de façon à respecter notre hypothèse puisque pour toute machine de Turing, le modèle le plus général d'environnement, il est possible de faire suivre de n transitions d'étiquette τ, toute transition de la machine. Par conséquent, pour tout Reset SFR Embarqué à n événements mémorisables $S_{?*} \,||\, E$, il existe un SFR Embarqué $S||E$, à n événements mémorisables et sans reset, qui le simule. Par conséquent :

Théorème 4.1
Tous les problèmes d'intérêt sont indécidables pour les Reset SFR Embarqués avec 2 événements mémorisables et un environnement périodique. ♦

Rappelons que tous les problèmes d'interêt sont au contraire décidables pour les SFR Embarqués avec 1 événement mémorisable, et un environnement rationnel (théorème 3.7, page 79). Donc :

Théorème 4.2
Tous les problèmes d'intérêt sont décidables pour les Reset SFR Embarqués avec 1 événement mémorisable et un environnement rationnel. ♦

4.2 Lossy SFR Embarqués

Considérons maintenant que la file de mémorisation des systèmes à file réactifs (embarqués) n'est pas fiable : elle peut perdre des occurrences d'événements mémorisées à tout moment et de façon non déterministe. À partir de toute configuration $\langle q_R, w, q_E \rangle$ un système à perte $S \overset{\rightsquigarrow}{\parallel} E$ peut atteindre toutes les configurations $\langle q_R, w', q_E \rangle$ avec $w' \preccurlyeq w$, suite à une action de *perte* notée $\overset{\rightsquigarrow}{\tau}$. Bien entendu, $\overset{\rightsquigarrow}{\tau}$ est sans effet sur les états de R et de E. L'opération de perte $\overset{\rightsquigarrow}{\tau}$ n'implique pas de modification syntaxique du modèle (c'est à dire de l'AFR), mais uniquement une extension de sa sémantique obtenue en ajoutant la transition suivante à la définition 2.21 :

4.(d) $\langle q_R, w, q_E \rangle \xrightarrow{\overset{\rightsquigarrow}{\tau}}_{S \overset{\rightsquigarrow}{\parallel} E} \langle q_R, w', q_E \rangle$ si $w' \preccurlyeq w$

Notons que cette sémantique de la perte d'événements correspond exactement à celle de [1, 2, 24] pour les machines communicantes ou à celle nommée «relation de perte classique» dans [77] pour les machines à compteurs. Formellement, un système à file réactif avec perte est défini par :

Définition 4.3 (Lossy SFR)
Un *Lossy système à file réactif (Lossy SFR)* est le système de transitions étiqueté $\overset{\rightsquigarrow}{S} = (Q_{\overset{\rightsquigarrow}{S}}, q^0_{\overset{\rightsquigarrow}{S}}, A_{\overset{\rightsquigarrow}{S}}, \rightarrow_{\overset{\rightsquigarrow}{S}})$, défini à partir d'un SFR $S = (Q_S, q^0_S, A_S, \rightarrow_S)$, par :

1. $Q_{\overset{\leadsto}{S}}$ est l'ensemble des *configurations* de $\overset{\leadsto}{S}$ obtenu depuis $q^0_{\overset{\leadsto}{S}}$ par l'application de la clôture transitive et réflexive de $\rightarrow_{\overset{\leadsto}{S}}$,
2. $q^0_{\overset{\leadsto}{S}} = q^0_S$ est la configuration initiale,
3. $A_{\overset{\leadsto}{S}} = A_S \cup \{\rightsquigarrow\}$ est l'ensemble des actions de $\overset{\leadsto}{S}$,
4. $\rightarrow_{\overset{\leadsto}{S}}$ étend $\rightarrow_S$ par la définition de la perte d'occurrences d'événements mémorisées :

$$\rightarrow_{\overset{\leadsto}{S}} \; = \; \rightarrow_S \; \cup \; \left\{ (q, w) \xrightarrow{\rightsquigarrow}_{\overset{\leadsto}{S}} (q', w') \,|\, q = q' \text{ et } w' \preccurlyeq w \right\}$$

♦

Un *Lossy SFR Embarqué* $S \overset{\rightsquigarrow}{\parallel} E$ est alors obtenu en appliquant la définition 2.21 (page 51) depuis $\overset{\leadsto}{S}$. Les notions de configurations stables ou instables s'appliquent de la même manière que pour les systèmes sans perte. Une séquence de stabilisation peut maintenant contenir des transitions de perte en plus des transitions de consommation.

Remarque 4.4
Notons qu'une transition de perte peut n'avoir aucun effet : la transition

$$\langle q_R, w, q_E \rangle \xrightarrow{\rightsquigarrow}_{S \overset{\rightsquigarrow}{\parallel} E} \langle q_R, w, q_E \rangle$$

est une transition de perte. En terme de relation, nous avons $\xrightarrow{id} \subseteq \xrightarrow{\rightsquigarrow}$.

De même, deux transitions de perte successives :

$$\langle q_R, w, q_E \rangle \xrightarrow{\rightsquigarrow}_{S \overset{\rightsquigarrow}{\parallel} E} \langle q_R, w', q_E \rangle \xrightarrow{\rightsquigarrow}_{S \overset{\rightsquigarrow}{\parallel} E} \langle q_R, w'', q_E \rangle \qquad \equiv \qquad \langle q_R, w, q_E \rangle \xrightarrow{\rightsquigarrow}_{S \overset{\rightsquigarrow}{\parallel} E} \langle q_R, w'', q_E \rangle$$

peuvent être regroupées en une seule puisque $w'' \preccurlyeq w' \preccurlyeq w$. ♦

Par conséquent, nous supposerons dans la suite que toute exécution $\sigma \in [\![\, S \overset{\rightsquigarrow}{\parallel} E \,]\!]$ d'un Lossy SFR Embarqué $S \overset{\rightsquigarrow}{\parallel} E$ est une succession de transitions du système (traitement immédiat, mémorisation et consommation) et de transitions de perte :

$$\sigma = \langle q^0_R, \varepsilon, q^0_E \rangle \xrightarrow{a_0}_{S \overset{\rightsquigarrow}{\parallel} E} \langle q'_R, w', q'_E \rangle \xrightarrow{\rightsquigarrow}_{S \overset{\rightsquigarrow}{\parallel} E} \ldots \langle q''_R, w'', q''_E \rangle \xrightarrow{a_n}_{S \overset{\rightsquigarrow}{\parallel} E} \langle q'''_R, w''', q'''_E \rangle \xrightarrow{\rightsquigarrow}_{S \overset{\rightsquigarrow}{\parallel} E} \ldots$$

avec $\forall i \geq 0,\; a_i \neq \rightsquigarrow$.

Remarque 4.5
D'après la définition 4.3 (page 90) l'action de perte $\rightsquigarrow$ peut se produire aussi bien dans une configuration instable que dans une configuration stable. Il semble pourtant plus naturel de restreindre les occurrences de $\rightsquigarrow$ aux configurations stables : les configurations instables permettent de modéliser facilement la gestion de la file, mais elles n'ont pas d'existence réelle dans le système modélisé. Ces deux approches sont cependant rigoureusement équivalentes : les configurations stables accessibles sont les mêmes.

En effet, considérons une séquence de stabilisation[2] :

$$\langle q_R, w, q_E\rangle \xrightarrow{\mathtt{e}}_{S \ddot{\Vert} E} \langle q_R^1, w_1, q_E^1\rangle \xrightarrow{?\mathtt{e}_1}_{S \ddot{\Vert} E} \ldots \langle q_R^n, w_n, q_E^n\rangle \xrightarrow{?\mathtt{e}_n}_{S \ddot{\Vert} E} \langle q'_R, w', q'_E\rangle$$

où $\langle q_R, w, q_E\rangle$ et $\langle q'_R, w', q'_E\rangle$ sont deux configurations stables du système, e est un événement et les configurations $\langle q_R^1, w_1, q_E^1\rangle, \ldots, \langle q_R^n, w_n, q_E^n\rangle$ sont instables. Considérons la possibilité de perdre une occurrence mémorisée dans l'une des configurations instables de cette séquence. Par définition il n'y a aucune transition de mémorisation sur cette séquence, donc l'occurrence perdue est déjà mémorisée en $\langle q_R, w, q_E\rangle$ et peut donc y être perdue. Par conséquent toutes les transitions de perte issues d'une configuration instable peuvent avoir lieu dans la dernière configuration stable rencontrée, et avec le même effet. Dans la suite, nous permettons l'occurrence des transitions de perte dans les configurations instables pour faciliter les preuves. ♦

Reprenons l'exemple des lecteurs/écrivains décrit dans le chapitre 2 (section 2.1.1, page 34). Supposons que la mémoire de stockage des occurrences d'événements n'est pas fiable, l'arbre des exécutions de la figure 2.5 (page 47) doit être étendu pour tenir compte des pertes d'occurrences mémorisées. Ainsi, pour la configuration $(\varnothing, \mathtt{r_1wr_2})$, nous devons ajouter à l'arbre ses successeurs immédiats obtenus par la perte d'occurrences mémorisées :

$$\begin{tikzcd}[column sep=small]
 & & & & (\varnothing, \mathtt{r_1wr_2}) \arrow[dllll, "?\mathtt{r_1}"'] \arrow[dlll, "\Upsilon"'] \arrow[dll, "\Upsilon"'] \arrow[dl, "\Upsilon"'] \arrow[d, "\Upsilon"] \arrow[dr, "\Upsilon"] \arrow[drr, "\Upsilon"] \arrow[drrr, "\Upsilon"] \arrow[drrrr, "\Upsilon"] & & & & \\
(R_1, \mathtt{wr_2}) & (\varnothing, \varepsilon) & (\varnothing, \mathtt{r_1}) & (\varnothing, \mathtt{w}) & (\varnothing, \mathtt{r_2}) & (\varnothing, \mathtt{r_1w}) & (\varnothing, \mathtt{wr_2}) & (\varnothing, \mathtt{r_1r_2}) & (\varnothing, \mathtt{r_1wr_2})
\end{tikzcd}$$

De la même façon que pour les systèmes sans perte, l'arbre d'accessibilité du système embarqué est obtenu par synchronisation avec l'environnement. Par exemple, la séquence :

$$\langle \varnothing, \varepsilon, 0\rangle \xrightarrow{\mathtt{w}}_{S \ddot{\Vert} E} \langle W, \varepsilon, 1\rangle \xrightarrow{\Upsilon}_{S \ddot{\Vert} E} \langle W, \varepsilon, 1\rangle \xrightarrow{!\mathtt{r_1}}_{S \ddot{\Vert} E} \langle W, \mathtt{r_1}, 2\rangle \xrightarrow{\Upsilon}_{S \ddot{\Vert} E} \langle W, \varepsilon, 2\rangle \ldots$$

est le préfixe d'une exécution du Lossy SFR Embarqué ainsi obtenu.

Propriété 4.6

Un Lossy SFR Embarqué $S \ddot{\Vert} E$ termine (problème (T)) ssi son environnement E termine. ♦

Preuve. D'après la définition de Υ, en toute configuration de $S \ddot{\Vert} E$, il est possible de franchir une transition de perte (éventuellement dépourvue d'effet). Il est donc possible de considérer qu'un Lossy SFR Embarqué ne termine jamais. Mais le fait de perdre des occurrences mémorisées depuis une file vide n'a pas tellement de sens en pratique.

C'est pouquoi, nous considérons plutôt le résultat énoncé ci-dessus. Il est prouvé en suivant l'argumentaire de la remarque 2.24 (page 53). ♦

[2]La séquence de stabilisation mène à proprement parler de $\langle q_R^1, w_1, q_E^1\rangle$ à $\langle q'_R, w', q'_E\rangle$.

Nous introduisons, dans la section suivante, les *systèmes de transitions bien structurés (STEBS).* Puis, nous prouvons que les Lossy SFR Embarqués sont bien structurés si l'environnement est lui-aussi bien structuré.

4.2.1 Résultats de décidabilité dans le cadre d'environnements bien structurés

Systèmes de transitions bien structurés

Les *systèmes de transitions bien structurés* fournissent un cadre général pour l'étude des systèmes de transitions infinis. Les premières définitions de systèmes bien structurés apparaissent dans [35, 36, 37] concernant particulièrement les réseaux de Petri, puis indépendamment Abdulla et *al.*[3, 4] proposent d'autres définitions, cette fois-ci guidées par l'étude des machines communicantes avec perte. Enfin, [41] généralise ces résultats aux systèmes de transitions.

Dans ces approches, les auteurs considèrent des systèmes de transitions S avec un ensemble (infini) d'états Q et un beau-quasi-ordre $\leq$ (voir la section 1.1.1, page 25) sur Q. Ils montrent que la compatibilité entre la relation de transition du système et $\leq$ dans le sens suivant :

"Pour tout état $q_1 \in Q$ tel que $q_1 \to q_2$ et $q_1 \leq q_1'$, $q_1' \in Q$, il existe $q_2' \in Q$ tel que $q_1' \xrightarrow{*} q_2'$ et $q_1' \leq q_2'$"

$$\forall \quad \begin{tikzcd} q_1 \arrow[r, dash, "\leq"] \arrow[d] & q_1' \arrow[d, dashed, "*"] \\ q_2 \arrow[r, dash, dashed, "\leq"] & q_2' \end{tikzcd} \quad \exists$$

permet d'observer certaines propriétés de S. Par exemple, si sur une même branche de l'arbre d'accessibilité de S il existe deux états q et q' (q' accessible depuis q) tels que $q \leq q'$, alors par la compatibilité le système admet une exécution infinie. Cette (bonne) structuration de S permet, sous réserve d'autres hypothèses explicitées dans la suite, de décider certains des problèmes classiques tels que la couverture ou la terminaison.

Définitions et propriétés des STEBS. Par souci de généralité, les articles précédemment cités, et [41] en particulier, considèrent des systèmes de transitions réduits à leur plus simple expression : un ensemble d'états et une relation de transition binaire entre eux. Dans le cadre de notre étude, nous considérons en plus un ensemble d'étiquettes, et donc une relation de transition étiquetée.

Définition 4.7 (Système de transitions étiquetés bien structurés)
Un *système de transitions étiqueté bien structuré (STEBS)* est un triplet $(S, \leq, \pi_S)$, où $S = (Q, q_0 \to, L)$ est un système de transitions étiqueté et $\leq$ un beau-quasi-ordre sur Q, et π_S un morphisme sur $(L, .)$, vérifiant :

"Pour tout état $q_1 \in Q$ tel que $q_1 \xrightarrow{a} q_2$, $a \in L$ et $q_1 \leq q_1'$, $q_1' \in Q$, il existe $q_2' \in Q$ tel que $q_1' \xrightarrow{\sigma} q_2'$, $\sigma \in L^+$, $\pi_S(\sigma) = a$ et $q_2 \leq q_2'$"

$$\forall \quad \begin{tikzcd} q_1 \arrow[r, dash, "\leq"] \arrow[d, "a"'] & q_1' \arrow[d, dashed, "\sigma"] \\ q_2 \arrow[r, dash, dashed, "\leq"] & q_2' \end{tikzcd} \quad \exists$$

♦

Le morphisme π_S permet de définir un *critère d'observation* sur les étiquettes des transitions de S. Par exemple, supposons que S donne la sémantique d'un système avec perte. En choisissant $\pi_S(\vec{\tau}) = \varepsilon$ et $\pi_S(a) = a,\ a \neq \vec{\tau}$, cela permet d'obtenir l'équivalence (pour la bonne structuration) entre les traces sans perte et avec perte, pour peu qu'elles contiennent les même étiquettes $a \neq \vec{\tau}$, en même nombre et dans le même ordre. C'est à dire que pour un observateur extérieur au système modélisé, qui ne peut donc pas observer les transitions de perte, π_S capture l'égalité sur les séquences d'étiquettes du système.

Cette définition diffère légèrement de celle de [41] où $\pi_S = id$, entrainant : $\sigma = a$. La définition 4.7 requiert simplement que le système soit bien structuré par rapport au critère d'observation défini par π_S. Notons que lorsque $\pi_S = id$, la relation $\leq$ correspond exactement à la notion de simulation usuelle («à la Milner») pour les systèmes de transitions étiquetés.

Dans la définition 4.7, la compatibilité[3] entre $\rightarrow$ et $\leq$ repose sur l'existence d'une séquence menant de q'_1 à q'_2. Nous n'avons cependant fait jusqu'ici aucune hypothèse sur cette séquence. Dans la suite, nous considérons trois formes[4] de compatibilité [41] :

- **Compatibilité forte.** Dans le cas où la séquence de q'_1 à q'_2 est constituée d'une et une seule transition. C'est à dire $\pi_S = id$.
- **Compatibilité transitive.** Lorsque la séquence considérée est composée d'au moins une transition, c'est à dire $\pi_S(\sigma) = a$.
- **Compatibilité propagée.** Compatibilité transitive, et de plus, tous les états rencontrés sur la séquence de q'_1 à q'_2, $q'_{11}, \ldots, q'_{1n}$, vérifient $\forall i \in \{1, \ldots, n\},\ q_1 \leq q'_{1i}$. La figure 4.1 (page 94) représente une situation de compatibilité propagée.

Remarque 4.8

Dans le cadre d'une compatibilité propagée, la relation liant q_1 à q'_1 se transmet à tous les états rencontrés sur la séquence excepté q'_2, d'où son nom. ♦

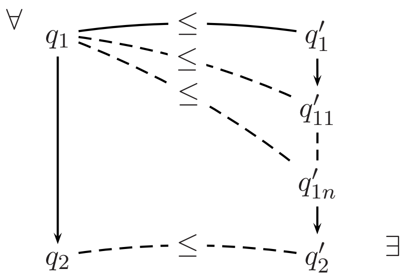


FIG. 4.1 – Compatibilité propagée pour les STEBS.

Enfin, une compatibilité (forte, transitive, propagée) est dite *stricte* lorsque nous en considérons la relation $<$ en lieu et place de $\leq$ dans les définitions précédentes.

[3] Notons que cette forme de compatibilité est dite «vers le haut»(par analogie à la fermeture d'un ensemble). Il existe aussi des résultats de décidabilité lorsque la compatibilité inverse (donc «vers le bas») est considérée [72, 41].

[4] Dans[41], les auteurs considèrent de plus la compatibilité *refléxive* pour laquelle la séquence de q'_1 à q'_2 est soit vide, soit de longueur 1. Cette propriété s'avère utile dans le cadre d'une compatibilité "vers le bas", c'est pourquoi nous ne la considérons pas présentement.

Résultats de décidabilité pour les STEBS. Nous décrivons maintenant comment les propriétés des beaux-quasi-ordres (voir en section 1.1.1, page 25) permettent de décider les problèmes de couverture (C), de terminaison (T), de bornitude (B) et de model-checking de (EG) pour les STEBS (sous réserve de certaines hypothèses). Les résultats et les argumentations qui les accompagnent proviennent de [41].

– **Couverture (C).** Rappelons que $Pred$ donne l'ensemble des *prédécesseurs immédiats* d'un état (ou d'un ensemble d'états) d'un système de transitions étiqueté (voir la section 1.3.1, page 28), et qu'une *base finie* pour un ensemble clos par le haut $X \subseteq E$ est un ensemble fini $Y \subseteq E$ tel que $\uparrow Y = X$ (voir section 1.1.2, page 25).

  Définition 4.9 (Pred-base effective [41])
  Un STEBS a une *Pred-base effective* [41] s'il existe un algorithme qui calcule une base finie $pb(q)$ pour $\uparrow Pred(\uparrow q)$, quel que soit l'état $q \in Q$. ♦

  Rappelons que (C) consiste à décider si $q' \in Pred^*(\uparrow q)$ (q' couvre q). Supposons qu'un STEBS S dispose d'une Pred-base effective. Alors, la série $(\uparrow K_n)_{n \geq 0}$ définie par $K_0 = q$ et $K_{n+1} = K_n \cup pb(K_n)$ converge, au sens où $\exists m, \uparrow K_m = \uparrow K_{m+1}$, (par le lemme 1.5, page 26) vers $Pred^*(\uparrow q)$. Il suffit alors que $\leq$ soit une relation décidable pour que (C) soit décidable.

– **Terminaison (T).**

  Définition 4.10 (Arbre d'accessibilité fini [41])
  L'arbre d'accessibilité fini [41] $FRT(S)$ du système de transitions S est l'arbre orienté dont les nœuds sont étiquettés par les états q de S (noté $n : q$) de la façon suivante :
  – La racine n_0 est associée à l'état initial de S : $n_0 : q_0$,
  – Un nœud $n : q$ a un fils par successeur de q, sauf lorsque sur la branche de n_0 à n il existe un nœud $n' : q'$ avec $q' \leq q$ (q couvre q') auquel cas le nœud est une feuille de l'arbre (il n'est pas développé).

  ♦

  Une feuille $n : q$ est *couvrante* s'il existe un noeud $n' : q'$ sur la branche menant de la racine $n_0 : q_0$ à $n : q$, tel que $q' \leq q$.
  Alors, S admet une exécution infinie si et seulement si $FRT(S)$ admet une feuille couvrante (par la compatibilité). Le calcul de $FRT(S)$ est possible à partir du moment où $Succ$ est effective.

– **Bornitude (B).** Nous considérons, pour ce problème, que la compatibilité entre $\leq$ et $\rightarrow$ est stricte. La bornitude de S se réduit à décider s'il existe une feuille de $FRT(S)$ qui est *strictement* couverte. En effet, si une telle feuille existe, par la compatibilité stricte, S admet une exécution infinie où un nombre infini d'états différents sont visités. À l'inverse, si S n'est pas borné, puisque chacun de ses états accessibles peut être atteint sans exécuter de cycle, il existe une exécution sans cycle de S qui est un chemin maximal de $FRT(S)$. Par la définition 4.10 (page 95), la feuille de cette branche (qui correspond au chemin maximal) $n' : q'$ est couverte par un nœud $n : q$ tels que $q' \neq q$, et, en supposant en outre que $<$ est antisymétrique[5], $q' < q$.

– **Model-checking de (EG).**

Définition 4.11 (Propriété close vers le haut ou vers le bas)
Nous dirons qu'une propriété ϕ est *close vers le haut (resp. vers le bas)* si et seulement si l'ensemble des états de S qui satisfont ϕ :

$$\{q \in Q \,|\, q \vDash \phi\}$$

est clos vers le haut (resp. vers le bas). ♦

Lorsque la compatibilité est propagée, S a une exécution telle que tous les états visités sont dans un ensemble clos I si et seulement si il existe un chemin maximal de $FRT(S)$ qui vérifie cette propriété. Alors, (EG), avec des propriétés atomiques closes vers le haut, est décidable.

Le tableau suivant présente des conditions suffisantes pour obtenir la décidabilité des problèmes de la couverture, de la bornitude, de la terminaison et du model-checking du fragment (EG) de la logique Ctl.

	Couverture **(C)**	Terminaison **(T)**	Bornitude **(B)**	Model-checking[a] **(EG)**
Compatibilité	transitive	transitive	transitive stricte	propagée
$\leq$	(bqo) décidable	(bqo) décidable	(bqo) antisymétrique décidable	(bqo) décidable
effectivité	Pred-base	$Succ$	$Succ$	$Succ$

[a] Pour des propriétés atomiques closes vers le haut.

Tab. 4.1 – Résultats de décidabilité pour les STEBS [41].

La bonne structuration des Lossy SFR Embarqués-BS

Définition 4.12 (Lossy SFR Embarqué-BS)
Un *Lossy SFR Embarqué à environnement bien structuré (Lossy SFR Embarqué-BS)* est un Lossy SFR Embarqué $S \overset{\leadsto}{\shortparallel} E$ tel que $(E, \leq_E, \pi_E)$ est un STEBS avec compatibilité transitive vérifiant :

– $\leq_E$ est un beau-quasi-ordre décidable sur Q_E l'ensemble des états de E,
– π_E est un morphisme sur A_E vérifiant $\pi_E(\mathsf{e}) = \mathsf{e}$ pour tout $\mathsf{e} \in (\Sigma)_S$ et $\pi_E(\mathsf{e}) \notin (\Sigma)_S$ si $\mathsf{e} \notin (\Sigma)_S$, $(\Sigma)_S$ étant l'alphabet des événements de S.
– $Succ_E$ est effective.

♦

Remarque 4.13
Par la définition de π_E imposée à l'environnement, et par la définition des STEBS (voir définition 4.7,

[5] Dans ce cas $\leq$ est un ordre partiel.

page 93) nous avons la certitude que l'action a et la séquence σ mises en oeuvre pour la compatibilité dans E :

$$\begin{tikzcd}
\forall & q_1 \arrow[r, dash, "\leq"] \arrow[d, "a"'] & q'_1 \arrow[d, dashed, "\sigma"] & \\
& q_2 \arrow[r, dash, dashed, "\leq"] & q'_2 & \exists
\end{tikzcd}$$

sont indistingables pour S. En effet, il retrouve dans a et σ la même séquence d'événements qui le contraignent. ♦

Dans cette section, nous prouvons que les problèmes de la couverture (C), de l'accessibilité (R), de la terminaison (T) et du model-checking du fragment (EG) sont décidables. Tous ces résultats découlent de ceux de [41] pour les STEBS. Il reste donc à établir, d'une part que les Lossy SFR Embarqués-BS sont bien structurés, et d'autre part, qu'ils disposent d'une Pred-base effective.

Les Lossy SFR Embarqués-BS sont bien structurés. Pour prouver ce résultat, nous introduisons tout d'abord un quasi-ordre sur les configurations de $S \overset{\rightsquigarrow}{\shortparallel} E$. Soit $\leq_{S \overset{\rightsquigarrow}{\shortparallel} E}$ la relation binaire sur $Q_{S \overset{\rightsquigarrow}{\shortparallel} E}$ définie par :

$$\langle q_R, w, q_E \rangle \leq_{S \overset{\rightsquigarrow}{\shortparallel} E} \langle q'_R, w', q'_E \rangle \;\Leftrightarrow\; q_R = q'_R \text{ et } w \preccurlyeq w' \text{ et } q_E \leq_E q'_E \tag{4.1}$$

Dans la suite, lorsque nous comparons deux configurations de $S \overset{\rightsquigarrow}{\shortparallel} E$, nous ne différencions pas les états de R : nous écrivons explicitement qu'elles partagent le même état de R.

Lemme 4.14

$\leq_{S \overset{\rightsquigarrow}{\shortparallel} E}$ est décidable. ♦

Preuve. Les relations $=$ et $\preccurlyeq$ sont triviallement décidables sur Q_R et Σ_M^* respectivement et par hypothèse, $\leq_E$ est décidable. ♦

Lemme 4.15

$\leq_{S \overset{\rightsquigarrow}{\shortparallel} E}$ est un beau-quasi-ordre. ♦

Preuve. L'ensemble des états de R, Q_R, est fini, donc $=$ est un bqo sur Q_R. Les mots w et w' sont des mots finis sur Σ_M, l'alphabet des événements mémorisables de R, et l'ordre sous-mot $\preccurlyeq$ est un bqo sur un espace (éventuellement infini) de mots finis par le lemme de Higman [62]. Finalement, $\leq_E$ est beau-quasi-ordre sur Q_E par hypothèse. Donc $\leq_{S \overset{\rightsquigarrow}{\shortparallel} E}$ est un bqo puisqu'il est l'intersection de trois beaux-quasi-ordres. ♦

Soit $\pi_{S \overset{\rightsquigarrow}{\shortparallel} E}$ le morphisme sur $A_{S \overset{\rightsquigarrow}{\shortparallel} E}$ défini par :

$$\pi_{S \overset{\rightsquigarrow}{\shortparallel} E}(a) = \begin{cases} \varepsilon & \text{si } a = \overset{\rightsquigarrow}{\tau} \\ a & \text{sinon} \end{cases} \tag{4.2}$$

Ce morphisme observe les exécutions du système en ignorant les transitions de perte.

Théorème 4.16
Soit $S \overset{\rightsquigarrow}{\shortparallel} E$ un Lossy SFR Embarqué-BS, $(S \overset{\rightsquigarrow}{\shortparallel} E, \leq_{S \overset{\rightsquigarrow}{\shortparallel} E}, \pi_{S \overset{\rightsquigarrow}{\shortparallel} E})$ est un STEBS avec compatibilité propagée. ♦

Preuve. Soit $(E, \leq_E, \pi_E)$ l'environnement bien structuré de $S \overset{\rightsquigarrow}{\shortparallel} E$. Considérons les trois configurations suivantes de $S \overset{\rightsquigarrow}{\shortparallel} E$:

$$\begin{tikzcd}
\langle q_R^1, w_1, q_E^1\rangle \arrow[r, dash, "\leq_{R \overset{\rightsquigarrow}{\shortparallel} E}"] \arrow[d, "a"'] & \langle q_R^1, w_1', q_E'^1\rangle \\
\langle q_R^2, w_2, q_E^2\rangle &
\end{tikzcd}$$

Nous prouvons l'existence d'une configuration $\langle q_R^2, w_2', q_E'^2\rangle$ qui établit la compatibilité propagée entre $\rightarrow_{S \overset{\rightsquigarrow}{\shortparallel} E}$ et $\leq_{S \overset{\rightsquigarrow}{\shortparallel} E}$ quelle que soit $a \in A_{S \overset{\rightsquigarrow}{\shortparallel} E}$.

Remarquons tout d'abord que si $a = \overset{\rightsquigarrow}{\imath}$ (a est une action de perte) il est possible d'exécuter exactement la même action depuis $\langle q_R^1, w_1', q_R'^1\rangle$ et la compatiblitée (propagée) est évidente. Nous considérons donc les autres action possibles pour $S \overset{\rightsquigarrow}{\shortparallel} E$ $(a \neq \overset{\rightsquigarrow}{\imath})$:

1. **soit** $\langle q_R^1, w_1, q_E^1\rangle$ **est stable** ; si $\langle q_R^1, w_1', q_E'^1\rangle$ est stable aussi, $S \overset{\rightsquigarrow}{\shortparallel} E$ peut exécuter la même action a car les deux configurations partagent le même état de R et l'environnement est bien structuré. Le système atteint alors la configuration $\langle q_R^2, w_2', q_E'^2\rangle$ où $q_E^2 \leq_E q_E'^2$ par hypothèse et $w_2 \preccurlyeq w_2'$ quelle que soit a (mémorisation ou traitement immédiat). Et puisque $\pi_{S \overset{\rightsquigarrow}{\shortparallel} E}(a) = a$:

$$\begin{tikzcd}
\langle q_R^1, w_1, q_E^1\rangle \arrow[r, dash, "\leq_{R \overset{\rightsquigarrow}{\shortparallel} E}"] \arrow[d, "a"'] & \langle q_R^1, w_1', q_E'^1\rangle \arrow[d, dashed, "a"] \\
\langle q_R^2, w_2, q_E^2\rangle \arrow[r, dash, dashed, "\leq_{R \overset{\rightsquigarrow}{\shortparallel} E}"] & \langle q_R^2, w_2', q_E'^2\rangle
\end{tikzcd}$$

   Mais, si $\langle q_R^1, w_1', q_E'^1\rangle$ est instable, $S \overset{\rightsquigarrow}{\shortparallel} E$ ne peut pas y exécuter a puisque la priorité est donnée au traitement des occurrences d'événements mémorisées. Cependant, le système peut tout d'abord perdre les occurrences mémorisées qui rendent cette configuration instable : ce sont les symboles de w_1' qui n'appartiennent pas à $\left(\Sigma^!_{q_R^1}\right)^*$ et atteindre ainsi la configuration $\langle q_R^1, w_1'', q_E'^1\rangle$. Nous avons $w_1 \preccurlyeq w_1''$ puisque $w_1 \in \left(\Sigma^!_{q_R^1}\right)^*$, $w_1 \preccurlyeq w_1'$ et w_1'' est obtenu en retirant tous les symboles de w_1' qui n'appartient pas à $\Sigma^!_{q_R^1}$. De plus, cette configuration est stable. Alors, $S \overset{\rightsquigarrow}{\shortparallel} E$ peut maintenant exécuter l'action a atteignant la configuration $\langle q_R^2, w_2', q_E'^2\rangle$. Finalement $\pi_{S \overset{\rightsquigarrow}{\shortparallel} E}(\overset{\rightsquigarrow}{\imath}\ .a) = a$ et nous obtenons la compatibilité propagée :

$$\begin{tikzcd}
\langle q_R^1, w_1, q_E^1\rangle \arrow[r, dash, "\leq_{R \overset{\rightsquigarrow}{\shortparallel} E}"] \arrow[dd, "a"'] \arrow[dr, dash, dashed, "\leq_{R \overset{\rightsquigarrow}{\shortparallel} E}"'] & \langle q_R^1, w_1', q_E'^1\rangle \arrow[d, dashed, "\overset{\rightsquigarrow}{\imath}"] \\
 & \langle q_R^1, w_1'', q_E'^1\rangle \arrow[d, dashed, "a"] \\
\langle q_R^2, w_2, q_E^2\rangle \arrow[r, dash, dashed, "\leq_{R \overset{\rightsquigarrow}{\shortparallel} E}"] & \langle q_R^2, w_2', q_E'^2\rangle
\end{tikzcd}$$

2. **soit** $\langle q_R^1, w_1, q_E^1\rangle$ **est instable**, et par conséquent $\langle q_R^1, w'_1, q'^1_E\rangle$ est aussi instable. Soient $w_{11} \in \left(\Sigma^!_{q_R^1}\right)^*$ et $w_{12} \in \Sigma_M^*$ les deux mots finis tels que $w_1 = w_{11}\mathsf{e}w_{12}$ avec $\mathsf{e} \notin \Sigma^!_{q_R^1}$. Alors, depuis $\langle q_R^1, w_1, q_E^1\rangle$, $S \overset{\rightsquigarrow}{\shortparallel} E$ consomme e. De même, nous notons $w'_{11} \in (\Sigma_M \setminus \{\mathsf{e}\})^*$ et $w'_{12} \in \Sigma_M^*$ les deux mots finis tels que $w'_1 = w'_{11}\mathsf{e}w'_{12}$.
Lorsque $w'_{11} \in \left(\Sigma^!_{q_R^1}\right)^*$, $S \overset{\rightsquigarrow}{\shortparallel} E$ consomme e depuis $\langle q_R^1, w'_1, q'^1_E\rangle$ puisque c'est la plus ancienne occurrence mémorisée qui peut être consommée dans cette configuration. En remarquant que $\pi_{S \overset{\rightsquigarrow}{\shortparallel} E}(?\mathsf{e}) = ?\mathsf{e}$ nous obtenons la compatibilité propagée entre $\rightarrow_{S \overset{\rightsquigarrow}{\shortparallel} E}$ et $\leq_{S \overset{\rightsquigarrow}{\shortparallel} E}$:

$$\begin{tikzcd}
\langle q_R^1, w_1, q_E^1\rangle \arrow[r, dash, "\leq_{R \overset{\rightsquigarrow}{\shortparallel} E}"] \arrow[d, "?\mathsf{e}"'] & \langle q_R^1, w'_1, q'^1_E\rangle \arrow[d, dashed, "?\mathsf{e}"] \\
\langle q_R^2, w_2, q_E^2\rangle \arrow[r, dash, dashed, "\leq_{R \overset{\rightsquigarrow}{\shortparallel} E}"] & \langle q_R^2, w'_2, q'^2_E\rangle
\end{tikzcd}$$

À l'opposé, si $w'_{11} \notin \left(\Sigma^!_{q_R^1}\right)^*$ il existe un événement $\mathsf{e}' \in \left(\Sigma_M \setminus \Sigma^!_{q_R^1}\right)$ dont une occurrence mémorisée figure dans w'_{11} : elle doit donc être consommée en priorité. Comme pour le cas précédent, $S \overset{\rightsquigarrow}{\shortparallel} E$ peut perdre les occurrences mémorisées gênantes : celles qui apparaîssent dans w'_{11} tout en n'appartenant pas à $\Sigma^!_{q_R^1}$. Le système atteint alors la configuration $\langle q_R^1, w''_{11}\mathsf{e}w'_2, q'^1_R\rangle$ avec $w_{11} \preccurlyeq w''_{11} \preccurlyeq w'_{11}$ puisque w''_{11} est obtenu depuis w'_{11} en perdant des occurrences mémorisées qui n'appartiennent pas à $\Sigma^!_{q_R^1}$ alors que $w_{11} \in \left(\Sigma^!_{q_R^1}\right)^*$ et $w_{11} \preccurlyeq w'_{11}$. Depuis cette dernière configuration, $S \overset{\rightsquigarrow}{\shortparallel} E$ consomme la plus ancienne occurrence mémorisée de e et puisque $\pi_{S \overset{\rightsquigarrow}{\shortparallel} E}(\overset{\rightsquigarrow}{l}\ .?\mathsf{e}) = ?\mathsf{e}$, nous obtenons la compatibilité propagée :

$$\begin{tikzcd}
\langle q_R^1, w_1, q_E^1\rangle \arrow[r, dash, "\leq_{R \overset{\rightsquigarrow}{\shortparallel} E}"] \arrow[dr, dash, dashed, "\leq_{R \overset{\rightsquigarrow}{\shortparallel} E}"] \arrow[dd, "?\mathsf{e}"] & \langle q_R^1, w'_1, q'^1_E\rangle \arrow[d, dashed, "\overset{\rightsquigarrow}{l}"] \\
 & \langle q_R^1, w''_1, q'^1_E\rangle \arrow[d, dashed, "?\mathsf{e}"] \\
\langle q_R^2, w_2, q_E^2\rangle \arrow[r, dash, dashed, "\leq_{R \overset{\rightsquigarrow}{\shortparallel} E}"] & \langle q_R^2, w'_2, q'^2_E\rangle
\end{tikzcd}$$

♦

La relation $\leq_{S \overset{\rightsquigarrow}{\shortparallel} E}$ n'entre pas en compatibilité stricte avec $\rightarrow_{S \overset{\rightsquigarrow}{\shortparallel} E}$. Il est donc interessant de chercher une relation qui permette ce type de compatibilité dans le but d'obtenir la décidabilité du problème de la bornitude.

Propriété 4.17
Il n'existe pas d'ordre partiel (i.e. de (bqo) antisymétrique) $<_{S \overset{\rightsquigarrow}{\shortparallel} E}$ tel que $(S \overset{\rightsquigarrow}{\shortparallel} E, <_{S \overset{\rightsquigarrow}{\shortparallel} E}, \pi_{S \overset{\rightsquigarrow}{\shortparallel} E})$ est un système de transitions étiquetté bien structuré avec compatibilité stricte. ♦

Preuve. Si $<_{S \overset{\rightsquigarrow}{\shortparallel} E}$ existe, alors (B) est décidable (voir tableau 4.1, page 96) alors que ce problème est indécidable comme nous le montrons dans le théorème 4.27, page 106. ♦

Les Lossy SFR Embarqués-BS ont une Pred-base effective. Rappelons que l'opération Pred-base consiste à calculer une base finie pour l'ensemble $\uparrow Pred(\uparrow q)$ (si elle est appliquée à q). Nous devons par conséquent étendre la définition de $Pred_{S||E}$ (définition 2.15, page 46) pour prendre en compte les transitions de perte. Depuis une configuration donnée $\langle q_R, w, q_E\rangle$, il est possible d'atteindre, par une transition de perte, toutes les configurations $\langle q_R, w', q_E\rangle$ avec $w' \preccurlyeq w$. Par conséquent, pour un Lossy SFR Embarqué-BS, $S \mathrel{\widetilde{\|}} E$, l'ensemble des prédecesseurs d'une configuration est défini par :

$$Pred_{S \widetilde{\|} E}(\langle q_R, w, q_E\rangle) = Pred_{S||E}(\langle q_R, w, q_E\rangle) \cup \{\langle q_R, w', q_E\rangle \mid w' \in \uparrow w\} \tag{4.3}$$

Propriété 4.18

Un Lossy SFR Embarqué-BS, $S \mathrel{\widetilde{\|}} E$, a une Pred-base effective si son environnement E a une Pred-base effective. ♦

Preuve. Soit $S \mathrel{\widetilde{\|}} E$, un Lossy SFR Embarqué-BS tel que $(E, \leq_E, \pi_E)$ est un STEBS avec Pred-base effective. Soit $\widetilde{S}$ le Lossy SFR obtenu depuis R en respect de la définition 4.3 (page 90). Nous notons $\leq_{\widetilde{S}}$ la relation binaire sur $Q_{\widetilde{S}}$ définie par :

$$(q_R, w) \leq_{\widetilde{S}} (q'_R, w') \;\Leftrightarrow\; q_R = q'_R \text{ et } w \preccurlyeq w'$$

Cette relation est un beau-quasi-ordre puisque $=$ est un bqo sur l'ensemble fini des états de R et $\preccurlyeq$ est un bqo sur un ensemble de mots finis par le lemme de Higman [62]. De plus, il est clair que $\leq_{\widetilde{S}}$ est décidable.

D'après la définition de $Pred_{S \widetilde{\|} E}$ de l'équation (4.3) ci-dessus, nous avons :

$$\begin{aligned}
Pred^{\mathsf{e}}_{S \widetilde{\|} E}(\uparrow\langle q_R, w, q_E\rangle) &= Pred^{\mathsf{e}}_{\widetilde{S}}(\uparrow(q_R, w)) \times Pred^{\mathsf{e}}_{E}(\uparrow q_E)\\
Pred^{!\mathsf{e}}_{S \widetilde{\|} E}(\uparrow\langle q_R, w, q_E\rangle) &= Pred^{!\mathsf{e}}_{\widetilde{S}}(\uparrow(q_R, w)) \times Pred^{\mathsf{e}}_{E}(\uparrow q_E)\\
Pred^{?\mathsf{e}}_{S \widetilde{\|} E}(\uparrow\langle q_R, w, q_E\rangle) &= Pred^{?\mathsf{e}}_{\widetilde{S}}(\uparrow(q_R, w)) \times q_E\\
Pred^{\widetilde{l}}_{S \widetilde{\|} E}(\uparrow\langle q_R, w, q_E\rangle) &= \uparrow\langle q_R, w, q_E\rangle
\end{aligned} \tag{4.4}$$

où la clôture vers le haut d'une configuration de $\widetilde{S}$, $\uparrow(q_R, w)$, est définie relativement à $\leq_{\widetilde{S}}$. Puisque par hypothèse $(E, \leq_E, \pi_E)$ a une Pred-base effective, il suffit de prouver que c'est aussi le cas de $\widetilde{S}$.

Le cas $a = \widetilde{l}$ est résolu puisqu'ici $\langle q_R, w, q_E\rangle$ est une base finie pour $Pred^{\widetilde{l}}_{S \widetilde{\|} E}(\uparrow\langle q_R, w, q_E\rangle)$. Nous considérons alors une action $a \in A_{\widetilde{S}}$ différente d'une action de perte $a \neq \widetilde{l}$. Lors du calcul de $Pred^{a}_{\widetilde{S}}((q_R, w))$ la principale difficulté provient du calcul pour la file. En effet, $Pred^{a}_{R}(q_R)$ est aisément obtenu, mais l'ensemble des mots $w' \in \Sigma^*_M$ tels qu'il existe $w'' \in \uparrow w$, $w \preccurlyeq w''$ et $(q'_R, w') \xrightarrow{a}_{\widetilde{S}} (q_R, w'')$ est plus difficilement caractérisable. Dans la suite, t représente la transition $q'_R \xrightarrow{a}_R q_R$ et nous notons :

$$Pred^{t}(\uparrow w) = \{w' \in \Sigma^*_M \mid \exists w'' \in \uparrow w \text{ t.q. } (q'_R, w') \xrightarrow{a}_{\widetilde{S}} (q_R, w'')\} \tag{4.5}$$

l'ensemble des contenus de file w' qui permettent de couvrir w, par un mot w'', après le franchissement de la transition $(q'_R, w') \xrightarrow{a}_{\widetilde{S}} (q_R, w'')$. Supposons que nous pouvons calculer une base finie pour $\uparrow Pred^t(\uparrow w)$, alors, d'après l'équation (4.4), nous avons :

$$\uparrow Pred^a_{\widetilde{S}}(\uparrow(q_R, w)) = \bigcup_{t=q'_R \xrightarrow{a}_R q_R} \left(q'_R \times \uparrow Pred^t(\uparrow w)\right) \tag{4.6}$$

et nous en déduisons que $\widetilde{S}$ a une Pred-base effective, et par conséquent il en est de même pour $S \,\widetilde{\shortparallel}\, E$.

Il nous reste maintenant à prouver que nous pouvons calculer une base finie pour $\uparrow Pred^t(\uparrow w)$. Notons tout d'abord que pour tout alphabet fini Σ, quel que soit $w \in \Sigma^*$, $\uparrow w = w \sqcup\!\sqcup \Sigma^*$ et quel que soit $\Sigma' \subseteq \Sigma$, $\uparrow(\Sigma'^*) = \Sigma^*$. Examinons maintenant chacune des possibilités pour l'action a :

$\boxed{\mathbf{a} = \mathsf{e}, \quad (t = q'_R \xrightarrow{\mathsf{e}}_R q_R)}$ Une transition de traitement immédiat $(q'_R, w') \xrightarrow{\mathsf{e}}_{\widetilde{S}} (q_R, w)$ laisse la file invariante : $w = w'$ (voir (4b), définition 2.11, page 44). Par conséquent, lorsque $w \notin (\Sigma^!_{q'_R})^*$, quel que soit $w'' \in \uparrow w$, $w'' \notin (\Sigma^!_{q'_R})^*$, donc la transition $(q'_R, w'') \xrightarrow{\mathsf{e}}_{\widetilde{S}} (q_R, w'')$ n'est pas franchissable. Alors $Pred^t(\uparrow w) = \emptyset$ et l'ensemble vide constitue une base finie pour $\uparrow Pred^t(\uparrow w)$.

À l'inverse, lorsque $w \in (\Sigma^!_{q'_R})^*$, $Pred^t(\uparrow w)$ est l'ensemble des mots w' tels que (1) (q'_R, w') est stable (la transition est franchissable) et (2) w' couvre w, c'est à dire $Pred^t(\uparrow w) = (\Sigma^!_{q'_R})^* \cap \uparrow w = (\Sigma^!_{q'_R})^* \sqcup\!\sqcup\, w$. Donc :

$$\begin{aligned} \uparrow Pred^t(\uparrow w) &= \uparrow\left(\left(\Sigma^!_{q'_R}\right)^* \sqcup\!\sqcup\, w\right) \\ &= \uparrow\left(\left(\Sigma^!_{q'_R}\right)^*\right) \sqcup\!\sqcup \uparrow w \\ &= (\Sigma_M)^* \sqcup\!\sqcup \uparrow w \\ &= \uparrow w \end{aligned}$$

Par conséquent, w constitue une base finie pour $\uparrow Pred^t(\uparrow w)$.

$\boxed{\mathbf{a} = !\mathsf{e}, \quad (t = q'_R \xrightarrow{!\mathsf{e}}_R q'_R)}$ Remarquons tout d'abord que même si w ne se termine pas par e, $Pred^t(\uparrow w)$ n'est pas vide puisque cet ensemble contient les prédecesseurs des mots qui couvrent w, qui peuvent, eux, se terminer par e. Nous devons donc différencier ces deux cas : notons z le mot défini par $z = w$ si $w \notin (\Sigma^*_M.\mathsf{e})$ et $z.\mathsf{e} = w$ sinon.

De même que pour le cas précédent, $Pred^t(\uparrow w) = \emptyset$ lorsque $w \notin (\Sigma^!_{q'_R})^*$ et la base vide est alors génératrice de $\uparrow Pred^t(\uparrow w)$.

Lorsque $w \in (\Sigma^!_{q'_R})^*$ les mots w' qui permettent (1) de franchir $q'_R \xrightarrow{!\mathsf{e}}_R q'_R$ (tels que (q'_R, w') est stable) et (2) de couvrir w sont définis par : $w' \in (\Sigma^!_{q'_R})^* \sqcup\!\sqcup \uparrow z$. Par le même calcul que pour le cas précédent $\uparrow Pred^t(\uparrow w) = \uparrow z$, alors z constitue la base finie recherchée.

$\boxed{\mathbf{a} = ?\mathsf{e}, \quad (t = q'_R \xrightarrow{?\mathsf{e}}_R q_R)}$ Soit w_1 le plus grand préfixe de w qui appartient à $(\Sigma^!_{q'_R})^*$ et w_2 le mot fini tel que $w = w_1 w_2$. Pour franchir la transition $q'_R \xrightarrow{?\mathsf{e}}_R q_R$ il faut que la file en q'_R contienne une occurrence de e qui soit la plus ancienne occurrence consommable en q'_R. Donc cette file appartient à $(w_1 \sqcup\!\sqcup\, \mathsf{e}).w_2 = \bigcup_{w_{11} \in prefix(w_1)} (w_{11}.\mathsf{e}).(w_{11}^{-1} w_1 w_2)$.

Nous cherchons l'ensemble des files w' (1) qui permettent de franchir la transition $q'_R \xrightarrow{?\mathsf{e}}_R q_R$ et (2) telles que les files w'' obtenues après l'exécution de la transition $(q'_R, w') \xrightarrow{?\mathsf{e}}_{\mathcal{S}} (q_R, w'')$ couvrent w (c'est à dire $w \preccurlyeq w''$). Par conséquent :

$$Pred^t(\uparrow w) = \bigcup_{w_{11} \in prefix(w_1)} \Big((w_{11}.\mathsf{e}) \sqcup\!\sqcup (\Sigma^!_{q'_R})^* \Big) .\uparrow(w_{11}^{-1} w_1 w_2)$$

Il vient alors :

$$\uparrow Pred^t(\uparrow w) = \uparrow \left[\bigcup_{w_{11} \in prefix(w_1)} \Big((w_{11}.\mathsf{e}) \sqcup\!\sqcup (\Sigma^!_{q'_R})^* \Big) .\uparrow(w_{11}^{-1} w_1 w_2) \right]$$

Par le lemme 1.3 (page 26) :

$$\begin{aligned}
\uparrow Pred^t(\uparrow w) &= \bigcup_{w_{11} \in prefix(w_1)} \uparrow\Big((w_{11}.\mathsf{e}) \sqcup\!\sqcup (\Sigma^!_{q'_R})^* \Big).\uparrow\big(\uparrow(w_{11}^{-1} w_1 w_2)\big) \\
&= \bigcup_{w_{11} \in prefix(w_1)} \uparrow(w_{11}.\mathsf{e}).\uparrow(w_{11}^{-1} w_1 w_2) \\
&= \uparrow \bigcup_{w_{11} \in prefix(w_1)} \big[(w_{11}.\mathsf{e}.w_{11}^{-1} w_1).w_2\big] \\
&= \uparrow((w_1 \sqcup\!\sqcup \mathsf{e}).w_2)
\end{aligned}$$

Puisque w_1 est fini (car w est fini), $(w_1 \sqcup\!\sqcup \mathsf{e}).w_2$ constitue une base finie pour $\uparrow Pred^t(\uparrow w)$.

♦

Résultats de décidabilité pour les Lossy SFR Embarqués-BS. Il suffit désormais d'appliquer les résultats du tableau 4.1, et nous obtenons :

Théorème 4.19

Les problèmes suivants sont décidables pour les Lossy SFR Embarqués-BS, pourvu que les conditions requises soient vérifiées :

	Couverture (C)	Accessibilité (R)	Model-checking (EF)	Model-checking (EG)
Conditions nécessaires	Pred-base effective pour E			Propriétés atomiques closes vers le haut

♦

Les trois premiers résultats sont la conséquence directe de la bonne structure des Lossy SFR Embarqués-BS. La décidabilité de l'accessibilité (R) vient d'une part de la décidabilité de la couverture, et, d'autre part, de la définition de $\curlyvee$. La décidabilité du model-checking de (EF) provient alors directement de l'équivalence avec (R).

Notons que la décidabilité du model-checking de (EF) permet de décider, par négation des formules, la satisfaction des propriétés de CTL contruites uniquement avec les modalités AG et AX, et des propriétés atomiques closes vers le bas. La plupart des formules de sûreté, informellement «une situation mauvaise de ne produit jamais», peuvent s'exprimer par une propriété de la forme AGϕ. Dans ce cas ϕ décrit les bonnes propriétés qui doivent toujours être vérifiées.

Remarque 4.20
Les Lossy SFR Embarqués-BS constituent une surapproximation[6] des SFR Embarqués, au sens où, quels que soient S et E, $[\![S||E]\!] \subseteq [\![S \stackrel{\rightarrow}{\shortparallel} E]\!]$. Toute formule de sûreté :

– exprimée sur les configurations stables,
– et invariante par bégaiement,

vérifiée par $S \stackrel{\rightarrow}{\shortparallel} E$ est donc aussi vérifiée par $S||E$. Ce résultat positif nous permet donc de vérifier des propriétés de sûreté pour les SFR Embarqués, à condition toute fois que la surapproximation soit assez fine. ♦

4.2.2 Résultats d'indécidabilité pour des environnements ultimement périodiques

La bonne structure des Lossy SFR Embarqués-BS n'est cependant pas suffisante pour obtenir la décidabilité des autres problèmes d'interêt. En effet, comme nous le montrons maintenant, les Lossy SFR Embarqués sont équivalents aux machines à compteurs avec perte (Lossy machines à compteurs (LMC)) pour lesquelles l'indécidabilité des problèmes (B), (MC-LTL) et (MC-CTL) est établie.

Lossy machines à compteurs

À la manière des systèmes à file réactifs embarqués, les *Lossy machines à compteurs* [77], sont obtenues depuis les machines à compteurs (définition 3.1, page 70) par une simple extension de leur sémantique, c'est à dire en ajoutant la règle :

4. $\langle q, m_1, \ldots, m_n\rangle \to \langle q, m_1', \ldots, m_n'\rangle$ si $\forall i \in \{1, \ldots, n\},\ m_i' \leq m_i$.

à celles présentées en section 3.1.1, page 70. Comme pour les SFR Embarqués, la perte, c'est à dire, la décroissance spontanée des compteurs, n'est pas déterministe, ni controllable par la machine (ce qui se traduit par le fait qu'elle ne change pas d'état q lors de la transition).

Remarque 4.21
Dans [77], Mayr considère une définition de l'action de perte plus générale que celle-ci :

$$\langle q, m_1, \ldots, m_n\rangle \to \langle q, m_1', \ldots, m_n'\rangle \text{ si } \sum_{i=1}^{n} m_i' \leq \sum_{i=1}^{n} m_i$$

La relation que nous considérons est par ailleurs baptisée «relation de perte classique» dans [77]. Nous présentons les résultats de [77] en nous restreignant à la «relation de perte classique» du fait

[6]Abstraction faite des transitions de perte.

de notre étude. ♦

Remarque 4.22
La construction de Minsky [80] qui permet de transformer une machine à n compteurs, en une machine à 2 compteurs équivalente, ne s'étend pas aux machines à compteurs avec perte : le codage utilisé (via les nombres de Gödel) n'est pas robuste vis à vis de la relation de perte. **Les nombres de compteurs cités dans les résultats ont donc leur importance**. ♦

Lemme 4.23 (Mayr [77])
Les résultats suivant sont indécidables pour les Lossy machines à compteurs :

	Bornitude (B)	Model-checking de LTL (Mc-LTL)	Model-checking de CTL (Mc-CTL)
Nb. de compteurs nécessaires	3^*	3^*	4
Type de machine	déterministe	Non déterministe	

♦

Pour les résultats marqués par un asterisque $*$ dans le lemme 4.23, Mayr donne une preuve pour des Lossy machines avec au moins 4 compteurs. Nous donnons en annexe B (page 201) les preuves complètes de ces résultats pour 3 compteurs.

Remarque 4.24
La différence portant sur le nombre d'événements mémorisables pour le model-checking de Ltl et de Ctl provient de la différence d'expressivité de ces deux logiques. Ainsi, le problème de l'état récurrent[7] utilisé pour la preuve de l'indécidabilité du model-checking de Ltl ne peut pas être exprimé en Ctl puisqu'elle ne peut pas distinguer une exécution particulière de la machine. À notre connaissance, le problème du model-checking de Ctl pour les Lossy machines à 2 et 3 compteurs, ainsi que le problème du model-checking de Ltl pour les Lossy machines à 2 compteurs sont ouverts. ♦

Corollaire 4.25
L'ensemble des configurations accessibles $RS(\overset{\leadsto}{M})$ d'une Lossy machine à 3 compteurs $\overset{\leadsto}{M}$ est reconnaissable, mais il n'existe pas de procédure effective pour en calculer une représentation finie. ♦

Preuve. Par la définition de $\overset{\leadsto}{\mathcal{T}}$, $RS(\overset{\leadsto}{M})$ est clos vers le bas et donc reconnaissable. Par contre, l'indécidabilité de la bornitude (B) implique qu'il n'existe pas de procédure effective pour calculer une représentation finie de $RS(\overset{\leadsto}{M})$. ♦

[7] Est-ce qu'un état q donné est visité infiniment souvent lors de l'exécution de la machine ?

Simulation des LMC par les Lossy SFR Embarqués

Maintenant que les résultats d'indécidabilité sont établis pour les Lossy Machines à compteurs, il reste à prouver leur simulation par les Lossy SFR Embarqués. Ce résultat n'est pas immédiat, car la simulation dans le cas sans perte (établie par le théorème 3.6, page 76) n'est pas préservée par l'ajout de la sémantique de perte : alors que les instructions d'une Lossy machine à compteurs sont atomiques vis-à-vis des actions de perte, cela n'est pas le cas des structures d'AFR utilisées pour la simulation (présentées en page 73) et en définition 3.5, page 75). En effet, la perte peut se produire au beau milieu de l'exécution de ces structures de simulation.

De plus, la remarque 4.22 (page 104) impose que nous prouvions la simulation des Lossy machines à n compteurs déterministes par les Lossy SFR Embarqués avec n événements mémorisables.

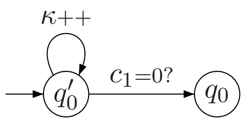
Cependant, cette relation de simulation est préservée malgré la sémantique de perte.

Théorème 4.26

Pour toute Lossy machine à n compteurs déterministe M, il existe un Lossy SFR Embarqué avec n événements mémorisables et un environnement périodique, qui simule M ♦

La preuve complète de ce théorème est donnée en annexe C, page 205.

Résultats d'indécidabilité. Le théorème 4.26 nous permet d'obtenir immédiatement l'indécidabilité de la bornitude pour les Lossy SFR Embarqués. Cependant, l'obtention de l'indécidabilité du model-checking pour Ltl et Ctl n'est pas aussi immédiate. En effet, les Lossy machines à compteurs $\overset{\leadsto}{M}$ utilisées pour ces preuves ne sont pas déterministes. Ceci provient de l'utilisation d'une boucle d'incrémentation sur l'état initial[8] :

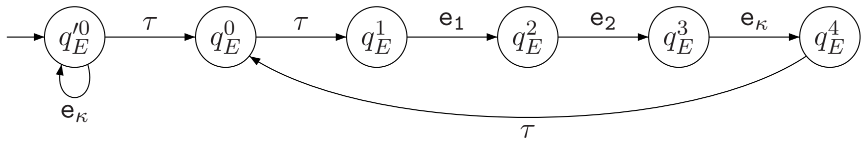


pour «deviner» la valeur de la capacité κ qui permet à la machine d'avoir une exécution infinie. Notons que les autres instructions de $\overset{\leadsto}{M}$ respectent le déterminisme.

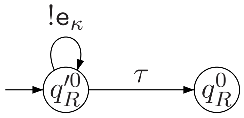
Or, un (Lossy) SFR Embarqué à environnement périodique ne peut pas simuler l'indéterminisme : il a une et une seule exécution. Nous devons donc augmenter la puissance de l'environnement afin de simuler les machines $\overset{\leadsto}{M}$ précédemment considérées.

Considérons la Lossy machine à 3 compteurs $\overset{\leadsto}{M}$ construite pour prouver l'indécidabilité du model-checking de Ltl (lemme B.1, page 201). Soit $S \overset{\leadsto}{\parallel} E$ le Lossy SFR Embarqué correspondant, par le théorème 4.26, à la machine $\overset{\leadsto}{M}$, excepté les instructions issues de q_0'. Il reste à simuler les instructions de q_0' représentées ci-dessus. Nous nommons $\mathtt{e}_1$, $\mathtt{e}_2$ et $\mathtt{e}_\kappa$ les événements qui simulent respectivement les compteurs c_1, c_2 et κ de $\overset{\leadsto}{M}$. Nous augmentons la puissance de l'environnement E en lui permettant tout d'abord d'émettre un nombre indéterminé d'occurrences de $\mathtt{e}_\kappa$:

[8] κ est le troisième compteur de la machine considérée dans cette preuve. Il sert à stocker la «capacité» [29] de la machine, c'est à dire, le nombre de transitions de perte qu'elle peut effectuer.



Puis les instructions de q_0' sont traduites dans R de façon à permettre la mémorisation de ces premières occurrences de e_κ pour atteindre une configuration $\langle q_R^0, w, q_E^0\rangle$ avec $\Psi w = \begin{pmatrix} 0 & 0 & n \end{pmatrix}$, $n \in \mathbb{N}$. Ceci se transcrit par la structure :



Nous obtenons donc un Lossy SFR Embarqué **à environnement ultimement périodique** qui simule $\overset{\leadsto}{M}$. Nous procédons de même pour l'indécidabilité de (Mc-CTL), avec cette fois-ci 4 compteurs.

Par conséquent, nous avons les résultats suivants :

Théorème 4.27
Les problèmes suivants sont indécidables pour les Lossy SFR Embarqués :

	Bornitude (B)	Model-checking Ltl (Mc-LTL)	Model-checking Ctl (Mc-CTL)
Nb d'événements mémorisables	3	3	4
Environnement	périodique	ultimement périodique	

De plus, l'espace des configurations accessibles d'un Lossy SFR Embarqué est reconnaissable, sans qu'il existe toutefois de procédure effective pour le calculer. ♦

Chapitre 5

Test de non-bornitude pour les SFR Embarqués

Dans les deux chapitres précédents, nous avons montré que le problème de la bornitude (B) n'est généralement pas décidable pour les systèmes à file réactifs embarqués. Cependant, cette propriété du modèle est cruciale puisque :

- dans le but de *vérifier* que celui-ci est bien conforme aux spécifications qui ont guidé sa conception, les techniques de vérification utilisées diffèrent suivant que le modèle est fini ou infini,
- dans le but d'*implanter* le système, il faut s'assurer qu'il est bien borné sachant qu'il n'existe pas de dispositif qui permette la mémorisation d'un nombre non borné d'éléments.

Nous proposons donc une procédure de test qui va parfois nous permettre d'apporter une réponse à ce problème. Elle repose sur les mêmes principes que les tests $\mathcal{U}$ de [66] et $\mathcal{V}$ de [39] définis pour les automates à file FIFO. Il s'agit de développer l'arbre d'accessibilité à la recherche d'un comportement du système qui rend la file non bornée.

Comme nous l'expliquons dans la première section, cela se résume à chercher des comportements périodiques qui sont les seuls vraiment détectables, comme dans le cas des automates à file FIFO [66, 39]. Dans la deuxième section, nous montrons justement comment décider si une séquence est itérable, en tenant donc compte de la politique FIFFO qui est appliquée à la file. Pour finir cette section, nous présentons un algorithme de décision de l'itérabilité dont la complexité est linéaire en la longueur de la séquence considérée. Ceci nous a permis d'implémenter notre test de façon efficace dans l'outil TETU, introduit en section 5.3.

5.1 Principe du test

L'idée de notre test est d'examiner les exécutions d'un système à file réactifs $S||E$, à la recherche d'éléments qui nous permettent d'établir que celui-ci n'est pas borné. Rappelons en effet que lorsque $S||E$ est borné, le développement de ses exécutions permet facilement de s'en assurer, et c'est par conséquent lorsque $S||E$ n'est pas borné qu'intervient la difficulté. Nous consacrons donc la suite de ce chapitre à la découverte d'un critère, noté $\mathcal{I}$, qui nous indique pendant l'exploration des exécutions de $S||E$ que celui-ci n'est pas borné.

5.1.1 Exemple

Prenons pour exemples d'une part le système à file réactif $(S||E)_{ap}$ de la figure 3.7 (page 83) dont l'exécution est apériodique et égale à la séquence d'actions :

$$[\![(S||E)_{ap}]\!] = \tau \mathsf{e}_1\mathsf{e}_2\tau\tau\mathsf{e}_1!\mathsf{e}_2\tau \bigodot_{n=1}^{\infty}\Big[(?\mathsf{e}_1\tau?\mathsf{e}_2!\mathsf{e}_1!\mathsf{e}_2\tau)^{2(n-1)} (?\mathsf{e}_2\tau!\mathsf{e}_1\mathsf{e}_2\tau)\, \tau?\mathsf{e}_1!\mathsf{e}_1\mathsf{e}_2\tau\tau!\mathsf{e}_1\mathsf{e}_2\tau$$
$$(?\mathsf{e}_1\tau?\mathsf{e}_1!\mathsf{e}_1!\mathsf{e}_2\tau)^{2n-1} (?\mathsf{e}_1\tau\mathsf{e}_1!\mathsf{e}_2\tau)\, \tau\mathsf{e}_1?\mathsf{e}_2!\mathsf{e}_2\tau\tau\mathsf{e}_1!\mathsf{e}_2\tau\Big]$$

et d'autre part le système à file réactif $(S||E)_p$ obtenu en plongeant l'AFR de la figure 3.5(a) (page 82) dans l'environnement représenté en figure 5.1 dont une partie de l'arbre d'accessibilité

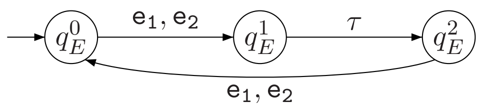


Fig. 5.1 – Environnement rationnel E.

est montrée en figure 5.2. Dans cet arbre, les nœuds n sont étiquetés par des configurations du système. Les nœuds encadrés ont déjà été rencontrés sur la même branche. Ils correspondent donc à des cycles dans l'exécution du système. Les deux systèmes, $(S||E)_p$ et $(S||E)_{ap}$, sont non

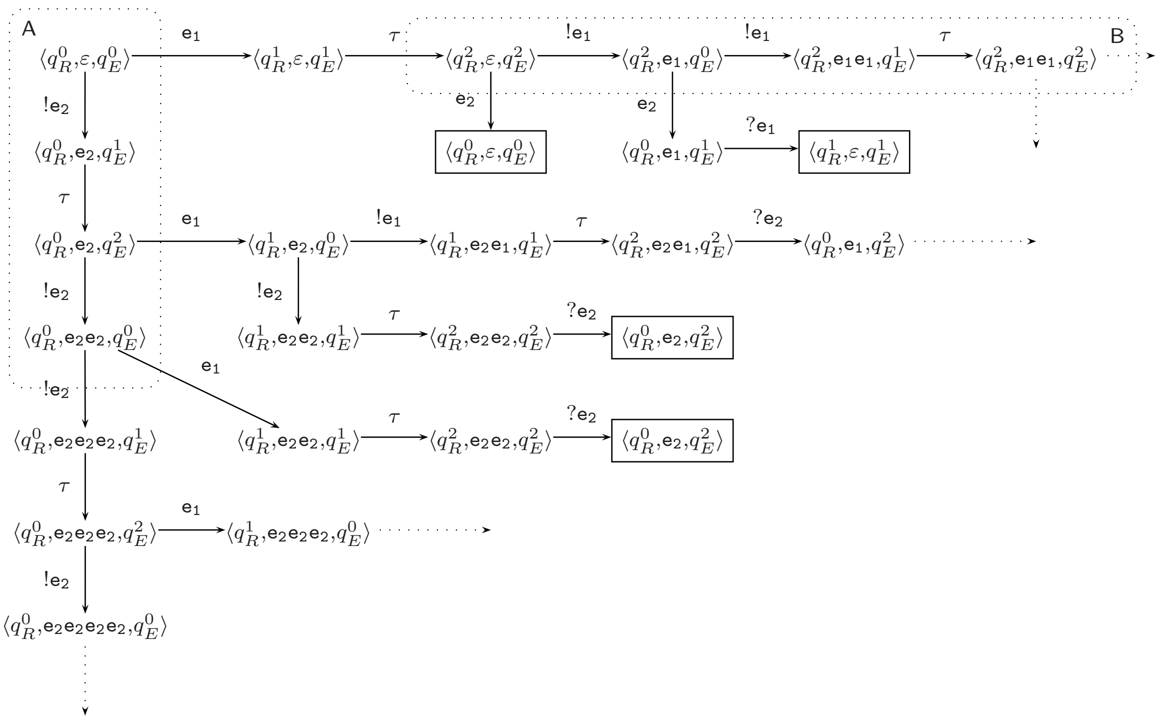


Fig. 5.2 – Arbre d'accessibilité $RT((S||E)_p)$.

bornés : $(S||E)_{ap}$ car son exécution est apériodique (voir la remarque 3.9, page 80) et $(S||E)_p$

car il apparaît clairement que les comportements A et B, entourés dans son arbre d'accessibilité, sont itérables et font croître la taille de la file.

5.1.2 Relation $\mathcal{I}$

Nous souhaitons trouver une relation $\mathcal{I}$, entre les nœuds de l'arbre d'accessibilité du système, qui capture la loi d'évolution de celui-ci. Ainsi, pour détecter la non-bornitude de $(S||E)_{ap}$, la relation $\mathcal{I}$ doit lier les configurations $\langle q_R^3, (\mathsf{e}_2)^{2n+1}, q_E^0\rangle$, avec $n \geq 0$, qui engendrent le comportement apériodique (voir section 3.3.1). Alors que pour $(S||E)_p$, il faut relier les configurations de la forme $\langle q_R^0, (\mathsf{e}_2)^{2n}, q_E^0\rangle$, avec $n \geq 0$, lorsque la séquence d'actions $!\mathsf{e}_2\tau!\mathsf{e}_2$ est considérée, et les configurations $\langle q_R^2, (\mathsf{e}_1)^{2n}, q_E^2\rangle$, où $n \geq 0$, lorsque nous considérons la séquence $!\mathsf{e}_1!\mathsf{e}_1\tau$ puisque ces exécutions sont itérables et elles font croître le nombre d'occurrences mémorisées.

D'une façon générale et à la manière de [66], nous cherchons donc une relation $\mathcal{I}$ telle que :

$$\forall \sigma \in \left(A_{S||E}\right)^*,\ s \xrightarrow{\sigma} s' \text{ et } \mathcal{I}(s,s') \ \Rightarrow\ \exists \sigma' \in \left(A_{S||E}\right)^*,\ s' \xrightarrow{\sigma'} s'' \text{ et } \mathcal{I}(s',s'') \tag{5.1}$$

où s, s' et s'' sont trois nœuds de l'arbre d'accessibilité du système $S||E$.

Cependant, la propriété (5.1) introduit une définition trop générale de $\mathcal{I}$. Il est en effet impossible dans la plupart des cas de déterminer l'existence d'un telle séquence σ'. Mais, bien souvent, lorsque le système n'est pas borné, il admet une exécution périodique (constituée par l'itération infinie d'une séquence σ) sur laquelle la taille de la file croît. Nous choisissons donc de restreindre $\mathcal{I}$ à la détection de comportements périodiques :

$$\forall \sigma \in \left(A_{S||E}\right)^*,\ s \xrightarrow{\sigma} s' \text{ et } \mathcal{I}(s,s') \ \Rightarrow\ s' \xrightarrow{\sigma} s'' \text{ et } \mathcal{I}(s',s'') \tag{5.2}$$

La relation $\mathcal{I}$ définie par l'équation (5.2) est donc une *relation d'iterabilité infinie.*

Lemme 5.1
Soit $S||E$ un système à file réactif. Supposons que nous connaissons une relation $\mathcal{I}$ correspondant à l'équation (5.2), et soit σ la séquence itérable capturée par $\mathcal{I}$.

S'il existe $\mathsf{e} \in \Sigma_M$ tel que $|\sigma|_{!\mathsf{e}} > |\sigma|_{?\mathsf{e}}$, alors $S||E$ est non borné. ♦

Preuve. Par la définition de $\mathcal{I}$, dans l'équation (5.2) ci-dessus, la séquence σ est infiniment itérable. S'il existe un tel événement e, alors dans les configurations qui étiquettent les nœuds $s, s', s'', \ldots$ de $RT(S||E)$ liés par $\mathcal{I}$, le nombre d'occurrences mémorisées de e est strictement croissant. On en déduit que $S||E$ admet une exécution pour laquelle il est non borné (celle qui consiste à itérer σ infiniment). ♦

Le test de non-bornitude s'effectue alors en développant l'arbre d'accessibilité à la recherche de deux nœuds s et s' apparaissant sur la même branche et tels que $\mathcal{I}(s,s')$. Lorsque de plus la séquence σ qui lie s à s' respecte les critères du lemme 5.1, le test s'arrête et nous concluons que le système n'est pas borné. Il est alors possible d'exhiber le comportement incriminé. Lorsque cette séquence laisse la file bornée, la recherche se poursuit puisqu'il est alors impossible de tirer la moindre conclusion sur la bornitude de $S||E$. D'où l'appelation de «Test de non-bornitude».

À la manière de Karp & Miller, lorsqu'un cycle est trouvé dans l'arbre d'accessibilité (deux occurrences d'une même configuration sur une même branche de l'arbre), le nœud concerné n'est pas développé. Dans le cas où E est fini, nous avons la garantie que notre test se termine lorsque $S||E$ est borné.

Rappelons que nous avons montré en section 3.3.1 (page 85) que pour des environnements périodiques, la périodicité de l'exécution d'un SFR Embarqué est indécidable lorsqu'il possède 2 événements mémorisables (théorème 3.12, page 86), alors qu'elle est décidable s'il ne possède pas plus d'un événement mémorisable (théorème 3.14, page 87). Par conséquent, l'obtention de la relation $\mathcal{I}$ nous donne, conjointement à la méthode de test, un algorithme pour calculer la borne de la file lorsque le SFR Embarqué considéré n'a pas plus d'un événement mémorisable. Dans le cas des SFR Embarqués avec 2 événements mémorisables (et plus), nous pouvons seulement espérer obtenir cette information par ce test puisqu'il n'existe pas d'algorithme.

Il reste maintenant à caractériser la relation $\mathcal{I}$, et donc, à s'intéresser aux conditions qui différencient une séquence itérable, d'une séquence non itérable.

5.2 Décision de l'itérabilité d'une séquence

Soit σ une séquence finie de transitions dans un SFR Embarqué $S||E$:

$$\sigma = \langle q_R^1, w_1, q_E^1 \rangle \xrightarrow{a_1}_{S||E} \langle q_R^2, w_2, q_E^2 \rangle \xrightarrow{a_2}_{S||E} \dots \xrightarrow{a_n}_{S||E} \langle q_R^{n+1}, w_{n+1}, q_E^{n+1} \rangle \tag{5.3}$$

Pour que cette séquence soit itérable, il faut tout d'abord que $q_R^1 = q_R^{n+1}$: c'est à dire que cette séquence forme un cycle dans R. Ensuite, les 2 conditions nécessaires suivantes, **(CN1)** et **(CN2)** doivent être satisfaites :

– l'environnement E peut répéter infiniment la séquence d'événements σ_E qui contraint σ :

(CN1) σ_E est un cycle itérable dans E

– σ est auto suffisante vis-à-vis des occurrences mémorisées. En effet, puisque, dans chaque configuration, il n'y a qu'un nombre fini d'occurrences mémorisées, si un événement donné est consommé plus de fois qu'il est mémorisé sur σ, après un nombre fini d'itérations de la séquence, il n'y a plus d'occurrence mémorisée de cet événement, et σ n'est donc plus exécutable.

(CN2) pour tout événement mémorisable $\mathtt{e}$, $|a_1 \dots a_n|_{!\mathtt{e}} \geq |a_1 \dots a_n|_{?\mathtt{e}}$.

Désormais, nous faisons donc l'hypothèse que les conditions **(CN1)** et **(CN2)** sont vérifiées par σ. Nous pouvons donc nous intéresser à l'itérabilité des cycles dans les SFR, indépendament de l'environnement considéré. Il reste donc à formuler des conditions suffisantes d'itérabilité, en considérant particulièrement les occurrences d'événements mémorisées.

Pour pouvoir exécuter la séquence σ, il faut que chacune des transitions qui composent σ soit franchissable. La possibilité de franchir une transition $(q, w) \xrightarrow{a}_S (q', w')$, dans un SFR S, est dépendante de sa priorité sur les autres transitions issues de q. Par exemple, une transition d'action $a = \mathsf{e}$, $\mathsf{e} \in \Sigma$, est inhibée en (q, w) dès lors que w contient une occurrence consommable en q : $w \notin \left(\Sigma^!_q\right)^*$ (point (4b), définition 2.11, page 44). Nous notons alors $p(a, q)$ les propriétés qui doivent être vérifiées par w pour que la transition $(q, w) \xrightarrow{a}_S (q', w')$ soit franchissable. Conformément à la définition 2.11 (page 44), les conditions de franchissement, pour chaque type de transitions, sont définies par les propriétés suivantes :

– Pour une transition de traitement immédiat $\langle q_R, w, q_E\rangle \xrightarrow{\mathsf{e}}_{S||E} \langle q'_R, w, q'_E\rangle$, il ne faut pas que w contienne une occurrence d'événement qui peut être consommée en q_R. En effet, dans ce cas, la transition de traitement différé a la priorité. La file w doit donc vérifier $w \in \left(\Sigma^!_{q_R}\right)^*$, ce que nous notons par la propriété : $p(\mathsf{e}, q_R) = \bigwedge_{\mathsf{e}' \in \left(\Sigma_M \setminus \Sigma^!_{q_R}\right)} [\mathsf{e}' \notin]$.

– Pour une transition de mémorisation $\langle q_R, w, q_E\rangle \xrightarrow{!\mathsf{e}}_{S||E} \langle q_R, w\mathsf{e}, q'_E\rangle$, de même que précédemment, la configuration $\langle q_R, w, q_E\rangle$ doit être stable. Ici encore, nous notons cette propriété : $p(!\mathsf{e}, q_R) = \bigwedge_{\mathsf{e}' \in \left(\Sigma_M \setminus \Sigma^!_{q_R}\right)} [\mathsf{e}' \notin]$.

– Finalement, pour une transition de consommation $\langle q_R, w_1\mathsf{e}w_2, q_E\rangle \xrightarrow{?\mathsf{e}}_{S||E} \langle q'_R, w_1w_2, q'_E\rangle$, il faut que e soit prioritaire, c'est à dire qu'il n'existe par d'événement dans w_1 qui puisse être consommé en q_R :$w_1 \in \left(\Sigma^!_{q_R}\right)^*$. De plus, e doit effectivement être mémorisé dans w. Nous notons tout ceci : $p(?\mathsf{e}, q_R) = \left(\bigwedge_{\mathsf{e}' \in \left(\Sigma_M \setminus \Sigma^!_{q_R}\right)} [\mathsf{e} \sqsubseteq \mathsf{e}']\right) \wedge [\mathsf{e} \in]$.

Ces propriétés expriment des notions d'appartenance et de précédence de symboles dans la file de mémorisation. À chaque instant, cette file ne contient qu'un nombre fini d'occurrences mémorisées d'événements. Nous définissons donc la sémantique de ces propriétés sur les mots finis w par :

– $w \models [\mathsf{e} \in] \;\Leftrightarrow\; \exists i \in \{1, \ldots, |w|\}$ t.q. $w_i = \mathsf{e}$,
– $w \models [\mathsf{e} \notin] \;\Leftrightarrow\; w \not\models [\mathsf{e} \in]$,
– $e \models [\mathsf{e} \sqsubseteq \mathsf{e}'] \;\Leftrightarrow\; \forall i < \min_k\{w_k = \mathsf{e}\},\; w_i \neq \mathsf{e}'$.

et un mot w vérifie une conjonction de propriétés s'il vérifie chacune d'elles.

Remarquons que les propriétés $p(a, q)$ sont formées de conjonctions de propriétés d'appartenance et/ou de précédence qui font intervenir, au plus, 2 événements. De plus, la satisfaction des propriétés $[\mathsf{e} \in]$, $[\mathsf{e} \notin]$ ne dépend que de la présence de l'événement e dans le mot w sur lequel elles sont interprétées. De même, la satisfaction d'une propriété de précédence $[\mathsf{e} \sqsubseteq \mathsf{e}']$ ne dépend que de l'ordre des premières occurrences de e et e', dans le mot w sur lequel elle est interprétée. Donc, la satisfaction des propriétés d'appartenance et de précédence est uniquement liée aux événements qu'elles impliquent, et en aucun cas, aux autres événements mémorisables de l'AFR considéré. Dans la suite, nous considérons donc que les propriétés d'appartenance et de précédence sont interprétées sur des mots finis dont l'alphabet est formé des événements utilisés dans la propriété. Par conséquent, $[e \in]$ est interprétée sur les mots finis de $\{\mathsf{e}\}^*$ et $[\mathsf{e} \sqsubseteq \mathsf{e}']$ est interprétée sur les mots finis formés sur l'alphabet $\{\mathsf{e}, \mathsf{e}'\}$.

En notant $p(a_i, q^i_R)$ la propriété qui définit les conditions dans lesquelles a_i est exécutable en q^i_R, il vient que la séquence σ est exécutable si et seulement si : $w_1 \models p(a_1, q^1_R)$, $w_2 \models p(a_2, q^2_R)$,. . .et $w_n \models p(a_n, q^n_R)$. En effet, chaque transition doit être franchissable : donc pour tout $i \in \{1, \ldots, n\}$, il est nécessaire que $w_i \models p(a_i, q^i_R)$. Et, si la séquence est exécutable, chacune de ses transitions est franchissable, et par conséquent, pour tout $i \in \{1, \ldots, n\}$: $w_i \vDash p(a_i, q_R^i)$. Puis, à partir de $\langle q_R^1, w_{n+1}, q_E^{n+1} \rangle$, il est à nouveau possible d'exécuter σ si et seulement si $w_{n+1} \vDash p(a_1, q_R^1)$, $w_{n+2} \vDash p(a_2, q_R^2)$,. . .et $w_{2n} \vDash p(a_n, q_R^n)$, et ainsi de suite.

L'itérabilité de σ se résume donc à l'invariance des propriétés associées à chacune des actions de σ, lors de la modification de la file par σ. Notons $\sigma_i = a_i \ldots a_n.a_1 \ldots a_{i-1}$ la séquence d'actions obtenue depuis σ par rotation de $i-1$ positions, et $\sigma_i(w)$ la file obtenue après exécution de la séquence σ_i depuis une configuration ayant pour file w. Dans ce formalisme, la condition $w_i \vDash p(a_i, q_R^i)$, avec $i \in \{1, \ldots, n\}$, nécessaire et suffisante pour l'exécution de a_i, lors de la seconde exécution de σ, s'écrit : $\sigma_i(w_i) \vDash p(a_i, q_R^i)$. Par conséquent, pour le franchissement de a_i lors de la m^{e} exécution consécutive de σ, la condition nécessaire est suffisante est : $\sigma_i^m(w_i) \vDash p(a_i, q_R^i)$.

L'itérabilité de σ se caractérise par la possibilité d'exécuter σ un nombre m infini de fois, c'est à dire :

$$\mathcal{I}(s, s') \;\Leftrightarrow\; \forall m \geq 0,\; \sigma_1^m(w_1) \vDash p(a_1, q_R^1) \,\wedge\, \ldots \,\wedge \sigma_n^m(w_n) \vDash p(a_n, q_R^n) \tag{5.4}$$

où s et s' sont deux nœuds de l'arbre d'accessibilité liés par σ.

Dans la suite nous considérons une séquence σ_i et sa propriété associée $p(a_i, q_R^i)$. Nous nous intéressons à l'invariance de $p(a_i, q_R^i)$ par les applications successives de σ_i.

5.2.1 Fiffo-transformation

Notre approche de l'itérabilité d'une séquence σ est donc basée sur la transformation apportée à la file par σ. Dans cette section, nous introduisons les Fiffo-*transformations* qui permettent la modélisation de l'effet de σ.

Définition 5.2 (Fiffo-transformation)
Soient Σ un alphabet fini et, α et ρ, deux mots finis sur Σ. La Fiffo-*transformation* $\theta_{\ominus\rho}^{+\alpha}$ est définie, pour tout mot fini w tel que $\forall \mathsf{e} \in \Sigma,\ |w.\alpha|_{\mathsf{e}} \geq |\rho|_{\mathsf{e}}$, par : $\theta_{\ominus\rho}^{+\alpha}(w) = \rho^{\ominus 1}(w.\alpha)$. ♦

Une Fiffo-transformation $\theta_{\ominus\rho}^{+\alpha}$ est *itérable* si pour tout événement $\mathsf{e} \in \Sigma$, $|\alpha|_{\mathsf{e}} \geq |\rho|_{\mathsf{e}}$. Cette condition impose que la Fiffo-transformation soit auto-suffisante, c'est à dire qu'elle ajoute au moins autant d'occurrences de e qu'elle en consomme. Dans le cas contraire, elle épuise cette ressource et ne peut donc être appliquée consécutivement qu'un nombre fini de fois. C'est la traduction de la condition nécessaire **(CN2)**. Nous introduisons tout d'abord quelques propriétés des Fiffo-transformations afin de prouver qu'elles permettent la modélisation des séquences de systèmes à file réactifs (embarqués) dans la propriété 5.4.

Lemme 5.3
Soient α, α', ρ et ρ' quatre mots finis sur Σ, et $\theta_{\ominus\rho}^{+\alpha}$ et $\theta_{\ominus\rho'}^{+\alpha'}$ les deux Fiffo-transformations ainsi définies. Alors :

1. $\theta_{\ominus\rho}^{+\alpha} \circ \theta_{\ominus\rho'}^{+\alpha'}(w) = (\rho'.\rho)^{\ominus 1}(w.\alpha'.\alpha) = \theta_{\ominus(\rho'.\rho)}^{+(\alpha'.\alpha)}(w)$
2. $\forall n \geq 0,\ \left(\theta_{\ominus\rho}^{+\alpha}\right)^n(w) = \rho^{\ominus n}(w.\alpha^n) = \theta_{\ominus\rho^n}^{+\alpha^n}(w)$

♦

Preuve. Pour le point (1), en utilisant les propriétés de $^{\ominus 1}$ données par le lemme 1.7 (page 27) :

$$\forall u, v, w \in \Sigma^*,\ (v^{\ominus 1}u).w = v^{\ominus 1}(u.w) \qquad \text{et} \qquad w^{\ominus 1}(u^{\ominus 1}v) = (u.w)^{\ominus 1}v$$

nous obtenons :

$$\begin{aligned}
\theta^{+\alpha}_{\ominus\rho} \circ \theta^{+\alpha'}_{\ominus\rho'}(w) &= \rho^{\ominus 1}\left(\rho'^{\ominus 1}\left(w.\alpha'\right).\alpha\right) \\
&= \rho^{\ominus 1}\left(\rho'^{\ominus 1}(w.\alpha'.\alpha)\right) \\
&= (\rho'.\rho)^{\ominus 1}(w.(\alpha'.\alpha)) \\
&= \theta^{+(\alpha'.\alpha)}_{\ominus(\rho'.\rho)}(w)
\end{aligned}$$

La propriété (2) est alors aisément prouvée par induction sur n d'après (1). ♦

Considérons maintenant une séquence de transitions d'un système à file réactif embarqué $S||E$:

$$\langle q_R^1, w_1, q_E^1\rangle \xrightarrow{a_1}_{S||E} \langle q_R^2, w_2, q_E^2\rangle \xrightarrow{a_2}_{S||E} \cdots \xrightarrow{a_n}_{S||E} \langle q_R^{n+1}, w_{n+1}, q_E^{n+1}\rangle$$

Nous montrons dans la propriété suivante que les FIFFO-transformations permettent de modéliser l'effet de la séquence $a_1, \ldots, a_n$ sur w_1 pour obtenir w_{n+1}.

Propriété 5.4

Posons $\alpha = \pi_!(a_1 \ldots a_n)$ et $\rho = \pi_?(a_1 \ldots a_n)$, nous avons : $w_{n+1} = \theta^{+\alpha}_{\ominus\rho}(w_1)$ ♦

Preuve. Par induction sur n, la longueur de la séquence considérée :

- $\mathbf{n = 1.}$ Alors, nous avons trois possibilités, a_1 est : soit un traitement immédiat (il existe un événement e tel que $a_1 = \mathsf{e}$), soit une mémorisation (il existe un événement e pour lequel $a_1 = !\mathsf{e}$) ou soit une consommation (il existe e tel que $a_1 = ?\mathsf{e}$). Dans chacun des cas, voici les définitions de $w_2 = w_{n+1}$ (d'après la définition 2.21) et de $\theta^{+\alpha}_{\ominus\rho}(w_1)$:
  - $\langle q_R^1, w_1, q_E^1\rangle \xrightarrow{\mathsf{e}}_{S||E} \langle q_R^2, w_1, q_E^2\rangle$
    et $\theta^{+\varepsilon}_{\ominus\varepsilon}(w_1) = w_1$,
  - $\langle q_R^1, w_1, q_E^1\rangle \xrightarrow{!\mathsf{e}}_{S||E} \langle q_R^1, w_1.\mathsf{e}, q_E^2\rangle$
    et $\theta^{+\mathsf{e}}_{\ominus\varepsilon} = w_1.\mathsf{e}$,
  - $\langle q_R^1, w_1, q_E^1\rangle \xrightarrow{?\mathsf{e}}_{S||E} \langle q_R^2, \mathsf{e}^{\ominus 1}w_1, q_E^2\rangle$
    et $\theta^{+\varepsilon}_{\ominus\mathsf{e}}(w_1) = \mathsf{e}^{\ominus 1}w$ et $|w_1|_{\mathsf{e}} > 0$ et $\theta^{+\varepsilon}_{\ominus\mathsf{e}} = \mathsf{e}^{\ominus 1}w$.

  Nous obtenons bien la coïncidence entre $\rightarrow_{S||E}$ et $\theta^{+\alpha}_{\ominus\rho}$, par conséquent, la propriété est vérifiée au rang 1.
- **Étape d'induction.** Supposons que la propriété 5.4 soit vérifiée jusqu'au rang k. Alors, l'action suivante, a_{k+1}, est soit un traitement immédiat, soit une mémorisation, soit une consommation. Pour chacun de ces cas, en nommant e l'événement considéré, nous obtenons là encore la correspondance recherchée entre $\rightarrow_{S||E}$ et $\theta^{+\alpha}_{\ominus\rho}$:
  - $\langle q_R^{k+1}, w_{k+1}, q_E^{k+1}\rangle \xrightarrow{\mathsf{e}}_{S||E} \langle q_R^{k+2}, w_{k+1}, q_E^{k+2}\rangle$
    et $\theta^{+\varepsilon}_{\ominus\varepsilon}(w_{k+1}) = w_{k+1} = \theta^{+(\pi_!(a_1 \ldots a_k).\varepsilon)}_{\ominus(\pi_?(a_1 \ldots a_k).\varepsilon)}(w_1) = \theta^{+\pi_!(a_1 \ldots a_{k+1})}_{\ominus\pi_?(a_1 \ldots a_{k+1})}$,
  - $\langle q_R^{k+1}, w_{k+1}, q_E^{k+1}\rangle \xrightarrow{!\mathsf{e}}_{S||E} \langle q_R^{k+1}, w_{k+1}.\mathsf{e}, q_E^{k+2}\rangle$

et $\theta^{+\mathsf{e}}_{\ominus\varepsilon}(w_{k+1}) = w_{k+1}.\mathsf{e} = \theta^{+(\pi_!(a_1 \ldots a_k).\mathsf{e})}_{\ominus(\pi_?(a_1\ldots a_k).\varepsilon)}(w) = \theta^{+\pi_!(a_1\ldots a_{k+1})}_{\ominus\pi_?(a_1\ldots a_{k+1})}$,

– $\langle q_R^{k+1}, w_{k+1}, q_E^{k+1}\rangle \xrightarrow{?\mathsf{e}}_{S||E} \langle q_R^{k+2}, \mathsf{e}^{\ominus 1}w_{k+1}, q_E^{k+2}\rangle$
et $\theta^{+\varepsilon}_{\ominus\mathsf{e}}(w_{k+1}) = \mathsf{e}^{\ominus 1}w_{k+1} = \theta^{+(\pi_!(a_1\ldots a_k).\varepsilon)}_{\ominus(\pi_?(a_1\ldots a_k).\mathsf{e})}(w) = \theta^{+\pi_!(a_1\ldots a_{k+1})}_{\ominus\pi_?(a_1\ldots a_{k+1})}$.

♦

À partir de la modélisation des séquences sous la forme de Fiffo-transformation, il est maintenant possible de s'intéresser à l'invariance des propriétés de file.

5.2.2 Invariance des propriétés de file

Définition 5.5 (Invariance par Fiffo-transformation)
Une propriété $p(a, q_R)$ est *invariante* par itération d'une Fiffo-transformation itérable $\theta^{+\alpha}_{\ominus\rho}$, depuis w, si et seulement si :
- $w \models p(a, q_R)$,
- et $\forall i \geq 1,\ \left(\theta^{+\alpha}_{\ominus\rho}\right)^i \models p(a, q_R)$.

♦

Considérons une Fiffo-transformation itérable $\theta^{+\alpha}_{\ominus\rho}$ où α et ρ sont deux mots finis. Nous prouvons maintenant la décidabilité de l'invariance d'une propriété $p(a, q_R)$ par itération de $\theta^{+\alpha}_{\ominus\rho}$ depuis un mot fini w_0.

Soit $(W_n)_{n\geq 0}$ la suite des mots finis définie par $W_0 = w_0$ et $W_{n+1} = \theta^{+\alpha}_{\ominus\rho}(W_n)$. Puisque $\theta^{+\alpha}_{\ominus\rho}$ est itérable, la suite $(W_n)_{n\geq 0}$ est infinie. Rappelons que nous supposons que les mots de $(W_n)_{n\geq 0}$ sont définis sur l'alphabet[1] de la propriété à vérifier (i.e $\{\mathsf{e}\}$ pour les propriétés $[\mathsf{e} \in]$ et $[\mathsf{e} \notin]$, et $\{\mathsf{e}, \mathsf{e}'\}$ pour $[\mathsf{e} \sqsubseteq \mathsf{e}']$). De plus, par le lemme 5.3, $W_n = \theta^{+\alpha^n}_{\ominus\rho^n}(w_0)$.

Pour que l'invariance de ces propriétés soit établie, il est nécessaire qu'elles soient vérifiées en chacun des mots de $(W_n)_{n\geq 0}$. Dans la suite, nous exprimons des conditions suffisantes sur un nombre fini de mots de $(W_n)_{n\geq 0}$ pour obtenir l'invariance.

Invariance des propriétés d'appartenance $[\mathsf{e} \in]$ et $[\mathsf{e} \notin]$

L'invariance des ces propriétés est aisément vérifiée. Pour la propriété d'appartenance $[\mathsf{e} \in]$, il faut et il suffit que W_0 la vérifie : $W_0 \models [\mathsf{e} \in]$. En effet, puisque $\theta^{+\alpha}_{\ominus\rho}$ est itérable, nous avons $|\alpha|_\mathsf{e} \geq |\rho|_\mathsf{e}$ et par conséquent $|W_{n+1}|_\mathsf{e} \geq |W_n|_\mathsf{e},\ \forall n \geq 0$. Alors, $W_0 \models [\mathsf{e} \in] \ \Rightarrow\ \forall n \geq 0,\ W_n \models [\mathsf{e} \in]$.

Pour une propriété de non appartenance, $[\mathsf{e} \notin]$, il faut que W_0 la vérifie, mais ce n'est plus suffisant. En effet, puisque $\theta^{+\alpha}_{\ominus\rho}$ est itérable, $|\alpha|_\mathsf{e} \geq |\rho|_\mathsf{e}$, et donc $|W_{n+1}|_\mathsf{e} \geq |W_n|_\mathsf{e},\ \forall n \geq 0$. Pour qu'aucun mot de $(W_n)_{n\geq 0}$ ne contienne de e, il faut s'assurer d'une part que $W_0 \models [\mathsf{e} \notin]$ et d'autre part, que $|\alpha|_\mathsf{e} = |\rho|_\mathsf{e}$. Ces conditions sont aussi suffisantes puisqu'alors $|W_n|_\mathsf{e} = 0,\ \forall n \geq 0$.

[1]Ceci n'entraîne aucune perte de généralité puisque cette propriété peut être obtenue par simple projection.

Invariance des propriétés de précédence $[\mathsf{e} \sqsubseteq \mathsf{e}']$

Puisque les mots de $(W_n)_{n\geq 0}$ sont définis sur l'alphabet $\{\mathsf{e}, \mathsf{e}'\}$, la propriété de précédence se transforme de fait en une propriété de préfixage[2] : $W_n \vDash [\mathsf{e} \sqsubseteq \mathsf{e}'] \;\Leftrightarrow\; \mathsf{e} \leq W_n,\ \forall n \geq 0$. Nous montrons maintenant que l'invariance de ce type de propriété dépend des rapports entre $|\alpha|_{\mathsf{e}}$ et $|\rho|_{\mathsf{e}}$ d'un côté, et $|\alpha|_{\mathsf{e}'}$ et $|\rho|_{\mathsf{e}'}$ de l'autre.

Remarque 5.6
La question de l'invariance de $[\mathsf{e} \sqsubseteq \mathsf{e}']$ se pose uniquement si le cycle modélisé par $\theta^{+\alpha}_{\ominus\rho}$ débute par une transition de consommation de e. Par conséquent, nous avons la certitude que $|\rho|_{\mathsf{e}} > 0$. ♦

Par la remarque précédente, il suffit donc, dans un premier temps, de différencier le cas $|\rho|_{\mathsf{e}'} = 0$, du cas $|\rho|_{\mathsf{e}'} > 0$.

$|\rho|_{\mathsf{e}'} = 0$. Dans ce cas, les occurrences de e' qui apparaissent dans w_0 sont intégralement transmises à tous les mots de $(W_n)_{n\geq 0}$. De même, lorsqu'une occurrence de e' est ajoutée par l'application de $\theta^{+\alpha}_{\ominus\rho}$ à $W_k,\ k \geq 0$, elle se retrouve dans tous les mots de $(W_n)_{n\geq 0}$ d'indices supérieurs à k puisqu'elle n'est jamais retirée par $\theta^{+\alpha}_{\ominus\rho}$.

Par conséquent, il convient de distinguer deux cas :

1. $|\alpha|_{\mathsf{e}'} = |\rho|_{\mathsf{e}'}$. Alors, l'application successive de $\theta^{+\alpha}_{\ominus\rho}$ depuis w_0 n'ajoute pas d'occurrence de e' : $\forall k \geq 0,\ |W_k|_{\mathsf{e}'} = |w_0|_{\mathsf{e}'}$. S'il y a une occurrence de e' dans w_0, les occurrences de e qui la précède (s'il y en a) sont peu à peu consommées par les applications successives de $\theta^{+\alpha}_{\ominus\rho}$. Puisque les mots de $(W_n)_{n\geq 0}$ sont finis, il existe un entier $n_0 \geq 0$ fini tel que $\mathsf{e}' \leq W_{n_0}$. Pour éviter ce scénario défavorable, il faut donc que w_0 ne contienne aucune occurrence de e'. Par conséquent :

$$\forall n \geq 0,\ W_n \vDash [\mathsf{e} \sqsubseteq \mathsf{e}'] \;\Leftrightarrow\; |w_0|_{\mathsf{e}'} = 0$$

2. $|\alpha|_{\mathsf{e}'} > |\rho|_{\mathsf{e}'}$. Nous avons l'assurance que W_1 contient une occurrence de e'. De plus, elle est inamovible puisque $|\rho|_{\mathsf{e}'} = 0$. En utilisant le même argument que pour le cas précédent, il existe un rang n_0 fini à partir duquel tous les mots de $(W_n)_{n\geq 0}$ commencent par e'. Par conséquent :

$$\exists n_0 \in \mathbb{N} \text{ t.q. } \forall n \geq n_0,\ W_n \nvDash [\mathsf{e} \sqsubseteq \mathsf{e}']$$

$|\rho|_{\mathsf{e}'} > 0$. Nous montrons que, dans ce cas, l'invariance des propriétés de précédence est décidable après l'examen d'un nombre fini de mots de $(W_n)_{n\geq 0}$. Notre méthode repose sur l'idée que le rapport $\frac{|\alpha|_{\mathsf{e}}}{|\rho|_{\mathsf{e}}}$ caractérise la vitesse de l'augmentation du nombre d'occurrences de e dans les mots de $(W_n)_{n\geq 0}$ à chaque application de $\theta^{+\alpha}_{\ominus\rho}$. Alors, l'invariance de la propriété $[\mathsf{e} \sqsubseteq \mathsf{e}']$ est caractérisée par la différence entre les rapports $\frac{|\alpha|_{\mathsf{e}}}{|\rho|_{\mathsf{e}}}$ et $\frac{|\alpha|_{\mathsf{e}'}}{|\rho|_{\mathsf{e}'}}$: c'est à dire par le fait que le nombre de e ajoutés par $\theta^{+\alpha}_{\ominus\rho}$ est supérieur au nombre de e' ajoutés par la même transformation.

Dans la suite, nous supposons que $\frac{|\alpha|_{\mathsf{e}}}{|\rho|_{\mathsf{e}}} \geq \frac{|\alpha|_{\mathsf{e}'}}{|\rho|_{\mathsf{e}'}}$. Le cas contraire se déduit des résultats suivants en inversant e et e' (voir la remarque 5.11).

---

[2]Rappelons que $u \leq v$ indique que u est un préfixe de v (voir la section 1.2, page 26).

Définition 5.7 (ξ_α et ξ_ρ)
Soient m le plus petit multiple commun à $|\alpha|_{\mathsf{e}'}$ et à $|\rho|_{\mathsf{e}'}$. Nous notons ξ_α et ξ_ρ les deux entiers tels que $\xi_\alpha.|\alpha|_{\mathsf{e}'} = m = \xi_\rho.|\rho|_{\mathsf{e}'}$. ♦

Puisque $\theta_{\ominus\rho}^{+\alpha}$ est itérable, nous avons $|\alpha|_{\mathsf{e}'} \geq |\rho|_{\mathsf{e}'}$ et donc $\xi_\rho \geq \xi_\alpha$. L'idée est la suivante : les mots ρ^{ξ_ρ} et α^{ξ_α} contiennent le même nombre d'occurrences de e', et par conséquent, en utilisant la commutativité de $^{\ominus 1}$ (point (1), lemme 1.7, page 27) :

$$\begin{aligned}\rho^{\ominus\xi_\rho}(\alpha^{\xi_\rho}) &= \left(\mathsf{e}^{\ominus\xi_\rho.|\rho|_\mathsf{e}}\mathsf{e}'^{\ominus\xi_\rho.|\rho|_{\mathsf{e}'}}(\alpha^{\xi_\alpha})\right).\alpha^{(\xi_\rho-\xi_\alpha)}\\ &= \left(\mathsf{e}^{\ominus\xi_\rho.|\rho|_\mathsf{e}}\left(\mathsf{e}^{\xi_\alpha.|\alpha|_\mathsf{e}}\right)\right).\alpha^{(\xi_\rho-\xi_\alpha)}\end{aligned}$$

Alors, suivant les valeurs relatives de $\xi_\rho.|\rho|_\mathsf{e}$ et de $\xi_\alpha.|\alpha|_\mathsf{e}$, nous montrons que les mots obtenus par applications successives de $\left(\theta_{\ominus\rho}^{+\alpha}\right)^{\xi_\rho}$ partagent une même structure, ce qui nous permet d'établir une condition suffisante pour l'itérabilité d'une séquence.

Cependant, la méthode que nous venons d'exquisser se fonde sur le fait que les mots de $(W_n)_{n\geq 0}$ ont une structure basée sur α. Or w_0 ne remplit généralement pas cette condition, c'est pourquoi, il est tout d'abord nécessaire d'appliquer $\theta_{\ominus\rho}^{+\alpha}$ un certain nombre de fois pour se ramener dans ce cas. Après $\xi_0 = \max\left\{\left\lceil\frac{|w_0|_\mathsf{e}}{|\rho|_\mathsf{e}}\right\rceil, \left\lceil\frac{|w_0|_{\mathsf{e}'}}{|\rho|_{\mathsf{e}'}}\right\rceil\right\}$ applications successives de $\theta_{\ominus\rho}^{+\alpha}$ à $W_0 = w_0$, le mot obtenu, W_{ξ_0}, ne contient plus aucun symbole qui constituait initialement w_0. Alors, en posant $\mu_0 = \xi_0.|\rho|_\mathsf{e} - |w_0|_\mathsf{e}$ et $\mu_0' = \xi_0.|\rho|_{\mathsf{e}'} - |w_0|_{\mathsf{e}'}$ nous exprimons les mots de $(W_n)_{n\geq 0}$ en fonction de α :

Propriété 5.8
Quel que soit $n \geq 0$,

$$W_{(\xi_0+n.\xi_\rho)} = \left(\mathsf{e}^{\ominus(\mu_0+n.\xi_\rho.|\rho|_\mathsf{e})}.\mathsf{e}^{(n.\xi_\alpha.|\alpha|_\mathsf{e})}\right).\left(\mathsf{e}'^{\ominus\mu_0'}.\alpha^{\xi_0}\right).\alpha^{n.(\xi_\rho-\xi_\alpha)} \tag{5.5}$$

♦

Il est possible que pour les petites valeurs de n, $\mu_0 + n.\xi_\rho.|\rho|_\mathsf{e} > n.\xi_\alpha.|\alpha|_\mathsf{e}$. Dans ce cas, l'équation (5.5) s'écrit :

$$W_{(\xi_0+n.\xi_\rho)} = \mathsf{e}^{\ominus(\mu_0+n.(\xi_\rho.|\rho|_\mathsf{e}-\xi_\alpha.|\alpha|_\mathsf{e}))}.\left(\mathsf{e}'^{\ominus\mu_0'}.\alpha^{\xi_0}\right).\alpha^{n.(\xi_\rho-\xi_\alpha)}$$

Preuve (Propriété 5.8). Nous prouvons cette propriété par induction sur n :
– **n = 0.** Alors :

$$\begin{aligned}W_{\xi_0} &= \rho^{\ominus\xi_0}.\left(W_0.\alpha^{\xi_0}\right)\\ &= \mathsf{e}^{\ominus\xi_0.|\rho|_\mathsf{e}}.\mathsf{e}'^{\ominus\xi_0.|\rho|_{\mathsf{e}'}}.\left(W_0.\alpha^{\xi_0}\right)\\ &= \mathsf{e}^{\ominus\xi_0.|\rho|_\mathsf{e}}.\mathsf{e}^{|W_0|_\mathsf{e}}.\mathsf{e}'^{\ominus\xi_0.|\rho|_{\mathsf{e}'}}\mathsf{e}'^{|W_0|_{\mathsf{e}'}}.\alpha^{\xi_0}\\ &= \mathsf{e}^{\ominus\mu_0}.\mathsf{e}'^{\ominus\mu_0'}.\alpha^{\xi_0}\end{aligned}$$

– **Étape d'induction.** Supposons que la propriété 5.8 soit vérifiée jusqu'au rang n.

$$\begin{aligned}
W_{(\xi_0+(n+1).\xi_\rho)} &= \rho^{\ominus\xi_\rho}\left(W_{(\xi_0+n.\xi_\rho)}.\alpha^{\xi_\rho}\right)\\
&= \mathsf{e}^{\ominus\xi_\rho.|\rho|_\mathsf{e}}.\mathsf{e}'^{\ominus\xi_\rho.|\rho|_{\mathsf{e}'}}.\left[\left(\mathsf{e}^{\ominus(\mu_0+n.\xi_\rho.|\rho|_\mathsf{e})}.\mathsf{e}^{(n.\xi_\alpha.|\alpha|_\mathsf{e})}\right).\left(\mathsf{e}'^{\ominus\mu'_0}.\alpha^{\xi_0}\right).\alpha^{n.(\xi_\rho-\xi_\alpha)}.\alpha^{\xi_\rho}\right]\\
&= \left(\mathsf{e}^{\ominus(\mu_0+(n+1).\xi_\rho.|\rho|_\mathsf{e})}.\mathsf{e}^{(n.\xi_\alpha.|\alpha|_\mathsf{e})}\right).\left(\mathsf{e}'^{\ominus(\mu'_0+\xi_\rho.|\rho|_{\mathsf{e}'})}.\alpha^{(\xi_0+\xi_\alpha)}\right).\alpha^{(n+1).(\xi_\rho-\xi_\alpha)}\\
&= \left(\mathsf{e}^{\ominus(\mu_0+(n+1).\xi_\rho.|\rho|_\mathsf{e})}.\mathsf{e}^{((n+1).\xi_\alpha.|\alpha|_\mathsf{e})}\right).\left(\mathsf{e}'^{\ominus\mu'_0}.\alpha^{\xi_0}\right).\alpha^{(n+1).(\xi_\rho-\xi_\alpha)}
\end{aligned}$$

♦

Rappelons que nous supposons que $\frac{|\alpha|_\mathsf{e}}{|\rho|_\mathsf{e}} \geq \frac{|\alpha|_{\mathsf{e}'}}{|\rho|_{\mathsf{e}'}}$. Nous distinguons maintenant les deux sous-cas : $\frac{|\alpha|_\mathsf{e}}{|\rho|_\mathsf{e}} = \frac{|\alpha|_{\mathsf{e}'}}{|\rho|_{\mathsf{e}'}}$ et $\frac{|\alpha|_\mathsf{e}}{|\rho|_\mathsf{e}} > \frac{|\alpha|_{\mathsf{e}'}}{|\rho|_{\mathsf{e}'}}$ qui, comme nous le montrons, correspondent à des comportements bien distincts de la fiffo transformation $\theta^{+\alpha}_{\ominus\rho}$. En effet, en suivant toujours l'intuition que $\frac{|\alpha|_\mathsf{e}}{|\rho|_\mathsf{e}}$ représente la vitesse d'accumulation des occurrences de e dans les mots de $(W_n)_{n\geq 0}$, dans le premier sous-cas, e et e' s'accumulent à la même vitesse, alors que dans le second, e s'accumule "plus vite" que e'.

$\boxed{\frac{|\alpha|_\mathsf{e}}{|\rho|_\mathsf{e}} = \frac{|\alpha|_{\mathsf{e}'}}{|\rho|_{\mathsf{e}'}}}$ Prenons pour exemple la FIFFO-transformation $\theta^{+\alpha}_{\ominus\rho}$ définie par $\alpha = \mathsf{eee'ee'e'}$ et $\rho = \mathsf{ee'}$. Depuis $w_0 = \mathsf{ee'}$, les premiers mots obtenus par applications successives de $\theta^{+\alpha}_{\ominus\rho}$ sont :

$$\mathsf{ee'} \xrightarrow{\theta^{+\alpha}_{\ominus\rho}} (\mathsf{eee'ee'e'}) \xrightarrow{\theta^{+\alpha}_{\ominus\rho}} \mathsf{eee'e'}.(\mathsf{eee'ee'e'}) \xrightarrow{\theta^{+\alpha}_{\ominus\rho}} \mathsf{ee'}.(\mathsf{eee'ee'e'})^2$$

$$\underbrace{\mathsf{ee'}}.(\alpha)^0 \qquad \underbrace{\quad}.(\alpha)^1 \qquad \underbrace{\mathsf{eee'e'}}.(\alpha)^1 \qquad \underbrace{\mathsf{ee'}}.(\alpha)^2$$

La seconde ligne représente les mots de $(W_n)_{n\geq 0}$ de la première ligne factorisés par α. Nous remarquons que les mots de $(W_n)_{n\geq 0}$ admettent un nombre fini de préfixes différents (identifiés sur le schéma par les accolades) et particulièrement, tous les $\xi_\rho = 3$ mots, le même préfixe apparaît.

En effet, dans ce cas, $\xi_\rho.|\rho|_\mathsf{e} = \xi_\alpha.|\alpha|_\mathsf{e}$, donc l'équation (5.5) se transforme en :

$$\forall n \geq 0,\ W_{(\xi_0+n.\xi_\rho)} = \left(\mathsf{e}^{\ominus\mu_0}.\mathsf{e}'^{\ominus\mu'_0}.\alpha^{\xi_0}\right).\alpha^{n.(\xi_\rho-\xi_\alpha)} \tag{5.6}$$

Il reste maintenant à prouver qu'il suffit d'examiner la validité de $[\mathsf{e} \sqsubseteq \mathsf{e}']$ pour les mots W_0 à $W_{\xi_0+\xi_\rho-1}$ pour obtenir l'invariance de cette propriété. Pour cela, il suffit de prouver le résultat suivant :

Lemme 5.9
$\forall n \geq 0,\ \forall k \in \{1, \ldots, \xi_\rho - 1\},\ \mathsf{e} \leq W_{(\xi_0+n.\xi_\rho+k)} \ \Rightarrow\ \mathsf{e} \leq W_{(\xi_0+(n+1).\xi_\rho+k)}$ ♦

Preuve.

$$\begin{aligned}
W_{(\xi_0+(n+1).\xi_\rho+k)} &= \rho^{\ominus k}\left[\left(\mathsf{e}^{\ominus\mu_0}.\mathsf{e}'^{\ominus\mu'_0}.\alpha^{\xi_0}\right).\alpha^{(n+1).(\xi_\rho-\xi_\alpha)}.\alpha^k\right]\\
&= \rho^{\ominus k}\left[\left(\mathsf{e}^{\ominus\mu_0}.\mathsf{e}'^{\ominus\mu'_0}.\alpha^{\xi_0}\right).\alpha^{n.(\xi_\rho-\xi_\alpha)}.\alpha^k\right].\alpha^{(\xi_\rho-\xi_\alpha)}\\
&= W_{(\xi_0+n.\xi_\rho+k)}.\alpha^{(\xi_\rho-\xi_\alpha)}
\end{aligned}$$

Alors, puisque $\mathsf{e} \leq W_{(\xi_0+n.\xi_\rho+k)}$, il vient $\mathsf{e} \leq W_{(\xi_0+(n+1).\xi_\rho+k)}$ ♦

Par conséquent, il est suffisant que $W_0 \models [\mathsf{e} \sqsubseteq \mathsf{e}'], \ldots, W_{(\xi_0+\xi_\rho-1)} \models [\mathsf{e} \sqsubseteq \mathsf{e}']$ pour que, quel que soit $n \geq 0$, $W_n \models [\mathsf{e} \sqsubseteq \mathsf{e}']$.
Notons que dans le cas particulier où $|\alpha|_\mathsf{e} = |\rho|_\mathsf{e}$ et $|\alpha|_{\mathsf{e}'} = |\rho|_{\mathsf{e}'}$, nous avons $\xi_\alpha = \xi_\rho = 1$ et l'expression (5.6) devient alors :

$$\forall n \geq 0,\ W_{(\xi_0+n.\xi_\rho)} = \mathsf{e}^{\ominus\mu_0}.\mathsf{e}'^{\ominus\mu'_0}.\alpha^{\xi_0}$$

Dans ce cas, $(W_n)_{n\geq 0}$ contient un nombre fini de mots différents puisque le nombre d'occurrences de e et de e' dans ces mots est invariant par $\theta^{+\alpha}_{\ominus\rho}$.
$\boxed{\frac{|\alpha|_\mathsf{e}}{|\rho|_\mathsf{e}} > \frac{|\alpha|_{\mathsf{e}'}}{|\rho|_{\mathsf{e}'}}}$ Considérons l'exemple de la FIFFO-transformation obtenue en prenant $\alpha = \mathsf{eee}'\mathsf{ee}'$ et $\rho = \mathsf{ee}'$ dont les premières applications depuis $w_0 = \mathsf{ee}'$ sont :

$$\underbrace{\mathsf{e}}\,\mathsf{e}' \xrightarrow{\theta^{+\alpha}_{\ominus\rho}} \underbrace{\mathsf{ee}}\,\mathsf{e}'\mathsf{ee}' \xrightarrow{\theta^{+\alpha}_{\ominus\rho}} \underbrace{\mathsf{ee}}\,\mathsf{e}'\mathsf{eee}'\mathsf{ee}' \xrightarrow{\theta^{+\alpha}_{\ominus\rho}} \underbrace{\mathsf{eee}}\,\mathsf{e}'\mathsf{ee}'\mathsf{eee}'\mathsf{ee}'$$

Il apparaît que des occurrences de e s'accumulent en tête des mots de $(W_n)_{n\geq 0}$, et plus précisement, le nombre de e qui figure en tête des mots de $(W_n)_{n\geq 0}$ est strictement croissant tous les $\xi_\rho = 2$ applications consécutives de $\theta^{+\alpha}_{\ominus\rho}$. Puisque $\frac{|\alpha|_{\mathsf{e}'}}{|\rho|_{\mathsf{e}'}} = \frac{\xi_\rho}{\xi_\alpha}$ nous avons $\xi_\alpha.|\alpha|_\mathsf{e} > \xi_\rho.|\rho|_\mathsf{e}$ et l'équation (5.5) s'écrit dans ce cas :

$$\forall n \geq 0,\ W_{(\xi_0+n.\xi_\rho)} = \mathsf{e}^{(n.(\xi_\alpha.|\alpha|_\mathsf{e}-\xi_\rho.|\rho|_\mathsf{e})-\mu_0)}.\left(\mathsf{e}'^{\ominus\mu'_0}.\alpha^{\xi_0}\right).\alpha^{n.(\xi_\rho-\xi_\alpha)} \tag{5.7}$$

Nous notons que le nombre de e qui s'accumulent en tête des mots de $(W_{(\xi_0+n.\xi_\rho)})_{n\geq 0}$ est strictement croissant. Soit n_0 l'entier tel que le nombre d'occurrences de e en préfixe de $W_{(\xi_0+n_0.\xi_\rho)}$ est supérieur à $\xi_\rho.|\rho|_\mathsf{e}$. D'après (5.7), n_0 est donc défini par $n_0.(\xi_\alpha.|\alpha|_\mathsf{e} - \xi_\rho.|\rho|_\mathsf{e}) - \mu_0 > \xi_\rho.|\rho|_\mathsf{e}$, c'est à dire :

$$n_0 = \left\lfloor \frac{(\xi_0+\xi_\rho).|\rho|_\mathsf{e} - |w_0|_\mathsf{e}}{\xi_\alpha.|\alpha|_\mathsf{e} - \xi_\rho.|\rho|_\mathsf{e}} \right\rfloor + 1 \tag{5.8}$$

À partir du rang n_0, tous les mots de $(W_n)_{n\geq 0}$ tels que $n \geq \xi_0 + n_0.\xi_\rho$ commencent par (au moins) une occurrence de e. Cela provient du résultat suivant :

Lemme 5.10
$\forall n \geq n_0,\ \forall k \in \{1,\ldots,\xi_\rho - 1\},\ \mathsf{e}^{\xi_\rho.|\rho|_\mathsf{e}} \leq W_{(\xi_0+n.\xi_\rho)} \ \Rightarrow\ \mathsf{e} \leq W_{(\xi_0+n.\xi_\rho+k)}$ ♦

Preuve. Il suffit de noter que $k.|\rho|_\mathsf{e} < \xi_\rho.|\rho|_\mathsf{e}$ quel que soit $k \in \{1,\ldots,\xi_\rho-1\}$. ♦
Alors, puisque le nombre de e en tête de mots de $(W_{(\xi_0+n.\xi_\rho)})_{n\geq 0}$ est strictement croissant, nous en déduisons qu'il est suffisant que W_0 à $W_{(\xi_0+n_0.\xi_\rho)}$ vérifient $[\mathsf{e} \sqsubseteq \mathsf{e}']$ pour qu'il en soit de même pour tous les mots de $(W_n)_{n\geq 0}$.

Remarque 5.11
Nous venons de montrer que lorsque $\frac{|\alpha|_\mathsf{e}}{|\rho|_\mathsf{e}} > \frac{|\alpha|_{\mathsf{e}'}}{|\rho|_{\mathsf{e}'}}$ des occurrences de e s'accumulent en tête des mots

de $(W_n)_{n\geq 0}$. Nous en déduisons que dans le cas contraire $\frac{|\alpha|_{\mathsf{e}}}{|\rho|_{\mathsf{e}}} < \frac{|\alpha|_{\mathsf{e}'}}{|\rho|_{\mathsf{e}'}}$, ce sont des occurrences de e' qui remplacent celles de e. Par conséquent, dans ce dernier cas, la propriété $[\mathsf{e} \sqsubseteq \mathsf{e}']$ est toujours invalidée à partir d'un rang fini dans $(W_n)_{n\geq 0}$. ♦

En conclusion, nous avons prouvé le théorème suivant :

Théorème 5.12
Soit :

$$\sigma = \langle q_R^1, w_1, q_E^1\rangle \xrightarrow{a_1}_{S||E} \langle q_R^2, w_2, q_E^2\rangle \xrightarrow{a_2}_{S||E} \ldots \xrightarrow{a_n}_{S||E} \langle q_R^{n+1}, w_{n+1}, q_E^{n+1}\rangle \quad \text{avec } q_R^1 = q_R^{n+1}$$

une séquence d'un système à file réactif embarqué. L'itérabilité de σ depuis $\langle q_R^1, w_1, q_E^1\rangle$ est décidable si l'itérabilité de $\pi_E(\sigma)$ est décidable dans E. ♦

Remarque 5.13
Lorsque l'environnement E du système à file réactif considéré est un système de transitions étiqueté fini, la condition nécessaire **(CN1)**, pour l'itérabilité de σ, est vérifiée dès que $q_E^1 = q_E^{n+1}$. ♦

5.3 Implantation du test : l'outil TeTU

Nous avons implanté notre méthode de test dans l'outil TeTU que nous décrivons maintenant. Nous ne nous étendons pas sur la partie qui concerne le calcul de l'arbre d'accessibilité du système considéré puisqu'il s'agit tout simplement de suivre la définition 2.21 (page 51).

Dans cette section, nous mettons l'accent sur l'algorithme de calcul de la relation $\mathcal{I}$ telle qu'elle est décrite par l'équation (5.4) (page 112).

L'algorithme qui découle immédiatement de la définition de $\mathcal{I}$ de l'équation (5.4) (page 112) n'est pas optimal. En effet, pour la séquence σ suivante :

$$\sigma = \langle q_R^1, w_1, q_E^1\rangle \xrightarrow{a_1}_{S||E} \langle q_R^2, w_2, q_E^2\rangle \xrightarrow{a_2}_{S||E} \ldots \xrightarrow{a_n}_{S||E} \langle q_R^{n+1}, w_{n+1}, q_E^{n+1}\rangle$$

il s'agit de considérer chacune des permutations circulaires :

$$\begin{aligned}
&\sigma_1 = q_R^1 \xrightarrow{a_1}_R q_R^2 \xrightarrow{a_2}_R \ldots q_R^n \xrightarrow{a_n}_R q_R^{n+1} = q_R^1\\
&\sigma_2 = q_R^2 \xrightarrow{a_2}_R \ldots q_R^n \xrightarrow{a_n}_R q_R^{n+1} = q_R^1 \xrightarrow{a_1}_R q_R^2\\
&\ldots\\
&\sigma_n = q_R^n \xrightarrow{a_n}_R q_R^{n+1} = q_R^1 \xrightarrow{a_1}_R \ldots q_R^{n-1} \xrightarrow{a_{n-1}}_R q_E^n
\end{aligned}$$

et de vérifier que les conditions suffisantes présentées en section 5.2.2 (page 114) sont bien remplies :

$$\forall m \geq 0,\ \sigma_1^m(w_1) \vDash p(a_1, q_R^1) \wedge \ldots \wedge \sigma_n^m(w_n) \vDash p(a_n, q_R^n)$$

Cependant, si l'algorithme de vérification est implanté tel quel, chaque transition est exécutée plusieurs fois lorsque σ_1 à σ_n sont franchies une fois. De même les contenus de files sont calculés plusieurs fois. Par exemple, pour le premier franchissement des séquences :

$$\begin{array}{l}
\sigma_1 : \ \langle q_R^1, w_1, q_E^1\rangle \xrightarrow{a_1}_{S||E} \langle q_R^2, w_2, q_E^2\rangle \xrightarrow{a_2}_{S||E} \dots \xrightarrow{a_n}_{S||E} \langle q_R^{n+1}, w_{n+1}, q_E^{n+1}\rangle \\
\sigma_2 : \ \langle q_R^2, w_2, q_E^2\rangle \xrightarrow{a_2}_{S||E} \dots \xrightarrow{a_n}_{S||E} \langle q_R^1, w_{n+1}, q_E^{n+1}\rangle \xrightarrow{a_1}_{S||E} \langle q_R^2, w_{n+2}, q_E^{n+2}\rangle \\
\dots \\
\sigma_n : \ \langle q_R^n, w_n, q_E^n\rangle \xrightarrow{a_n}_{S||E} \langle q_R^1, w_{n+1}, q_E^{n+1}\rangle \xrightarrow{a_1}_{S||E} \dots \xrightarrow{a_{n-1}}_{S||E} \langle q_R^n, w_{2n}, q_E^{2n}\rangle
\end{array}$$

la transition $\langle q_R^n, w_n, q_E^n\rangle \xrightarrow{a_n}_{S||E} \langle q_R^1, w_{n+1}, q_E^{n+1}\rangle$ est exécutée n fois, donnant lieu à n calculs de w_{n+1} et q_E^{n+1}.

Toutefois, il est possible d'implémenter l'algorithme de vérification de $\mathcal{I}$ de manière nettement plus efficace en franchissant chaque transition une et une seule fois. En effet, il suffit de remarquer que :

$$\forall i \in \{1, \dots, n\},\ \forall m \geq 0,\ \sigma_i^m(w_i) = w_{i+n.m}$$

Nous en déduisons donc la définition suivante pour $\mathcal{I}$ (s et s' sont deux noeuds de l'arbre d'accessibilité de $S||E$) :

$$\mathcal{I}(s, s') \ \Leftrightarrow\ \forall m \geq 0,\ w_{(1+m.n)} \vDash p(a_1, q_R^1) \ \wedge\ \dots\ \wedge w_{(n+m.n)} \vDash p(a_n, q_R^n) \tag{5.9}$$

en remplacement de celle donnée par l'équation (5.4) (page 112).

Notre algorithme consiste donc à développer l'exécution du système à file réactif embarqué à partir de $\langle q_R^1, w_1, q_E^1\rangle$ en vérifiant que la file obtenue dans chaque configuration vérifie bien la propriété associée : $w_{(i+m.n)} \vDash p(a_i, q_R^i)$ avec $i \in \{1, \dots, n\}$ et $m \geq 0$. S'il existe une valuation de i et m pour laquelle la propriété n'est pas validée, alors la séquence n'est pas itérable. Quant au problème de savoir si l'exécution a été suffisament développée, il se ramène à savoir si la séquence considérée a été itérée un nombre suffisant de fois, comme décrit en section 5.2.2.

Le tableau suivant résume les résultats que nous avons montré dans les deux premiers chapitres. Un **D** signifie que la propriété est décidable, et à l'inverse, un **I** indique que la propriété est indécidable. Pour chaque résultat, nous indiquons la classe de système à file réactif embarqué, ainsi que la classe d'environnement concernées. Les points d'interogations "?" qui figurent dans ce tableau indique que le problème correspondant est ouvert.

<table>
<tr><th rowspan="2"></th><th rowspan="2">$\lvert\Sigma_M\rvert$</th><th rowspan="2">(C)</th><th rowspan="2">(R)</th><th rowspan="2">(T)</th><th rowspan="2">(B)</th><th colspan="4">Model-checking</th></tr>
<tr><th>Ltl</th><th>Ctl</th><th>(EF)</th><th>(EG)</th></tr>
<tr><td rowspan="2">SFR Embarqués

Reset SFR Embarqués</td><td>≤ 1</td><td colspan="8">D Env. rationnel</td></tr>
<tr><td>≥ 2</td><td colspan="8">I Env. périodique</td></tr>
<tr><td rowspan="3">Lossy SFR Embarqués</td><td>2</td><td colspan="3" rowspan="3">D
Env. bien structuré</td><td>?</td><td>?</td><td>?</td><td colspan="2" rowspan="3">D Env. bien structuré</td></tr>
<tr><td>3</td><td colspan="2" rowspan="2">I Env. ultimement périodique</td><td>?</td></tr>
<tr><td>≥ 4</td><td></td></tr>
</table>

Nous n'avons pas répété les résultats pour les Lossy SFR Embarqués avec 1 compteurs puisqu'ils découlent directement de ceux pour les SFR Embarqués avec 1 compteurs (sans perte). En effet, tout Lossy SFR Embarqué peut être simulé par un SFR Embarqué, l'inverse n'étant généralement pas vrai.

Les problèmes de la bornitude et du model-checking de Ltl pour les Lossy SFR Embarqués avec 2 événements mémorisables d'un côté, et le problème du model-checking de Ctl pour les Lossy SFR Embarqués avec 2 et 3 événements mémorisables sont ouverts. À notre connaissance, ces problèmes sont actuellement ouverts pour les Lossy machines à compteurs.

L'indécidabilité de la bornitude (B) pour les classes générales de systèmes à file réactifs embarqués est problématique. Cette propriété est, en effet, importante pour choisir une technique de vérification appropriée, mais elle s'avère même cruciale lorsque le système modélisé est implanté, puisque toute mémoire physique est de taille bornée. Dans le dernier chapitre, nous avons donc présenté une procédure de test qui détecte les cycles des exécutions d'un SFR Embarqué, qui rendent la file de mémorisation non bornée. Ainsi, il est possible, dans certains cas, de savoir lorsque la file n'est pas bornée.

Maintenant que nous avons analysé le comportement des systèmes à file réactifs embarqués, nous nous interessons dans la partie suivante à la spécification de l'environnement, et à la vérification des systèmes alors obtenus.

Troisième partie

Application à la vérification de programmes temps-réel Electre.

Tout au long de la partie précédente, nous avons étudié l'expressivité du modèle obtenu en plongeant un système à file réactif dans un environnement, c'est à dire un système de transitions étiqueté. Nous avons montré que le modèle ainsi obtenu a (généralement) la puissance des machines de Turing. Ceci est particulièrement vrai pour des environnements très simples tels que des environnements périodiques. Nous avons donc introduit un test pour le problème de la bornitude.

Dans le chapitre 6, nous introduisons la *spécification temporelle* des environnements au travers d'informations telles que la durée des modules et la période d'occurences des événements. Nous obtenons ainsi les SFR Hybrides Linéaires. Il est clair que tout environnement périodique peut être exprimé par des spécifications temporelles, donc nous adaptons notre test à ce nouveau contexte.

Enfin, même lorsqu'un SFR Embarqué, ou Hybride Linéaire, est borné, sa vérification est souvent impossible en pratique à cause de sa taille. Nous introduisons dans le chapitre 7 une méthode de réduction du modèle de vérification.

Chapitre 6

Systèmes à file réactifs hybrides embarqués

Dans le chapitre 2 (page 33) nous montrons comment le langage ELECTRE peut être utilisé pour spécifier le comportement des applications réactives, mais nous mettons aussi en évidence la lacune de spécification inhérente au langage lui-même : la modélisation du mode d'occurrence des événements manque. Dans le cadre de notre étude, et jusqu'à ce point, cette lacune est comblée en spécifiant les séquences d'événements, produites par l'environnement du système modélisé, sous la forme d'un système de transitions étiqueté. Cependant, les environnements des applications temps-réel sont rarement donnés directement sous cette forme, mais plutôt par des informations temporelles quantitatives, telles que la durée de tâches, ou la fréquence d'occurrence des événements. Ils sont alors naturellement décrits par des *systèmes hybrides* [6, 7, 58] (définition 6.5, page 135), dont la sémantique est, elle, donnée par un système de transitions étiqueté, rejoignant ainsi notre définition précédente des environnements (définition 2.20, page 51).

Deux propositions de spécification temporelle des programmes ELECTRE ont déjà été formulées. Dans [89], Rusu étudie les systèmes à file réactifs sans mémorisation, pour lesquels il propose un formalisme de spécification de l'environnement. La seconde étude, de Boisieau [16], est consacrée à la vérification d'une application de contrôle d'un réacteur d'avion. Cette fois-ci, les événements à mémorisation unique sont considérés. Naturellement, notre langage $\mathcal{H}$-ELECTRE, pour la spécification temporelle de l'environnement, s'inspire de ces deux propositions, mais nous considérons en plus les événements à mémorisation multiple. Notre but est de tester la non-bornitude pour des systèmes à file réactifs plongés dans des environnement hybrides.

Nous considérons donc, dans ce chapitre, la spécification et la modélisation d'environnements traduits par des informations temporelles, sur les modules et les événements des programmes ELECTRE. À cet effet, la section 6.1 décrit notre langage pour la spécification temporelle des programmes ELECTRE : $\mathcal{H}$-ELECTRE. Puis en section 6.2, nous montrons comment construire un *système à file réactif hybride linéaire* à partir d'une spécification $\mathcal{H}$-ELECTRE. Enfin, le problème de la bornitude (B) n'est, bien entendu, pas décidable pour les programmes $\mathcal{H}$-ELECTRE : tout environnement périodique (pour lesquels (B) est indécidable, voir le théorème 3.6, page 76) peut être décrit en $\mathcal{H}$-ELECTRE. Par conséquent, en section 6.3, nous adaptons notre test de non-bornitude (présenté dans le chapitre 5, page 107) pour prendre en compte les environnements hybrides.

6.1 Temporisation du langage Electre : $\mathcal{H}$-Electre

Cette section présente le langage $\mathcal{H}$-ELECTRE pour la spécification temps-réel des programmes ELECTRE. Les choix de spécifications s'inspirent fortement des études de Rusu [89] et Boisieau [16].

6.1.1 Besoins et choix de spécification

Nous considérons un ensemble fini de modules, noté $\mathcal{M}$, et un ensemble fini d'événements, noté $\mathcal{E}$. La spécification temporelle à proprement parler se fait naturellement au travers de la durée des modules et de la fréquence d'occurrence des événements, puisque ce sont les éléments qui rythment l'évolution d'une application ELECTRE. Cependant, ce ne sont pas les seuls éléments que nous souhaitons spécifier. Nous cherchons en effet à définir un formalisme à la fois souple et expressif pour la modélisation d'applications temps-réel. Par exemple, tous les modules n'ont pas forcément la même priorité. Nous introduisons, dans la suite de cette section, tous les éléments qui constituent une spécification $\mathcal{H}$-ELECTRE. Les données temporelles sont exprimées dans une unité temporelle abstraite, nommée *unité de temps (u.t)*.

Durée et ordonnancement des modules

Durée. Un module est spécifié au travers de sa durée. Rappelons que la définition d'ELECTRE impose que cette durée soit *finie et non nulle*. Pour un module `M`, nous spécifions donc sa durée minimale, $\underline{d}(\mathtt{M})$ et sa durée maximale $\overline{d}(\mathtt{M})$. Bien entendu, ces deux valeurs vérifient : $\underline{d}(\mathtt{M}), \overline{d}(\mathtt{M}) \in \mathbb{Q}$ et $0 < \underline{d}(\mathtt{M}) \leq \overline{d}(\mathtt{M}) < \infty$.

Placement. Une application temps-réel est souvent distribuée sur plusieurs calculateurs. Par conséquent, tous les modules ne se partagent pas le même processeur, et cela influe sur leur vitesse d'exécution. Nous considérons donc un ensemble de *places* (ou *processeurs*), noté $\mathcal{P}$, sur lesquels les modules s'exécutent.

Nous considérons que le placement d'un module est statique : il s'exécute toujours sur le même processeur, et la fonction Pl attribue à `M` une place $p \in \mathcal{P}$.

Priorité. De plus, tous les modules qui s'exécutent sur une même place n'ont pas la même importance, suivant les sollicitations auxquelles ils répondent. Par exemple, le traitement d'une alarme est plus urgent que les autres activités. Nous utilisons la fonction Pr pour associer une *priorité* aux modules. Cette fonction vérifie, pour tout module `M`, $Pr(\mathtt{M}) \in \mathbb{N}_{>0}$. Plus $Pr(\mathtt{M})$ est grand, plus la priorité de ce module est élevée.

Il faut maintenant spécifier comment cette priorité intervient dans l'exécution des modules. Comme cela est habituel dans la majorité des système d'exploitation, plus un module est prioritaire, plus il bénéficie d'une part élevée de l'activité du processeur sur lequel il s'exécute. Nous jugeons suffisant ici de considérer la vitesse d'exécution moyenne (fixe) des modules. Cela suppose que l'ordonnanceur du système effectue correctement sa tâche, ce qui est un prérecquis essentiel au bon fonctionnement de toute application temps-réel. Nous proposons deux stratégies :

– (Ppf) : **Placement et Part Fixe.** Nous considérons le placement des modules, et chaque module s'exécute sur son processeur avec une part fixe du processeur qui lui est allouée. Cette part est d'autant plus grande que la priorité du module est élevée.
  Par exemple, $\mathtt{M}$, $\mathtt{M}'$ et $\mathtt{M}''$ partagent une même place, et $Pr(\mathtt{M}) = Pr(\mathtt{M}'') = 2$ alors que $Pr(\mathtt{M}') = 1$. Alors, $\mathtt{M}$ et $\mathtt{M}''$ bénéficient des $\frac{2}{5}$ du processeur, alors que $\mathtt{M}'$ n'en possède qu'un cinquième : $\frac{1}{5}$.
– (Ppmf) : **Placement et Part Maximale Fixe.** Cette sémantique est une variante de la précédente, où seuls les modules de priorité maximale se partagent le processeur à égalité, alors que les autres modules, moins prioritaires, ne s'exécutent pas. En considérant le même exemple que précédement, seuls $\mathtt{M}$ et $\mathtt{M}''$ s'exécutent en bénéficiant de la moitié du processeur chacun.

Mode d'occurrence des événements

Un programme Electre décrit les réactions de l'application aux occurrences des événements. Les dates d'occurrence de ceux-ci sont alors décrites lors de la spécification de l'environnement de l'application. Nous introduisons, dans cette section, les éléments qui permettent de décrire les occurrences des éváments.

Période d'occurrence. Pour chaque événement $\mathtt{e}$, nous spécifions sa période d'occurrence sous la forme d'un *délai minimal*, noté $\underline{d}(\mathtt{e})$, et d'un *délai maximal*, noté $\overline{d}(\mathtt{e})$, séparant deux occurrences successives de $\mathtt{e}$. Cette spécification laisse entendre que nous ne considérons que des événements qui surviennent régulièrement, toutes les $\underline{d}(\mathtt{e})$ à $\overline{d}(\mathtt{e})$ unités de temps. Ce n'est pas le cas puisque l'interprétation que nous faisons de ces deux valeurs dépend d'une autre spécification.

Obligation/Opportunité d'occurrence. Un événement n'a en effet pas nécessairement l'obligation de survenir. Ainsi, nous proposons deux sémantiques :

– (Eds) : **Événement Doit Survenir.** Dans ce cas, il se produit systématiquement une occurrence de $\mathtt{e}$, au plus tard à la date fixée par $\overline{d}(\mathtt{e})$,
– (Eps) : **Événement Peut Survenir.** Cette deuxième sémantique permet de relacher la contrainte d'obligation d'occurrence. En effet, $\mathtt{e}$ n'est pas obligé de survenir avant l'échéance fixée par $\overline{d}(\mathtt{e})$. Lorsque cet événement ne se produit pas, il faut à nouveau attendre au moins $\underline{d}(\mathtt{e})$ unités temporelles pour qu'il puisse, à nouveau, éventuellement survenir.

Anticipation. Enfin, la fonction δ introduit une *anticipation* de la première occurrence des événements. Ainsi, la première occurrence de $\mathtt{e} \in \mathcal{E}$ se produit de $\max\{\underline{d}(\mathtt{e})-\delta(\mathtt{e}), 0\}$ à $\overline{d}(\mathtt{e})-\delta(\mathtt{e})$ unités de temps après le début de l'exécution de l'application. Nous supposons donc le respect de la condition :

$$\textbf{(C1)} \qquad \delta(\mathtt{e}) \leq \overline{d}(\mathtt{e}).$$

L'anticipation s'avère particulièrement utile pour la spécification d'environnements périodiques. Par exemple, pour l'application des lecteurs/écrivains, décrite en section 2.1.1 (page 34), en donnant à $\mathtt{r_1}$ et $\mathtt{r_2}$ une période d'occurrence de 2 u.t. ($\underline{d}(\mathtt{r_1}) = \overline{d}(\mathtt{r_1}) = 2$ et $\underline{d}(\mathtt{r_2}) = \overline{d}(\mathtt{r_2}) = 2$), des anticipations différentes pour leur première occurrence : $\delta(\mathtt{r_1}) = 0$ et $\delta(\mathtt{r_2}) = 1$, permettent de spécifier un environnement périodique : $\mathtt{r_1}$ se produit aux instants absolus 2, 4, 6,..., alors que $\mathtt{r_2}$ se produit aux dates 1, 3, 5, etc. À la fin de cette section, nous spécifions, en figure 6.2 (page 134), une application pour laquelle nous utilisons l'anticipation pour obtenir ce même effet.

Définition formelle d'une spécification $\mathcal{H}$-Electre

À partir de ces éléments, nous pouvons donc définir la spécification temporelle d'un programme Electre.

Définition 6.1 (Spécification $\mathcal{H}$-Electre)
Une *spécification* $\mathcal{H}$-Electre est donnée par un 8-uplet $Spec = (\mathcal{M}, \mathcal{E}, \mathcal{P}, \underline{d}, \overline{d}, Pl, Pr, \delta)$ où :

1. $\mathcal{M}$ est un ensemble fini de modules,
2. $\mathcal{E} = \mathcal{E}_{(\mathrm{EDS})} \cup \mathcal{E}_{(\mathrm{EPS})}$ est un ensemble fini d'événements où :
   – $\mathcal{E}_{(\mathrm{EDS})}$ est l'ensemble des événements qui surviennent assurément,
   – $\mathcal{E}_{(\mathrm{EPS})}$ est l'ensemble des événements qui surviennent éventuellement,
   avec : $\mathcal{E}_{(\mathrm{EDS})} \cap \mathcal{E}_{(\mathrm{EPS})} = \emptyset$,
3. $\mathcal{P} = \mathcal{P}_{(\mathrm{PPF})} \cup \mathcal{P}_{(\mathrm{PPMF})}$ est un ensemble fini de places (ou processeurs) où :
   – $\mathcal{P}_{(\mathrm{PPF})}$ est l'ensemble des places à politique (PPF),
   – $\mathcal{P}_{(\mathrm{PPMF})}$ est l'ensemble des places gérées en (PPMF),
   vérifiant : $\mathcal{P}_{(\mathrm{PPF})} \cap \mathcal{P}_{(\mathrm{PPMF})} = \emptyset$,
4. $\underline{d}, \overline{d} : (\mathcal{M} \cup \mathcal{E}) \rightarrow \mathbb{Q}$ associent, à chaque module, une durée minimale et une durée maximale d'exécution, et à chaque événement, un délai minimal et un délai maximal séparant deux occurrences consécutives. Ces fonctions vérifient :
   – $\forall \mathtt{e} \in \mathcal{E},\ 0 < \underline{d}(\mathtt{e}) \leq \overline{d}(\mathtt{e}) < \infty$,
   – $\forall \mathtt{M} \in \mathcal{M},\ 0 < \underline{d}(\mathtt{M}) \leq \overline{d}(\mathtt{M}) < \infty$,
5. $Pl : \mathcal{M} \rightarrow \mathcal{P}$ associe une place à chaque module,
6. $Pr : \mathcal{M} \rightarrow \mathbb{N}_{>0}$ associe une priorité à chaque module,
7. $\delta : \mathcal{E} \rightarrow \mathbb{Q}$ spécifie, pour chaque événement, le délai d'anticipation pour sa première occurrence,

♦

Une spécification $\mathcal{H}$-Electre est *correcte* si elle vérifie la condition **(C1)**. Dans la suite, nous ne considérons que des spécifications $\mathcal{H}$-Electre correctes.

6.1.2 Le langage de spécification $\mathcal{H}$-Electre

Une spécification $\mathcal{H}$-ELECTRE consiste en la définition des caractéristiques des places, des modules et des événements d'un programme ELECTRE. Dans nos descriptions, une liste `<,`X`,...>` indique une liste optionnelle d'objets de même type que X. Une structure `(`X`|`Y`)` indique un choix : soit X est choisi, sinon c'est Y. Pour les intervalles `[`a`,`b`]`, il est supposé que $a \leq b$.

Spécification des places

La spécification des places, où les modules d'un programme ELECTRE s'exécutent, est rudimentaire. Il s'agit simplement d'indiquer le mode de gestion de la place : (PPF) ou (PPMF).

Ainsi, la spécification suivante définit une place P_1 (et éventuellement, les places P_2,...) de type (PPF).

```
PPF P1 <,P2,...>;
```

La déclaration d'une place gérée en (PPMF) se fait simplement en substituant le mot clef (PPMF) au mot clef (PPF) de la définition précédente. La spécification suivante définit donc la place P_1 (et éventuellement, les place P_2,...) gérées en respect de la politique (PPMF).

```
PPMF P1 <,P2,...>;
```

Spécification des modules

Les informations qui caractérisent un module sont :
- d'une part sa durée,
- d'autre part la place en laquelle il s'exécute,
- et enfin sa priorité dans cette place.

La durée du module est spécifiée par un intervalle : une durée minimale et une durée maximale, toutes les deux rationnelles. De plus, elles doivent respecter les hypothèses du langage ELECTRE : un module a une *durée finie et non nulle*. La priorité du module est donnée par un entier naturel. Plus cette valeur est élevée et plus le module est prioritaire. Voici la syntaxe de la spécification des modules :

```
module M1 <,M2,...> is
  lasts [a,b];
  processor Pl;
  priority Pr;
end
```

Les modules $M_1, M_2, \ldots$ partagent tous le même type défini par les informations de durée (mot clef : `lasts`), de placement (mot clef : `processor`) et de priorité (mot clef : `priority`).

Spécification des événements

Il est nécessaire de spécifier la période des occurrences (mot clef : `every`), l'obligation (mot clef : `occurs EDS`), ou au contraire la liberté d'occurrence (mot clef : `occurs EPS`), et finalement, l'anticipation de la première occurrence (mot clef : `offset`). La syntaxe de spécification des événements est par conséquent la suivante :

```
event e1 <,e2,...> is
  occurs (EPS|EDS);
  every [a,b];
  offset c;
end
```

où les nombres a, b et c sont des rationnels. De la même façon que pour les modules, les événements $e_1, e_2, \ldots$ partagent ici tous le même type caractérisé par le mode d'occurrence, la période d'occurrence et l'anticipation.

Structure et exemple d'une spécification $\mathcal{H}$-Electre

Une spécification $\mathcal{H}$-Electre est organisée de la façon suivante :
- tout d'abord les spécifications des places,
- ensuite, les descriptions des modules,
- et, finalement, les déclarations des événements.

La grammaire de notre langage est donnée dans le tableau 6.1, page 133. Prenons à nouveau l'exemple des lecteurs/écrivains, décrit par le programme de la figure 2.1 (page 35). La figure 6.2 (page 134) représente une spécification $\mathcal{H}$-Electre qui produit des séquences d'événements, qui sont celles de l'environnement présenté en figure 2.6 (page 52).

Le chronogramme de la figure 6.1 (ci-dessous) présente, sur une période de 7 unités de temps, les occurrences d'événements spécifiées en figure 6.2 (page 134). La zone grisée représente la

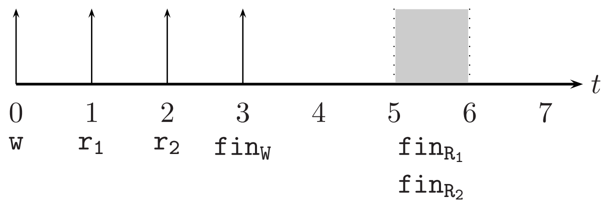


Fig. 6.1 – Description de l'environnement de la figure 6.2 sur une période de 7 unités de temps.

durée pendant laquelle $\mathtt{fin}_{\mathtt{R}_1}$ et $\mathtt{fin}_{\mathtt{R}_2}$ peuvent se produire, dans n'importe quel ordre. D'après ces spécifications, le motif présenté ici se répète avec une période de 7 u.t.

6.2 Systèmes à file réactifs hybrides linéaires

Nous introduisons maintenant les *systèmes à file réactifs hybrides linéaires* : le modèle obtenu par l'association des systèmes à file réactifs et des informations temporelles quantitatives données par les spécifications $\mathcal{H}$-Electre. Nous définissons tout d'abord les *systèmes hybrides linéaires* [6, 58, 7] qui permettent de modéliser des systèmes contenant à la fois une composante dynamique (dans notre contexte, des tâches qui s'exécutent) et une composante discrète (ici, les changement d'activité des tâches, comme, par exemple, leur terminaison). Puis, nous montrons

S	$::=$	$LP\ LM\ LE$
LP	$::=$	P
	$\mid$	$P\ LP$
P	$::=$	`PPF` LId `;`
	$\mid$	`PPMF` LId `;`
LM	$::=$	M
	$\mid$	$M\ LM$
M	$::=$	$Mh\ Md\ MPl\ MPr\ Fin$
Mh	$::=$	`module` LId `is`
Md	$::=$	`lasts` $Intervalle$ `;`
MPl	$::=$	`processor` Id `;`
MPr	$::=$	`priority` Ent `;`
LE	$::=$	E
	$\mid$	$E\ LE$
E	$::=$	$Eh\ Em\ Ep\ Eo\ Fin$
Eh	$::=$	`event` LId `is`
Em	$::=$	`occurs` Occ `;`
Occ	$::=$	`EPS`
	$\mid$	`EDS`
Ep	$::=$	`every` $Intervalle$ `;`
Eo	$::=$	`offset` Rat `;`
$Intervalle$	$::=$	`[`Rat`,`Rat`]`
Rat	$::=$	Ent`/`Ent
	$\mid$	Ent
Ent	$::=$	[0-9]+
LId	$::=$	Id
	$\mid$	Id`,`LId
Id	$::=$	[a-z,A-Z,0-9,_]+
Fin	$::=$	`end`

Tab. 6.1 – Grammaire du langage $\mathcal{H}$-Electre.

```
PPF P1;

module R1,R2 is
  lasts [1,3/2];
  processor P1;
  priority 1;
end

module W is
  lasts [3,3];
  processor P1;
  priority 1;
end
```

```
event w is
  occurs EDS;
  every [7,7];
  offset 7;
end

event r1 is
  occurs EDS;
  every [7,7];
  offset 6;
end

event r2 is
  occurs EDS;
  every [7,7];
  offset 5;
end
```

FIG. 6.2 – Spécification de l'environnement de la figure 2.6 (page 52) en $\mathcal{H}$-ELECTRE.

comment une spécification $\mathcal{H}$-ELECTRE est utilisée pour construire un *AFR Hybride Linéaire* dont la sémantique est donnée par un *SFR Hybride Linéaire*.

6.2.1 Les systèmes hybrides linéaires

Les *systèmes hybrides* [6, 7, 58] décrivent l'évolution de systèmes discrets dans un environnement qui évolue, lui, de façon continue. La composante continue modélise généralement un système physique, par exemple le niveau d'un réservoir alimenté par une vanne et vidangé par une autre. Un *pas continu*, dans l'évolution du système, consiste à faire évoluer le niveau du réservoir, suivant les lois physiques appropriées, en tenant compte des vannes ouvertes et fermées. Un *état discret* du système est donc décrit par l'état des vannes. Supposons que la vanne de vidange soit fermée et que celle d'alimentation soit ouverte, après un certain temps (correspondant au déroulement d'un pas continu), le niveau du réservoir atteint son niveau maximal. Il se produit alors un *pas discret* qui correspond à un changement d'état discret : la vanne d'alimentation est fermée, alors que la vanne de vidange est ouverte. Dans ce nouvel état discret, la loi qui décrit l'évolution du niveau est adaptée au nouvel état des vannes.

Dans notre description informelle des systèmes hybrides, nous n'avons pas précisé la forme que peuvent revêtir les lois d'évolution des variables qui décrivent le système, ou les conditions qui indiquent que ces variables atteignent leurs valeurs limites. La définition de Henzinger [58] n'introduit absolument aucune limite sur ces éléments. Ainsi, une loi d'évolution peut être donnée par une équation différentielle, et les conditions de limitation peuvent mettre en relation plusieurs variables, sous la forme d'équations non-linéaires, par exemple. Cette richesse d'expression a cependant l'inconvénient de rendre parfois impossible les calculs de successeurs ou de prédecesseurs nécessaires pour vérifier un système hybride. Nous n'avons pas besoin d'une telle expressivité pour modéliser les programmes ELECTRE, c'est pourquoi nous considérons une sous-classe des systèmes hybrides : les *systèmes hybrides linéaires* [6, 58, 7] pour lesquels les calculs de successeurs et de prédecesseurs sont possibles.

Définition et sémantique

Soit X un ensemble fini de n variables à valeurs dans $\mathbb{R}$. Une *valuation* v est un vecteur de $\mathbb{R}^n$. Une *expression linéaire* φ sur X est de la forme $\sum_{i=1}^{n} a_i.x_i$ avec, $\forall i \in \{1,\ldots,n\}$, $a_i \in \mathbb{Q}$ et $x_i \in X$.

Définition 6.2 (Contrainte linéaire)
Une *contrainte linéaire* [22] Φ, sur un ensemble de n variables X, est une formule propositionnelle utilisant les connecteurs $\vee, \wedge$ et $\neg$ sur des propositions atomiques de la forme $\varphi \bowtie c$, où $\bowtie \in \{<, \leq, =, \geq, >\}$, φ est une expression linéaire sur X, et $c \in \mathbb{Q}$. ♦

Définition 6.3 (Application affine)
Une *application affine*, sur un ensemble de n variables X, est définie par : $X := A.X + B$ où A est une matrice $n \times n$ à coefficients dans $\mathbb{Q}$ et B est un vecteur de $\mathbb{Q}^n$. ♦

Notons $\mathcal{CL}(X)$ l'ensemble des contraintes linéaires sur X, et $\mathcal{AL}(X)$ l'ensemble des applications affines sur X. Pour une valuation v et une contrainte linéaire Φ, nous notons $v \in \Phi$ si v satisfait Φ.

Définition 6.4 (Rectangle)
Un *rectangle* [59] I de $\mathbb{R}^n$, de dimension n, est un produit cartésien $I = \prod_{i=1}^{n} I_i$ où I_i est :

- soit un intervalle borné de $\mathbb{R}$ à bornes rationnelles : $I_i = [a,b],\ a,b \in \mathbb{Q}$ et $a \leq b$,
- soit un semi-ouvert de $\mathbb{R}$: $I_i = (a,b],\ a \in (\mathbb{Q} \cup -\infty), b \in \mathbb{Q}$ ou $I_i = [a,b),\ a \in \mathbb{Q}, b \in (\mathbb{Q} \cup +\infty)$, et $a \leq b$,
- sinon un ouvert de $\mathbb{R}$: $I_i = (a,b),\ a \in (\mathbb{Q} \cup -\infty), b \in (\mathbb{Q} \cup +\infty)$ et $a \leq b$.

♦

Définition 6.5 (Système hybride linéaire)
Un *système hybride linéaire (SHL)* [22] est un 10-uplet :

$$H = (Q_H, q_H^0, X_H, init_H, A_H, \rightarrow_H, act_H, inv_H, pre_H, post_H)$$

où :

1. Q_H est un ensemble fini d'états,
2. $q_H^0 \in Q_H$ est l'état initial de H,
3. X_H est un ensemble fini de variables à valeurs dans $\mathbb{R}$,
4. $init_H$ est la valuation initiale,
5. A_H est un ensemble d'actions,
6. $\rightarrow_H \subseteq Q_H \times A_H \times Q_H$ est la relation de transition de H,

7. $act_H : Q_H \to (\mathbb{Q}\times\mathbb{Q})^n$ associe, à chaque état, une *activité* de la forme $act_H(q) = \prod_{i=1}^n [\underline{d}_i; \overline{d}_i]$, signifiant que la première dérivé de $x_i \in X_H$ en q appartient à l'intervalle compact et borné $[\underline{d}_i; \overline{d}_i]$ de $\mathbb{Q}$,
8. $inv_H : Q_H \to \mathcal{CL}(X_H)$ associe un *invariant* à chaque état,
9. $pre_H : (\to_H) \to \mathcal{CL}(X_H)$ associe une *garde* à chaque transition,
10. $post_H : (\to_H) \to \mathcal{AL}(X_H)$ associe une transformation discrète des variables de X_H, à chaque transition de H.

♦

Une *évolution continue* des variables de X_H, depuis une valuation $v \in \mathbb{R}^n$, en un état $q \in Q_H$ et pendant une durée $t \in \mathbb{R}_{\geq 0}$ définit la valuation v', obtenue, en choisissant $D \in act_H(q)$ par : $v' = v + D.t$. Alors, du fait de la forme rectangulaire de $act_H(q) = \prod_{i=1}^n [\underline{d}_i; \overline{d}_i]$, en considérant l'ensemble des vecteurs D possibles, une *évolution continue* définit un cône de sommet v, dont les rayons sont les demi-droites de pentes $\underline{d}_i$ et $\overline{d}_i$ pour la dimension i. La figure 6.3 représente ceci pour la dimension i.

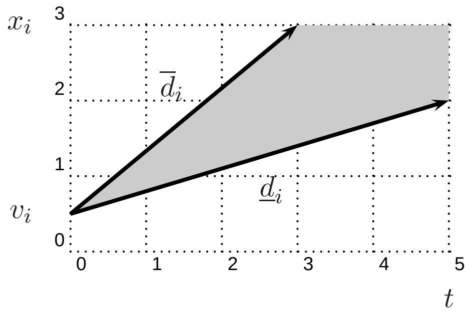


FIG. 6.3 – Cône défini par $act_H(q) = \prod_{i=1}^n [\underline{d}_i; \overline{d}_i]$ depuis v pour la dimension i.

Un système hybride linéaire H est *rectangulaire* si :
- quel que soit $q \in Q_H$, l'ensemble des valuations définit par $inv_H(q)$ est rectangulaire,
- et quel que soit $t \in (\to_H)$, $pre_H(t)$ est un ensemble rectangulaire et $post_H(t)$ associe à chaque variable, soit sa valeur actuelle, soit une valeur choisie aléatoirement dans une zone rectangulaire.

Définition 6.6 (Sémantique d'un système hybride linéaire)
La *sémantique d'un SHL* [22] $H = (Q_H, q_H^0, X_H, init_H, A_H, \to_H, act_H, inv_H, pre_H, post_H)$ est donnée par le système de transitions étiqueté : $(Q, q_0, L, \to)$ défini par (posons $n = Card(X_H)$) :

1. $Q = Q_H \times \mathbb{R}^n$,
2. $q_0 = \left(q_H^0, init_H\right)$,
3. $L = A_H \cup \mathbb{R}_{\geq 0}$,
4. $\to \subseteq Q \times L \times Q$ est définie par :
   (a) $\forall a \in A_H,\ (q, v) \xrightarrow{a} (q', v')$ si et seulement si :
   - $q \xrightarrow{a}_H q'$, notons t cette transition,

- et $v \in pre_H(t)$,
- et $v' \in post_H(t)$,
- et $v' \in inv_H(q')$.

(b) $\forall \delta \in \mathbb{R}_{\geq 0},\ (q, v) \xrightarrow{\delta} (q', v')$ si et seulement si :
- $q = q'$,
- et $\exists D \in act_H(q)$ t.q. $v' = (v + D.\delta)$,
- et $\forall 0 \leq \delta' \leq \delta,\ (v + D.\delta') \in inv_H(q)$.

♦

Dans la suite, $(q, v) \in Q$ est appelée une *configuration* de H. Depuis le début de cette section, nous avons utilisé les termes «pas continu» et «pas discret» pour désigner les transitions qui modélisent le passage du temps, et les changement d'états discrets du système, respectivement. En nous basant sur la définition 6.6, nous donnons une définition formelle de ces deux termes.

Définition 6.7 (Pas continu)
Un *pas continu* dans un système hybride linéaire H, dont la sémantique est donnée par le système de transitions $(Q, q_0, L, \rightarrow)$ est une transition :

$$(q, v) \xrightarrow{\delta} \big(q, v'\big)$$

avec $\delta \in \mathbb{R}_{\geq 0}$. ♦

Définition 6.8 (Pas discret)
Un *pas discret* dans un système hybride H, dont la sémantique est donnée par le système de transitions $(Q, q_0, L, \rightarrow)$ est une transition :

$$(q, v) \xrightarrow{a} \big(q', v'\big)$$

avec $a \in A_H$. ♦

Remarque 6.9
Par la définition 6.6, deux pas continus consécutifs de durées δ et δ' peuvent être rassemblés en un seul de durée $\delta + \delta'$:

$$(q, v) \xrightarrow{\delta} \big(q, v'\big) \xrightarrow{\delta'} \big(q, v''\big) \qquad \text{équivaut à} \qquad (q, v) \xrightarrow{(\delta + \delta')} \big(q, v''\big)$$

♦

Un *pas continu-discret* est alors l'enchainement d'un pas continu, puis d'un pas discret. Nous pouvons maintenant définir les exécutions d'un système hybride.

Définition 6.10 (Exécution d'un SHL)
Soit H un SHL dont la sémantique est donnée par le système de transitions $(Q, q_0, L, \rightarrow)$. Une *exécution* de H est une séquence de pas continus-discrets à partir de la configuration initiale $q_0 =$

$\left(q_H^0, init_H\right)$:

$$\left(q_H^0, init_H\right) \xrightarrow{\delta_0} (q_1, v_1) \xrightarrow{a_0} \left(q_1', v_1'\right) \ldots (q_i, v_i) \xrightarrow{\delta_i} (q_{i+1}, v_{i+1}) \xrightarrow{a_i} \left(q_{i+1}', v_{i+1}'\right) \ldots$$

♦

Dans la suite, $[\![H]\!]$ désigne l'ensemble des exécutions d'un système hybride H.

Graphe des régions

Le système de transitions qui définit la sémantique d'un système hybride linéaire n'est généralement pas calculable, puisque les pas continus impliquent qu'une configuration donnée a une infinité de successeurs immédiats. Pour cela, nous introduisons les *graphes de régions* qui, basés sur une relation d'équivalence sur les configurations, permet la vérification des systèmes hybrides linéaires. Pour cela, nous introduisons une relation d'équivalence sur les configurations des systèmes hybrides linéaires (nommée *équivalence discrète*, définition 6.11, ci-dessous) qui abstrait les pas continus qui permettent, ensuite, le franchissement des mêmes transitions discrètes.

Définition 6.11 (Équivalence discrète)
Soit H un SHL et $(Q, q_0, A, \rightarrow)$ le système de transitions qui en donne la sémantique. Nous considérons une relation d'*équivalence discrète*, notée $\cong$, est la relation binaire sur Q définie $\forall (q,v), (q',v')$ par :

$$\begin{aligned}
(q,v) \cong \left(q',v'\right) \Leftrightarrow q = q' \text{ et : } & \forall t \in \mathbb{R}_{\geq 0},\ \forall D \in act_H(q) \text{ t.q. } (q, v + D.t) \xrightarrow{a} \left(q'', v''\right) \\
& \exists t' \in \mathbb{R}_{\geq 0},\ \exists D' \in act_H(q') \text{ t.q. } \left(q', v' + D'.t'\right) \xrightarrow{a} \left(q''', v'''\right) \\
& \text{réciproquement : } \forall t' \in \mathbb{R}_{\geq 0},\ \forall D' \in act_H(q') \text{ t.q. } \left(q', v' + D'.t'\right) \xrightarrow{a} \left(q''', v'''\right) \\
& \exists t \in \mathbb{R}_{\geq 0},\ \exists D \in act_H(q) \text{ t.q. } (q, v + D.t) \xrightarrow{a} \left(q'', v''\right) \\
& \text{et dans les deux cas : } \left(q'', v''\right) \cong \left(q''', v'''\right)
\end{aligned}$$

♦

Deux configurations (q,v) et (q',v') sont donc $\cong$-équivalentes si depuis chacune d'elles, il est possible, après avoir laisser s'écouler le temps nécessaire, de franchir les mêmes transitions discrètes. Dans la suite une *région* désigne une classe d'équivalence pour $\cong$. Compte tenu de la définition de cette relation, par la suite, nous appelons aussi «*région*» les ensembles de valuations qui assurent l'équivalence.

Définition 6.12 (Graphe des régions)
Soit H un SHL de sémantique $(Q, q_0, A, \rightarrow)$. Le *graphe des régions* de H, noté $(H)_{\cong}$ est le système de transitions $(Q_{\cong}, Q_{\cong}^0, A_{\cong}, \rightarrow_{\cong})$ défini par :

1. $Q_{\cong}$ est la partition de Q définie par $\cong$,
2. $Q_{\cong}^0 = \left\{Q' \in Q_{\cong} \mid q_H^0 \in Q'\right\}$,

3. $A_{\cong} = A_H$,
4. $\rightarrow_{\cong} \subseteq Q_{\cong} \times A_{\cong} \times Q_{\cong}$ est définie $\forall Q', Q'' \in Q_{\cong}$ et $\forall a \in A_{\cong}$ par :

$$Q' \xrightarrow{a}_{\cong} Q'' \Leftrightarrow \exists \left(q', v'\right) \in Q',\ \exists \left(q'', v''\right) \in Q'' \text{ et } \exists \delta \geq 0 \text{ t.q. : } \left(q', v'\right) \xrightarrow{\delta} \xrightarrow{a} \langle q'', v'' \rangle$$

♦

La définition de $\cong$ induit de fait une relation de bisimulation, faisant abstraction des transitions temporelles, entre $(H)_{\cong}$ et la sémantique de H. Une *exécution* d'un graphe des régions est une chemin dans celui-ci. L'ensemble des éxécutions d'un graphe des régions $(H)_{\cong}$ est noté $\llbracket (H)_{\cong} \rrbracket$.

Le calcul d'un graphe des régions se fait en caractérisant les valuations qui permettent de franchir les transitions issues de chaque état q de H. Le lecteur trouvera dans [6] (par exemple), une méthode pour le calcul du graphe des régions d'un système hybride linéaire.

6.2.2 Systèmes à file réactifs hybrides linéaires

À partir d'une spécification $\mathcal{H}$-Electre *Spec* et d'un programme Electre P, nous souhaitons construire un SFR Embarqué qui corresponde à ces spécifications. Les informations temporelles quantitatives, apportées par *Spec*, nous conduisent naturellement à modéliser *Spec* sous la forme d'un système hybride linéaire H.

Notons R l'AFR obtenu par compilation du programme P. Certaines informations nécessaires pour la construction de H, comme le démarrage ou la préemption des modules, sont contenues dans R : elles en définissent les états (voir section 2.1.3, page 39 et section 2.1.3, page 40). Par ailleurs, la mesure du temps d'exécution d'un module `M`, nécessite l'emploi d'une variable qui doit avoir une activité nulle dans les états où `M` est préempté, et qui doit être initialisée lorsque `M` débute une nouvelle exécution. Ces informations définissent, justement, la structure même de R. De plus, les transitions de consommation, dans R, agissent sur l'état des modules, puisqu'elles sont les duales des transitions de traitement immédiat. Par conséquent, les informations de *Spec* sont directement intégrées à R pour obtenir un *automates à file réactif hybride linéaire (AFR Hybride Linéaire)*. La sémantique est alors donnée par un *système à file réactif hybride linéaire (SFR Hybride Linéaire)* qui allie les sémantiques des SFR et des SHL.

Construction de l'AFR Hybride Linéaire

Considérons un événement (Eps) `e`. Rappelons que la période d'occurrence de `e` lui fournit une «fenêtre» pendant laquelle il peut survenir, mais que cela n'a rien d'obligatoire. Le temps qui sépare deux occurrences successives de `e` est mesuré par une variable. Lorsque la valeur de cette variable atteint la période maximale séparant deux occurrences de `e`, et lorsque `e` ne survient pas, il faut tout de même réinitialiser cette variable. Nous devons donc ajouter des transitions qui se chargent de remettre à zéro les horloges des variables associées à un événement (Eps) qui ne survient pas. Il en est de même des événements fugaces qui surviennent à un moment où l'application modélisée ne peut pas les prendre en compte. Nous introduisons, pour cela, une nouvelle étiquette, λ, qui estampille ces transitions de réinitialisation des horloges.

Nous rappelons que les fonctions A, D et P associent, à chaque transition de R, les modules qui sont activés, démarrés et préemptés respectivement (voir en section 2.1.3, page 39). À partir de ces trois fonctions, nous construisons les fonctions :

- «$\mathcal{A}$» qui donne l'ensemble des modules qui s'exécutent dans un état de l'AFR. Attention cependant, cette fonction ne tient bien évidemment pas compte des priorités de modules. Par conséquent, un module s'exécutant sur une place (PPMF) peut être actif selon $\mathcal{A}$ tout en ne s'exécutant pas réellement, n'étant pas le plus prioritaire.
- «On» qui associe à chaque transition t de R les modules qui commencent une nouvelle exécution dans l'état cible de t.

Le calcul de ces deux fonctions à partir de A, D et P, est abordé dans [20, 16].

Soit $\mathcal{M}^{Pl}(p) = \{\mathtt{M} \in \mathcal{M} \,|\, Pl(\mathtt{M}) = p\}$ l'ensemble des modules qui s'exécutent dans la place $p \in \mathcal{P}$. Nous notons $\mathcal{A}_p(q) = \mathcal{A}(q) \cap \mathcal{M}^{Pl}(p)$ l'ensemble des modules d'une place p actifs en un état q d'un AFR, et $max_p^{Pr}(q) = \{\mathtt{M} \in \mathcal{A}_p(q) \,|\, \forall \mathtt{M}' \in \big(\mathcal{A}(q) \cap \mathcal{M}^{Pl}(Pl(\mathtt{M}))\big),\ Pr(\mathtt{M}) \geq Pr(\mathtt{M}')\}$ l'ensemble des modules de la place p, de priorité maximale, qui sont actifs en q.

En considérant ces éléments, nous construisons maintenant, l'*AFR Hybride Linéaire* défini par un AFR et une spécification $\mathcal{H}$-ELECTRE.

Définition 6.13 (AFR Hybride Linéaire)
Soit $Spec = (\mathcal{M}, \mathcal{E}, \mathcal{P}, \underline{d}, \overline{d}, Pl, Pr, \delta)$ la spécification $\mathcal{H}$-ELECTRE pour un programme ELECTRE P. Soit $R = (Q_R, q_R^0, A_R, \rightarrow_R)$ l'AFR obtenu par compilation de P.

L'*automate à file réactif hybride linéaire (AFR Hybride Linéaire)*

$$R_H = (Q_{R_H}, q_{R_H}^0, X_{R_H}, init_{R_H}, A_{R_H}, \rightarrow_{R_H}, act_{R_H}, inv_{R_H}, pre_{R_H}, post_{R_H})$$

est défini depuis R et $Spec$ par :

1. $Q_{R_H} = Q_R$,
2. $q_{R_H}^0 = q_R^0$,
3. $A_{R_H} = A_R \cup \{\lambda\}$, avec $\lambda \notin (\Sigma)_R$,
4. $X_{R_H} = \{x_\mathtt{M} \,|\, \mathtt{M} \in \mathcal{M}\} \cup \{x_\mathsf{e} \,|\, \mathsf{e} \in \mathcal{E}\}$,
5. $init_{R_H} = \left(\bigwedge_{\mathtt{M} \in \mathcal{M}} x_\mathtt{M} := 0\right) \wedge \left(\bigwedge_{\mathsf{e} \in \mathcal{E}} x_\mathsf{e} := \delta(\mathsf{e})\right)$,
6. quel que soit $q \in Q_H$, $inv_{R_H}(q) = \left(\bigwedge_{\mathtt{M} \in \mathcal{A}(q)} x_\mathtt{M} \leq \overline{d}(\mathtt{M})\right) \wedge \left(\bigwedge_{\mathsf{e} \in \mathcal{E}} x_\mathsf{e} \leq \overline{d}(\mathsf{e})\right)$,
7. quel que soit $q \in Q_H$:

$$act_{R_H}(q) = \left(\bigwedge_{\mathtt{M} \in \mathcal{M}} \dot{x}_\mathtt{M} = \begin{cases} \dfrac{Pr(\mathtt{M})}{\sum\limits_{\mathtt{M}' \in \mathcal{A}_{Pl(\mathtt{M})}(q)} Pr(\mathtt{M}')} & \text{si } \mathtt{M} \in \mathcal{A}(q) \text{ et } Pl(\mathtt{M}) \in \mathcal{P}_{(\mathrm{PPF})} \\ \dfrac{1}{Card(max_{Pl(\mathtt{M})}^{Pr}(q))} & \text{si } \mathtt{M} \in max_{Pl(\mathtt{M})}^{Pr}(q) \text{ et } Pl(\mathtt{M}) \in \mathcal{P}_{(\mathrm{PPMF})} \\ 0 & \text{sinon} \end{cases}\right)$$
$$\wedge \left(\bigwedge_{\mathsf{e} \in \mathcal{E}} \dot{x}_\mathsf{e} = 1\right)$$

8. et $\rightarrow_{R_H} \subseteq Q_H \times A_H \times Q_H$ est définie par :
   (a) quel que soit $\mathsf{e} \in (\Sigma)_R$, $q \xrightarrow{\mathsf{e}}_{R_H} q'$ si $q \xrightarrow{\mathsf{e}}_R q'$, et, en notant t cette transition :
   - $pre_{R_H}(t) = \big(\underline{d}(\mathsf{e}) \leq x_{\mathsf{e}} \leq \overline{d}(\mathsf{e})\big)$,
   - $post_{R_H}(t) = (x'_{\mathsf{e}} := 0) \wedge \left(\bigwedge_{\mathtt{M} \in On(t)} x'_{\mathtt{M}} := 0\right)$.

   (b) quel que soit $\mathsf{e} \in (\Sigma_M)_R$, $q \xrightarrow{?\mathsf{e}}_{R_H} q'$ si $q \xrightarrow{?\mathsf{e}}_R q'$ et :
   - $pre_{R_H}(t) = true$,
   - $post_{R_H}(t) = \left(\bigwedge_{\mathtt{M} \in On(t)} x'_{\mathtt{M}} := 0\right)$.

   (c) quel que soit $\mathsf{e} \in (\Sigma_M)_R$, $q \xrightarrow{!\mathsf{e}}_{R_H} q$ si $q \xrightarrow{!\mathsf{e}}_R q$ et :
   - $pre_{R_H}(t) = \big(\underline{d}(\mathsf{e}) \leq x_{\mathsf{e}} \leq \overline{d}(\mathsf{e})\big)$,
   - $post_{R_H}(t) = (x'_{\mathsf{e}} := 0)$.

   (d) quel que soit $\mathtt{M} \in \mathcal{M}$, $q \xrightarrow{\mathtt{fin}_{\mathtt{M}}}_{R_H} q'$ si $q \xrightarrow{\mathtt{fin}_{\mathtt{M}}}_R q'$, et en notant t cette transition de R_H :
   - $pre_{R_H}(t) = \big(\underline{d}(\mathtt{M}) \leq x_{\mathtt{M}} \leq \overline{d}(\mathtt{M})\big)$,
   - $post_{R_H}(t) = id$.

   (e) quel que soit $q \in Q_H$, et quel que soit $\mathsf{e} \in (\mathcal{E}_{(\text{EPS})} \cup (\Sigma_F \backslash out(q)))$, nous avons $q \xrightarrow{\lambda}_{R_H} q$, et :
   - $pre_{R_H}(t) = \big(x_{\mathsf{e}} = \overline{d}(\mathsf{e})\big)$,
   - $post_{R_H}(t) = (x'_{\mathsf{e}} := 0)$.

♦

Dans cette définition, id représente l'application identité, et x' représente la valeur de la variable $x \in X_{R_H}$ après une transformation discrète. En n'imposant aucune contrainte pour le franchissement des transitions de consommations ($pre_{R_H} = true$), nous donnons la priorité au traitement des occurrences mémorisées, respectant ainsi la sémantique des SFR. Enfin, la définition de act_{R_H} prend en compte les politiques (PPF) et (PPMF). Pour la première, la vitesse d'exécution du module est définie par la priorité du module, divisée par la somme des priorités des modules actifs dans l'état considéré, et s'exécutant dans la même place :

$$\frac{Pr(\mathtt{M})}{\sum_{\mathtt{M}' \in \mathcal{A}_{Pl(\mathtt{M})}(q)} Pr(\mathtt{M}')} \quad \text{si } \mathtt{M} \in \mathcal{A}(q) \text{ et } Pl(\mathtt{M}) \in \mathcal{P}_{(\text{PPF})}$$

Pour la politique (PPMF), seuls les modules de priorité maximale, de chaque place, qui sont actifs dans l'état considéré, s'exécutent, et ils se partagent alors la ressource équitablement.

$$\frac{1}{Card(max^{Pr}_{Pl(\mathtt{M})}(q))} \quad \text{si } \mathtt{M} \in max^{Pr}_{Pl(\mathtt{M})}(q) \text{ et } Pl(\mathtt{M}) \in \mathcal{P}_{(\text{PPMF})}$$

Remarque 6.14

Il est clair, suivant la forme des contraintes linéaires pre_{R_H}, $post_{R_H}$ et inv_{R_H} que les AFR Hybrides Linéaires sont rectangulaires. ♦

Considérons à nouveau l'exemple des lecteurs/écrivains. L'AFR R correspondant est représenté en figure 2.4 (page 43). Nous considérons la spécification $\mathcal{H}$-ELECTRE *Spec* de la figure 6.2 (page 134). La figure 6.4 représente une partie[1] de l'AFR Hybride Linéaire construit à partir de R et *Spec*.

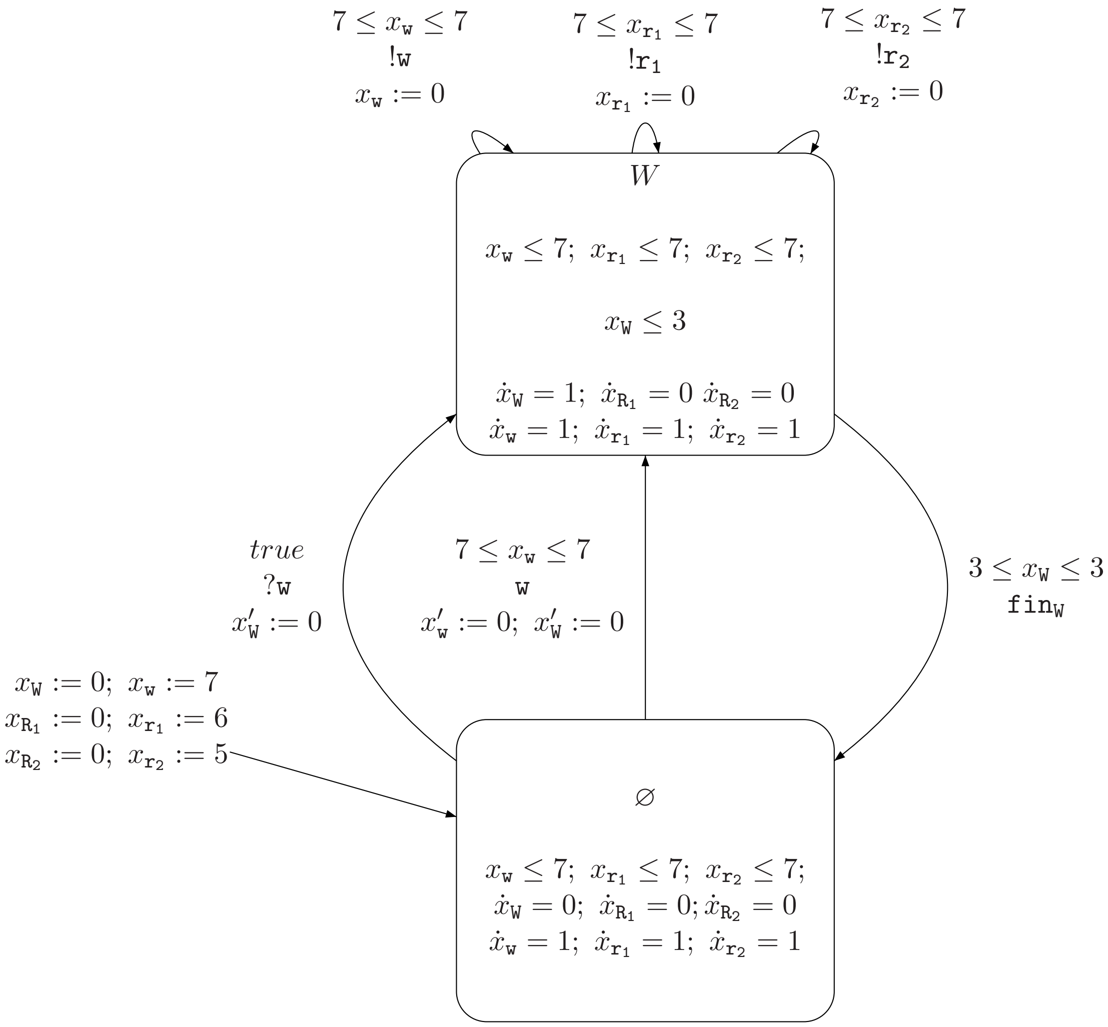


FIG. 6.4 – Une partie de l'AFR Hybride Linéaire pour les lecteurs/écrivains.

Sémantique et propriétés des AFR Hybrides Linéaires

La sémantique d'un AFR Hybride Linéaire combine la sémantique des AFR et celle des systèmes hybrides linéaires. La particularité de ce nouveau modèle est qu'il faut tenir compte des interactions entre la gestion de la file de mémorisation et les temporisations. En particulier, puisque les occurrences mémorisées ont la priorité, dès lors qu'il est possible de consommer une occurrence mémorisée, il n'est pas possible de laisser s'écouler du temps. Une *configuration* d'un SFR Hybride Linéaire est donc définie par un état $q \in Q_{R_H}$, un contenu de file $w \in (\Sigma_M)^*_{R_H}$ et une valuation $v \in \mathbb{R}^n$ des variables de X_{R_H}.

[1]Nous ne montrons ici que les états $\varnothing$ et W, faute de place pour représenter l'AFR Hybride Linéaire en entier.

Définition 6.15 (SFR Hybride Linéaire)
Soit $R_H = (Q_{R_H}, q^0_{R_H}, A_{R_H}, \rightarrow_{R_H}, X_{R_H}, init_{R_H}, inv_{R_H}, act_{R_H}, pre_{R_H}, post_{R_H})$ un AFR Hybride Linéaire. Nous nommons $R = (Q_R, q^0_R, A_R, \rightarrow_R)$ l'AFR sous-jacent, et $S = (Q_S, q^0_S, A_S, \rightarrow_S)$ représente le SFR obtenu depuis R (voir la définition 2.11, page 44).

Le *système à file réactif hybride linéaire (SFR Hybride Linéaire)*, qui définit la sémantique de R_H, est le système de transitions $S_H = (Q_{S_H}, Q^0_{S_H}, A_{S_H}, \rightarrow_{S_H})$ défini par :

1. $Q_{S_H} \subseteq Q_{R_H} \times \Sigma^*_M \times \mathbb{R}^n$ est l'ensemble des configurations de S_H vérifiant : $\forall (q, w, v) \in Q_{S_H},\ v \in inv_{R_H}(q)$,
2. $q^0_{S_H} = (q^0_{R_H}, \varepsilon, init_{R_H})$ est la configuration initiale de S_H,
3. $A_{S_H} = A_{R_H} \cup \mathbb{R}_{\geq 0}$,
4. $\rightarrow_{S_H} \subseteq Q_{S_H} \times A_{S_H} \times Q_{S_H}$ est la relation de transition définie pour toute configuration $(q, w, v) \in Q_{S_H}$ par :
   (a) $\forall a \in A_{R_H},\ (q, w, v) \xrightarrow{a}_{S_H} (q', w', v')$ si :
   - $(q, w) \xrightarrow{a}_S (q', w')$,
   - et, en notant $t = q \xrightarrow{a}_{R_H} q'$:
     - $v \in pre_{R_H}(t)$,
     - et $v' \in post_{R_H}(t)$,
     - et $v' \in inv_{R_H}(q')$,

   (b) $\forall \delta \in \mathbb{R}_{\geq 0},\ (q, w, v) \xrightarrow{\delta}_{S_H} (q', w', v')$ si :
   - $(q', w') = (q, w)$,
   - et $\mathbf{w} \in (\mathbf{\Sigma^!_q})^*$,
   - et $\exists D \in act_{R_H}(q)$ t.q. $v' = (v + D.\delta)$,
   - et $\forall 0 \leq \delta' \leq \delta,\ (v + \delta'.t) \in inv_{R_H}(q)$.

♦

Notons la particularité des SFR Hybrides Linéaires : les transitions de consommation doivent être franchies au plus tôt, comme la condition $w \in (\Sigma^!_q)^*$ l'impose dans le point (4b) ci-dessus. Il est donc possible de laisser s'écouler du temps, uniquement dans les configurations stables (par extension de cette notion pour les SFR). Les spécifications apportées par les contraintes temporelles dans un SFR Hybride Linéaire S_H permettent de sélectionner certaines exécutions du SFR S, sous-jacent à S_H, et par conséquent, les SFR Hybrides Linéaires forment une classe particulière de SFR Embarqués.

La relation d'*équivalence discrète* définie pour les systèmes hybrides linéaires (définition 6.11, page 138) s'adapte très facilement aux SFR Hybrides Linéaires. Deux configurations sont équivalentes si elles le sont au sens de la définition 6.11, et si leurs deux files sont égales. Nous notons $\cong_f$ cette nouvelle relation d'équivalence discrète qui tient compte de la file.

$$\langle q, w, v\rangle \cong_f \langle q', w', v'\rangle \Leftrightarrow (q, v) \cong \left(q', v'\right) \text{ et } w = w' \tag{6.1}$$

La notion de *graphe des régions* (définition 6.12, page 138) s'étend alors naturellement aux SFR Hybrides Linéaires.

Finalement, tout environnement périodique peut s'exprimer en $\mathcal{H}$-Electre, donc :

Théorème 6.16
Tous les problèmes d'interêt sont indécidables pour les SFR Hybrides Linéaires avec 2 événements mémorisables. ♦

6.3 Test de non-bornitude pour les SFR Hybrides Linéaires

Le problème de la bornitude n'est pas décidable pour les SFR Hybrides Linéaires (par le théorème 6.16, ci-dessus). Cependant, il est important de s'assurer que le système modélisé se contente d'une mémoire de taille bornée. Dans cette section, nous adaptons notre test de non-bornitude, présenté au chapitre 5 (page 107) au contexte hybride.

6.3.1 Principe du test pour les SFR Hybrides Linéaires

Le test de non-bornitude pour les SFR Embarqués, que nous présentons au chapitre 5 (page 107), se base sur l'exploration de l'arbre des exécutions du SFR Embarqué $S||E$ considéré, à la recherche des *cycles* dans les exécutions de $S||E$. Un cycle σ est alors un chemin allant d'une configuration $\langle q_R, w, q_E\rangle$ à une configuration $\langle q'_R, w', q'_E\rangle$ avec $q_R = q'_R$. Lorsqu'un cycle est trouvé, nous décidons son *itérabilité infinie*, c'est à dire l'existence d'une exécution de $S||E$, depuis $\langle q_R, w, q_E\rangle$, qui consiste à répéter infiniment les transitions de σ. Un cycle *rend la file non bornée* si et seulement s'il existe un événement mémorisable e qui est mémorisé plus de fois qu'il est consommé sur σ. Lorsqu'un cycle infiniment itérable, et rendant la file non bornée, est trouvé, nous concluons à la non-bornitude de $S||E$.

Nous adaptons ici ce principe aux SFR Hybrides Linéaires. Considérons donc un SFR Hybride Linéaire S_H. Pour tester la non-bornitude de S_H, nous devons :

1. être en mesure de développer l'arbre d'accessibilité de S_H,
2. pouvoir détecter les cycles dans les exécutions de S_H,
3. être capables de déterminer si un cycle est infiniment itérable,
4. enfin, déterminer si un cycle rend la file non bornée.

(Point 1) Il n'est pas possible de développer l'arbre d'accessibilité de S_H. En effet depuis une configuration $\langle q, w, v\rangle$, dès qu'il est possible de franchir un pas continu de durée non nulle, alors, $\mathbb{R}_{\geq 0}$ étant un domaine dense, $\langle q, w, v\rangle$ a une infinité de successeurs.

La solution consiste alors à développer l'arbre d'accessibilité du graphe des régions $(S_H)_{\cong_f}$, de S_H. En effet, par la définition même de $\cong_f$, à tout cycle de S_H (définition 6.17 ci-dessous) correspond un cycle de $(S_H)_{\cong_f}$, et vice-versa. En effet, seules les transitions discrètes nous intéressent pour la bornitude. De plus, la relation de transition de R_H (l'AFR Hybride Linéaire de sémantique S_H) étant finie, toute configurations de $(S_H)_{\cong_f}$ admet un nombre fini de successeurs. La méthode de calcul du graphe des régions de [6], pourra alors facilement être adaptée aux SFR Hybrides Linéaires, compte tenu de la définition de $\cong_f$.

Notons cependant que même lorsque le SFR Hybride Linéaire est borné, notre test de non-bornitude peut ne pas terminer : le graphe des régions d'un SFR Hybride Linéaire n'est à priori pas fini.

Nous considérons donc désormais $(S_H)_{\cong_f}$ pour notre test de non-bornitude. Les configurations de ce système sont de la forme $\langle q, w, R\rangle$, où R est une région (définition 6.12, page 138).

(Point 2) Afin de détecter une exécution finie d'un SFR Hybride Linéaire qui rend la file non bornée, nous recherchons des cycles qui peuvent être exécutés une infinité de fois consécutivement. Un cycle est donc un chemin partant et aboutisant dans un même état q, indépendamment de la file et des valuations des variables.

Définition 6.17 (Cycle dans un SFR Hybride Linéaire)
Un chemin σ, dans $(S_H)_{\cong_f}$, le graphe des régions d'un SFR Hybride Linéaire, allant de $\langle q, w, R\rangle$ à $\langle q', w', R'\rangle$ est un *cycle* si et seulement si $q = q'$. ♦

(Point 3) L'itérabilité infinie d'un cycle :

$$\sigma = \langle q_1, w_1, R_1\rangle \xrightarrow{a_1} \ldots \xrightarrow{a_n} \langle q_{n+1}, w_{n+1}, R_{n+1}\rangle$$

avec $q_1 = q_{n+1}$, est liée à la possibilité de franchir une infinité de fois chacune des transitions de σ. La possibilité de franchir une transition :

$$\langle q_i, w_i, R_i\rangle \xrightarrow{a} \langle q_{i+1}, w_{i+1}, R_{i+1}\rangle$$

du graphe des régions $(S_H)_{\cong_f}$ est définie par :

- le contenu de la file w_i,
- les valuations de R_i.

Notre critère d'itérabilité d'un cycle de $(S_H)_{\cong_f}$ doit donc tenir compte de ces deux éléments. Nous dirons que σ est itérable *vis-à-vis de la file*, si les configurations (q_1, w_1, R_1) et $(q_{n+1}, w_{n+1}, R_{n+1})$ respectent la relation $\mathcal{I}$ (section 5.1.2, page 109), i.e. $((q_1, w_1, R_1), (q_{n+1}, w_{n+1}, R_{n+1})) \in \mathcal{I}$. Dans ce cas, conformément aux résultats présentés dans le chapitre 5 (page 107), la transformation de la file induite par σ préserve la possibilité de franchir les transitions de σ.

Il nous reste donc à considérer les conditions sur les régions pour l'itérabilité de σ. Cette condition apparaît dans la condition **(CN1)** (page 110) de notre test de non-bornitude pour les SFR Embarqués. Nous nous concentrons, dans les deux sections qui suivent, sur l'itérabilité d'un cycle *vis-à-vis des régions*.

(Point 4) Le critère pour qu'un cycle rende la file non bornée est le même que celui utilisé au chapitre 5 (page 107) : il faut et il suffit qu'un événement mémorisable soit mémorisé plus de fois qu'il est consommé sur le cycle.

6.3.2 Modélisation des cycles dans les SFR Hybrides Linéaires

Avant de considérer l'itérabilité d'un cycle σ vis-à-vis des régions, nous modélisons σ sous la forme d'une *chaîne hybride* [71].

Définition 6.18 (Chaîne hybride)
Soit :

$$\sigma = \langle q_1, w_1, R_1 \rangle \xrightarrow{a_1} \langle q_2, w_2, R_2 \rangle \xrightarrow{a_2} \ldots \xrightarrow{a_k} \langle q_{k+1}, w_{k+1}, R_{k+1} \rangle$$

un cycle d'un SFR Hybride Linéaire. La *chaîne hybride* [71] définie par σ est la séquence de k triplets $[\gamma_1, \rho_1, \delta_1], \ldots, [\gamma_k, \rho_k, \delta_k]$ où $[\gamma_i, \rho_i, \delta_i]$ est défini par :

1. $\gamma_i = pre_H(q_i \xrightarrow{a_i} q_{i+1}) \cap inv_H(q_i)$,
2. $\rho_i = post_H(q_i \xrightarrow{a_i} q_{i+1}) \cap inv_H(q_{i+1})$,
3. $\delta_i = \begin{cases} \left(\bigwedge_{x \in X_H} \dot{x} = 0 \right) & \text{si } a_{i+1} = ?\mathtt{e},\ \mathtt{e} \in \Sigma_M \\ act_H(q_{i+1}) & \text{sinon} \end{cases}$

avec $q_{k+1} = q_1$ et $a_{k+1} = a_1$. ♦

Remarquons la particularité du point (3) de cette définition. Lorsque $a_{i+1} = ?\mathtt{e}$, $\mathtt{e} \in \Sigma_M$, par la sémantique des SFR Hybrides Linéaires (définition 6.15, page 142), il n'est pas possible de laisser s'écouler de temps dans l'etat q_{i+1}. La modélisation sous la forme de chaînes hybrides doit donc tenir compte de la sémantique particulière des SFR Hybrides Linéaires. La condition $\left(\bigwedge_{x \in X_H} \dot{x} = 0 \right)$ permet d'imposer que les valeurs des variables de X_H n'évoluent pas en q_{i+1}. Ce qui conduit au même effet : la valeur de X_H reste inchangée.

De même que pour les SHL, une chaîne hybride est *rectangulaire* si pour tous les triplets $[\gamma_i, \rho_i, \delta_i]$, γ_i, ρ_i et δ_i sont des rectangles.

Une chaîne hybride pour un cycle σ représente la transformation appliquée aux variables de R_H lors de l'exécution de σ. Suivant [71], nous modélisons maintenant cette transformation sous la forme d'un système discret.

Considérons un triplet $[\gamma_i, \rho_i, \delta_i]$ et une valuation $v \in \gamma_i$. Lors du parcours de ce triplet, v est d'abord transformée par ρ_i. Par construction, ρ_i est une application linéaire, donc cette transformation s'écrit : $v' = A_i.v + B_i$, où A_i et B_i sont la matrice $n \times n$ de $\mathbb{R}$ et le vecteur de $\mathbb{R}^n$ qui définissent ρ_i.

Ensuite, nous appliquons à v', la transformée de v, l'évolution continue définie par δ_i. Rappelons que $\delta_i = prod_{j=1}^{n}[\underline{d}_j; \overline{d}_j]$ est un rectangle compact et borné de $\mathbb{R}^n$ qui défini l'ensemble des vitesses d'évolution possibles, pour les variables de H, dans l'état q_{i+1}. Alors, pour un vecteur $D_i \in \delta_i$, depuis v', et après un temps $t_i \in \mathbb{R}_{\geq 0}$, la nouvelle valuation est $v'' = v' + D_i.t_i$. Cependant, δ_i définit un ensemble de vitesses possibles (c'est à dire un ensemble de vecteurs D_i) et, la transformation est appliquée pour un temps t_i quelconque. Par conséquent, la transformation de v' par δ_i définit un cône de sommet v', et donc les rayons sont de pente $\underline{d}_j$ et $\overline{d}_j$ dans la dimension j. La figure 6.3 représente ceci pour une dimension i donnée. Notons $C_i = \bigcup_{t_i \geq 0} t_i.\delta_i$ le cône ainsi défini. La transformation de v par le triplet $[\gamma_i, \rho_i, \delta_i]$ défini le sous-espace R de $\mathbb{R}^n$:

$$R = A_i.v + B_i + C_i$$

Puis, parmi les points de R ainsi obtenus, seuls ceux qui permettent de franchir le motif suivant (d'indice $i + 1$), nous intéressent. Nous définissons donc la *transformation avant* en ce sens : nous tenons compte de γ_{i+1}.

Définition 6.19 (Transformation avant)
Soit $[\gamma_1, \rho_1, \delta_1], \ldots, [\gamma_k, \rho_k, \delta_k]$ une chaine hybride et $v \in \mathbb{R}^n$ une valuation. La *transformation avant* de v par le triplet d'indice i est définie par :

$$T_i^+(v) = (A_i.v + B_i + C_i) \cap \gamma_{i+1}$$

avec $\gamma_{k+1} = \gamma_1$.

La *transformation avant* de v par la chaine entière est alors définie par la composition des transformations avant pour les rangs 1 à n :

$$T^+(v) = T_k^+(T_{k-1}^+(\ldots T_1^+(v)\ldots))$$

♦

Notons que l'opération de transformation avant est étendue à des régions R en utilisant les mêmes méthodes que celles évoquées pour le calcul du graphe des régions (par exemple, celle de [6]).

Définition 6.20 (Trajectoire)
Une *trajectoire* d'une chaîne hybride $[\gamma_1, \rho_1, \delta_1], \ldots, [\gamma_k, \rho_k, \delta_k]$ est une séquence de valuations pour X :

$$\{V(1), \ldots, V(k)\}_1, \{V(1), \ldots, V(k)\}_2, \ldots, \{V(1), \ldots, V(k)\}_L$$

avec : $V(i+1) = T_i^+(V(i))$ et $V(k)_1 = V(1)_2, \ldots, V(k)_{L-1} = V(1)_L$. ♦

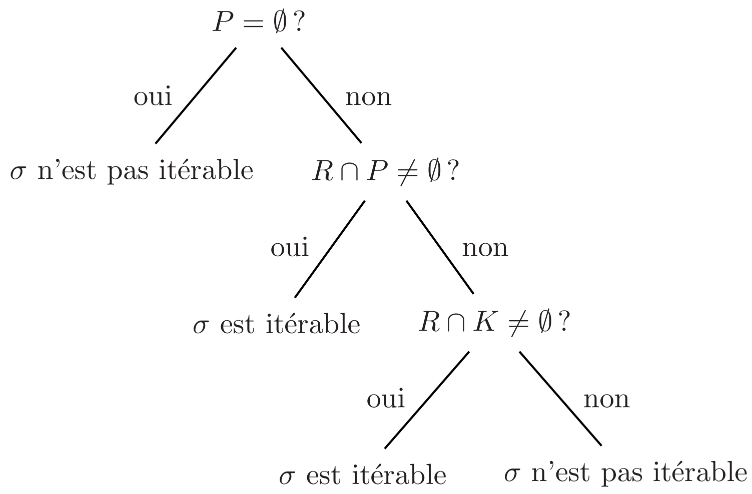
Une trajectoire est *viable* si elle est infinie : $L = \infty$. Une valuation est *viable* si elle est l'origine d'une trajectoire viable. Un *point fixe* d'une chaîne hybride est une valuation $v \in \mathbb{R}^n$ telle que $V(1)_1 = V(1)_2 = \cdots = V(1)_L = v$.

Définition 6.21 (Noyau de stabilité)
Le *noyau de stabilité* [71] d'une chaîne hybride est l'ensemble de ses points viables. ♦

À partir de tout point du noyau de stabilité, il est donc possible d'itérer infiniment le cycle σ considéré. Il reste à déterminer l'existence d'un point viable, et cas échéant, à le calculer, pour décider, ensuite, si ce point figure dans la configuration $\langle q, w, R\rangle$, depuis laquelle nous souhaitons connaitre l'itérabilité de σ.

Lemme 6.22 (Kourjanski & Varaiya [71])
Une chaîne hybride admet un point viable si et seulement si elle admet un point fixe. ♦

Dans la section suivante, nous donnons un semi-algorithme pour le calcul du noyau de stabilité d'une chaîne hybride. Puis, dans la section 6.3.4, nous indiquons une méthode pour calculer l'ensemble des points fixes d'une chaîne hybride rectangulaire. Ceci permet d'éviter de calculer le noyau de stabilité lorsque celui-ci est vide, cas dans lequel les risques que le semi-algorithme de termine pas est très élevé. Finalement, nous avons donc un semi-algorithme pour tester l'itérabilité vis-à-vis des régions, d'un cycle σ, à partir d'une configuration $\langle q, w, R \rangle$, dans un SFR Hybride Linéaire :



où : P désigne l'ensemble des points fixes de σ, et K son noyau de stabilité (lorsqu'ils sont calculables).

FIG. 6.5 – Semi-algorithme pour l'itérabilité vis-à-vis des régions d'un cycle σ depuis une configuration $\langle q, w, R \rangle$ d'un SFR Hybride Linéaire.

6.3.3 Calcul du noyau de stabilité K

Considérons une chaîne hybride $[\gamma_1, \rho_1, \delta_1], \ldots, [\gamma_k, \rho_k, \delta_k]$ et T^+ sa transformation avant (définition 6.19, page 147). Dans [71], Kourjanski et Varaiya proposent un semi-algorithme pour le calcul du noyau de stabilité de chaque état, c'est à dire l'ensemble des points viables de chaque γ_i.

Pour notre test d'itérabilité, nous avons uniquement besoin du noyau de stabilité dans le premier état, c'est à dire de $K \subseteq \gamma_1$. Nous présentons donc ci-dessous un semi-algorithme plus simple dans sa formulation.

$K \leftarrow \gamma_1$
tant que $T^+(K) \not\subseteq K$ **faire**
  $K \leftarrow K \cap T^+(K)$
fin tant que
return K

Algorithme 6.1: Semi-algorithme pour le calcul du noyau de stabilité d'une chaîne hybride.

Ce semi-algorithme ne termine pas lorsqu'il existe une suite infinie et strictement décroissante $\gamma_1 \supseteq K_1 \supseteq K_2 \supseteq \ldots$ de régions contenant des points non-viables. En effet, dans ce cas, le test $T^+(K) \not\subseteq K$, échoue systématiquement puisque l'algorithme ne parvient jamais à éliminer de K tous les points non-viables. Par contre, si à un moment donné, K ne contient plus que des points viables, l'algorithme converge. La figure 6.6 ci-dessous présente un SHL pour lequel l'algorithme 6.1 ne converge pas. En effet, à partir de la région initiale $[-1;1]$, l'algorithme calcule la suite des régions : $[\frac{-1}{2};\frac{1}{2}], [\frac{-1}{4};\frac{1}{4}], \ldots, [\frac{-1}{2^k};\frac{1}{2^k}], \ldots$ sans jamais converger sur l'unique point viable : 0.

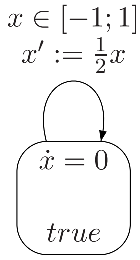


Fig. 6.6 – Un SHL pour lequel l'algorithme 6.1 ne converge pas.

Il est par conséquent très bénéfique de savoir, avant d'utiliser cet algorithme, si la chaîne hybride considérée possède des points viables. Outre l'inutilité du calcul de K dans le cas contraire, l'algorithme 6.1 présente alors de grands risques de ne pas terminer.

6.3.4 Calcul de l'ensemble des points fixes P

Nous pouvons déterminer l'existence de points viables pour une chaîne hybride si nous pouvons déterminer l'existence de points fixes (lemme 6.22, page 147). Nous présentons, dans cette section, un algorithme de calcul de ces points fixes pour les systèmes hybrides rectangulaires (définition 6.4, page 135). Cet algorithme est du à Kourjanski et Varaiya [71]. Rappelons que les AFR Hybrides Linéaires sont rectangulaire (remarque 6.14, page 141).

Considérons une chaîne hybride $[\gamma_1, \rho_1, \delta_1], \ldots, [\gamma_k, \rho_k, \delta_k]$. Du fait que les γ_i, ρ_i et δ_i sont des rectangles, la modélisation de la transformation induite par les triplets $[\gamma_i, \rho_i, \delta_i]$ de la définition 6.19 (page 147) peut être donnée sous la forme d'un système linéaire. Dans la suite, $j \in \{1, \ldots, n\}$ représente l'indice dans la dimension du SFR Hybride Linéaire. Par définition d'une chaîne hybride rectangulaires :

$$\gamma_i = \prod_{j=1}^{n} [\underline{\gamma_i}_j ; \overline{\gamma_i}_j] \quad \text{et} \quad \rho_i = \prod_{j=1}^{n} [\underline{\rho_i}_j ; \overline{\rho_i}_j] \quad \text{et} \quad \delta_i = \prod_{j=1}^{n} [\underline{\delta_i}_j ; \overline{\delta_i}_j]$$

L'ensemble des valuations qui permettent de franchir le triplet de rang i est défini par γ_i. Par conséquent, l'ensemble de ces valuations $V(i)$, est donné, pour la dimension j, par :

$$\underline{\gamma_i}_j \leq V(i)_j \leq \overline{\gamma_i}_j \tag{6.2}$$

Puis, l'ensemble de valuations $V(i)$ est obtenu à partir de celui de rang $V(i-1)$ (uniquement pour $2 \leq i \leq k$), en appliquant la transformation discrète donnée par ρ_{i-1}, puis l'évolution continue donnée par δ_i. Dans le cadre d'une chaîne hybride rectangulaire, la transformation discrète de rang i, est une application affine définie par la matrice $n \times n$, $A(i)$ de $\mathbb{R}$ et le vecteur ρ_i. La matrice $A(i)$ est telle que tous ses éléments non diagonaux sont nuls, et :

- si l'élément de dimension j, $A(i)_{jj}$, est nul, x_j est réinitialisée lors du passage du triplet de rang i à celui de rang $i+1$.
- sinon, l'élément diagonal de dimension j vaut 1 : $A(i)_{jj} = 1$, et $\underline{\rho_i}_j = \overline{\rho_i}_j = 0$: la valeur de x_j reste constante.

En notant $V'(i)$ l'ensemble des valuations obtenues depuis $V(i-1)$ suite à l'application de cette transformation discrète, nous avons, pour la dimension j :

$$V'(i)_j = A(i-1)_{jj}.V(i-1)_j + \rho_{i-1j} \qquad \text{avec} \qquad \underline{\rho_{i-1}}_j \leq \rho_{i-1j} \leq \overline{\rho_{i-1}}_j$$

Enfin, l'évolution continue définie par δ_i, conduit, pour un instant $t(i) \geq 0$, de passage du triplet de rang i à celui de rang $i+1$, à l'ensemble des valuations $V(i)$:

$$V'(i)_j + \underline{\delta_i}_j.t(i) \leq V(i) \leq V'(i) + \overline{\delta_i}_j.t(i)$$

Ces deux étapes peuvent maintenant s'écrire en une seule par la formulation :

$$A(i-1)_{jj}.V(i-1)_j + \underline{\delta_i}_j.t(i) \leq V(i)_j - \rho_{i-1j} \leq A(i-1)_{jj}.V(i-1)_j + \overline{\delta_i}_j.t(i) \tag{6.3}$$

avec :

$$\underline{\rho_{i-1}}_j \leq \rho_{i-1j} \leq \overline{\rho_{i-1}}_j \tag{6.4}$$

Les équations (6.2) à (6.4) définissent les conditions pour le franchissement de la chaîne. Lorsqu'un point fixe est recherché, il suffit alors d'imposer que la valeur de chaque variable de X_{R_H} soit stable par l'exécution de la chaîne en ajoutant la condition :

$$V(1)_j = V(k)_j, \; j \in \{1, \ldots, n\} \tag{6.5}$$

Les équations (6.2) à (6.5) forment alors un problème de programmation linéaire qui admet une solution si et seulement si la chaîne hybride admet un point fixe.

Lemme 6.23 (Kourjanski & Varaiya [71])
L'ensemble des points fixes d'une chaîne hybride rectangulaire peut être calculé par la résolution d'un problème de programmation linéaire. ♦

Le lecteur trouvera, par exemple, dans [68] une méthode efficace pour la résolution des probèmes de programmation linéaire.

Chapitre 7

Réduction du modèle de vérification par méthodes d'ordre partiel

Lorsque nous avons la certitude que le système modélisé est borné, c'est à dire lorsqu'il remplit notre critère de bonne spécification (voir le chapitre 2, page 33), il est possible de s'intéresser à la vérification d'autres propriétés du système. Il peut par exemple s'agir de s'assurer que le système des lecteurs/écrivains décrit au chapitre 2 ne possède pas d'état bloquant. Bien souvent, ces vérifications sont pratiquement impossibles à cause de problème de *l'explosion combinatoire de l'espace d'états* : le modèle du système contient trop d'états, et ne peut être exploré de façon exhaustive, généralement parce que l'espace de stockage nécessaire à cette opération n'est pas disponible. Ce problème se produit pour les systèmes à file réactifs lorsque la borne sur le nombre d'occurrences mémorisées simultanément est trop grande : en notant n le nombre d'événements mémorisables (à mémorisation unique) et b cette borne, il y a $\frac{n!}{(n-b)!}$ contenus de file possibles de taille b. Nous présentons donc dans ce chapitre une technique de réduction du modèle de vérification des programmes ELECTRE.

Nous présentons d'abord ce modèle en section 7.1, en particulier, nous montrons comment il est possible de le réduire. Notre technique de réduction [61] est basée sur les *méthodes d'ordre-partiel* [82, 93, 50, 48, 69, 79, 49, 46] que nous introduisons en section 7.2. Finalement, la section 7.3 présente le principe et l'agorithme de calcul du modèle réduit, les résultats de vérification qui sont préservés par notre méthode de réduction, ainsi qu'une analyse de la complexité du modèle obtenu. Dans cette section, nous montrons, en particulier, que notre technique de réduction préserve l'accessibilité des configurations stables du système (modulo une relation d'équivalence nommée *trace*, c.f. la définition 7.3, page 158), ce qui permet de transposer la plupart des résultats de vérification du modèle réduit au modèle complet.

7.1 Le modèle de vérification des programmes Electre

Du fait qu'il n'existe pas de model-checker dédié à ELECTRE ou aux automates à file réactifs, le modèle de vérification doit contenir toutes les informations nécessaires au respect de la sémantique du langage, en particulier un modèle pour la file de mémorisation et le respect de la gestion FIFFO des occurrences d'événements mémorisées. Lors de la compilation d'un programme ELECTRE, nous obtenons un modèle comportemental du programme. Il reste donc à modéliser la file de mémorisation.

7.1.1 Séparation du contrôle et des données

Par exemple, pour la vérification de propriétés du programme des lecteurs/écrivains représenté en figure 2.1, page 35, il nous faut en plus de l'AFR (modèle comportemental) de la figure 2.4, représenter la file de mémorisation, ainsi que la synchronisation entre les deux, en respect de la politique FIFFO. La file de mémorisation est représentée par un système de transitions dont les états sont les contenus de file et les transitions sont les opérations (ajout et retrait) sur les occurrences mémorisées. La figure 7.1 (page 153) représente le modèle de la file pour le programme des lecteurs/écrivains (figure 2.1, page 35 et figure 2.4, page 43) lorsque nous supposons qu'elle est de taille maximale 3. Par soucis de lisibilité, nous n'avons pas représenté les transitions de retrait des occurrences mémorisées. Le système complet contient donc en chaque état une transition de retrait pour chaque occurrence d'événement consommable. Par exemple, dans l'état $\mathtt{r_1wr_1}$, il y a une transition de retrait pour l'événement $\mathtt{r_1}$ qui amène dans l'état $\mathtt{wr_1}$, et une transition de retrait pour l'événement $\mathtt{w}$ qui conduit dans l'état $\mathtt{r_1r_1}$. Bien entendu, la seconde occurrence mémorisée de $\mathtt{r_1}$ ne peut pas être consommée dans cet état.

Le modèle complet, noté $R \,||\, F$, est alors obtenu en synchronisant les transitions de l'automate à file réactif R et du modèle de la file F. Le système de transitions ainsi obtenu correspond exactement au système à file réactif obtenu depuis R par la définition 2.11 (page 44). Chaque transition de mémorisation de R est associée à la transition d'ajout correspondante dans F. Par exemple, si l'état courant de R est W et celui de F est $\mathtt{r_2r_1}$, lorsqu'une occurrence de $\mathtt{w}$ survient, les transitions $W \xrightarrow{!\mathtt{w}}_R W$ et $\mathtt{r_2r_1} \xrightarrow{+\mathtt{w}} \mathtt{r_2r_1w}$ sont franchies simultanément dans R et F. Cela correspond au franchissement de la transition $(W, \mathtt{r_2r_1}) \xrightarrow{!\mathtt{w}}_S (W, \mathtt{r_2r_1w})$ dans le système à file réactif S défini par R.

La synchronisation a lieu de même pour les transitions de consommation/retrait. Dans ce dernier cas, il faut en plus s'assurer du respect de la gestion FIFFO de la file : c'est à dire consommer l'occurrence mémorisée la plus ancienne. Ceci est réalisé d'une part en étiquetant les transitions de consommation par la position de l'événement dans la file, et d'autre part en spécifiant des vecteurs de synchronisation, à la Arnold et Nivat, qui assurent l'évolution correcte du système. Par exemple, les transitions de consommation issues de l'état $\mathtt{r_1r_2}$ du modèle de la figure 7.1 sont étiquetées $(-\mathtt{r_1})_1$ et $(-\mathtt{r_2})_2$, alors que celles issues de l'état $\mathtt{r_2r_1}$ sont étiquetés à l'inverse : $(-\mathtt{r_1})_2$ et $(-\mathtt{r_2})_1$.

Enfin, pour être complet, le modèle de la file dispose aussi d'un événement particulier, noté τ, qui étiquette une transition qui boucle sur chaque état, et qui sert à la synchronisation avec les transitions de l'AFR sans effet sur la file. De la même façon, il y a une transition boucle, de mémorisation, en chaque état, et pour chaque événement à mémorisation unique qui y apparaît. Ces transitions de mémorisation sont nécessaire pour la synchronisation avec les AFR, lorsque ceux-ci tentent de mémoriser un événement à mémorisation unique qui l'est déjà. Dans la suite, nous faisons abstraction de transitions, ainsi que des transitions τ (qui n'interviennent pas lors de la réduction) excepté dans l'algorithme 7.5 page 170 qui explicite la construction du modèle réduit.

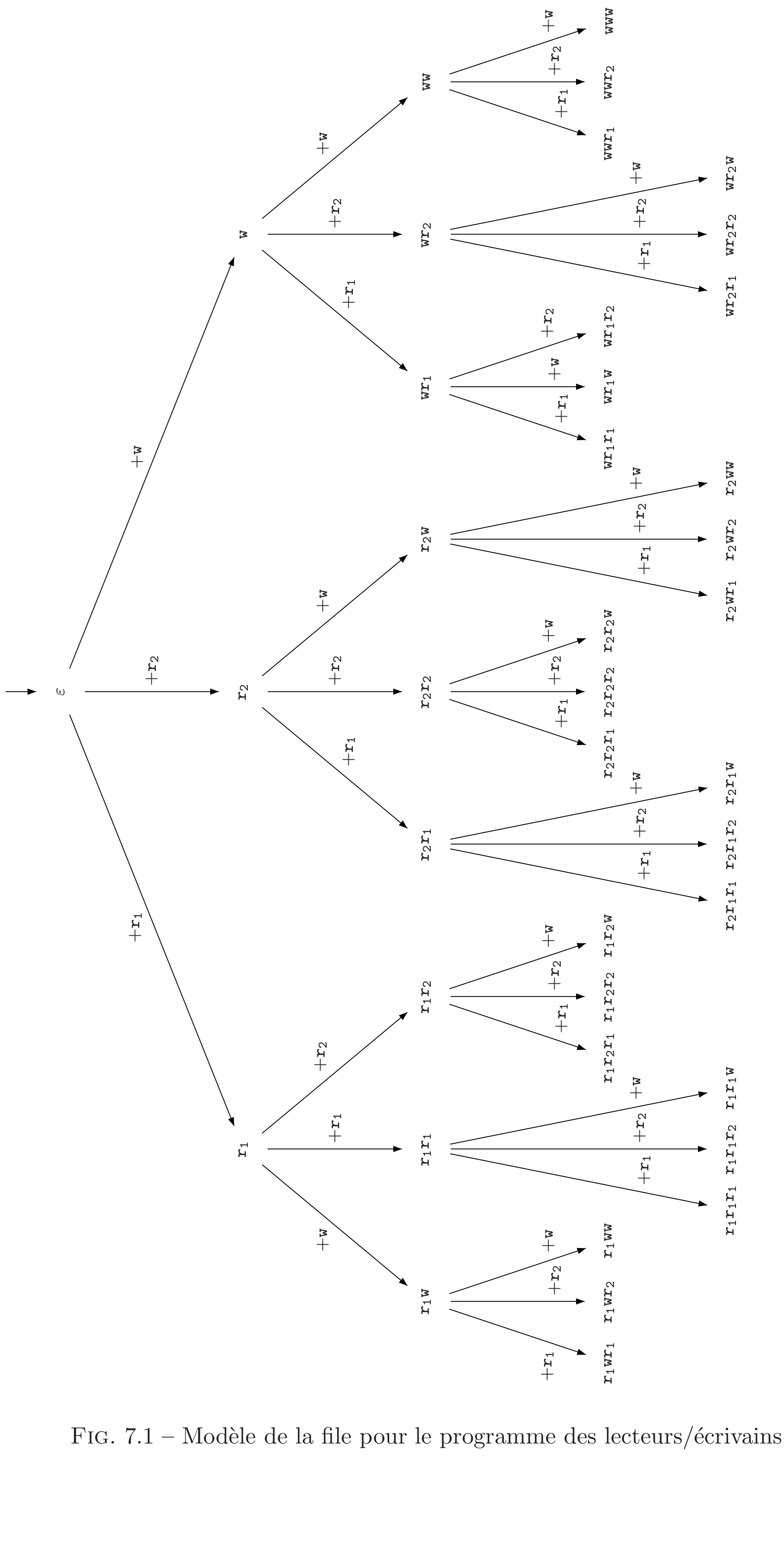


FIG. 7.1 – Modèle de la file pour le programme des lecteurs/écrivains.

Lorsque R et F sont plongés dans un environnement E, il faut de plus tenir compte de cette contrainte en synchronisant les trois systèmes. La synchronisation de R et F avec E se fait de la même manière que pour les systèmes à file réactifs embarqués (définition 2.21, page 51) c'est pourquoi nous ne développons pas plus ce point.

7.1.2 Possibilités de réduction

Reprenons l'exemple des lecteurs/écrivains de la figure 2.4 (page 43) et considérons que la file contient $\mathtt{r}_2\mathtt{r}_1$ alors que l'automate à file réactif R se trouve dans l'état $\varnothing$. Il est possible de consommer ces deux occurrences, et R franchit les transitions : $\varnothing \xrightarrow{?\mathtt{r}_2}_R R_2$ et $R_2 \xrightarrow{?\mathtt{r}_1}_R R_1 \,||\, R_2$. Lorsque la file contient $\mathtt{r}_1\mathtt{r}_2$ (donc dans l'ordre inverse), les transitions franchies par R depuis $\varnothing$ sont : $\varnothing \xrightarrow{?\mathtt{r}_1}_R R_1$ et $R_1 \xrightarrow{?\mathtt{r}_2}_R R_1 \,||\, R_2$. Nous remarquons que l'état finalement atteint, $R_1 \,||\, R_2$, est indépendant de l'ordre dans lequel $\mathtt{r}_1$ et $\mathtt{r}_2$ avaient été mémorisés dans la file. Ainsi, en tenant compte de la spécificité des AFR où les occurrences mémorisées sont prioritaires, cet ordre n'influence par exemple pas la recherche de configurations (stables) puits dans le système. Du point de vue de l'environnement du système (c'est à dire d'un observateur quelconque), les entrelacements $\mathtt{r}_1\mathtt{r}_2$ et $\mathtt{r}_2\mathtt{r}_1$ donnent lieu à des comportements indistinguables, puisque les états stables des deux séquences sont les mêmes. Dans la suite, nous mettons particulièrement l'accent sur les configurations stables, plutôt que sur les configurations instables, car seules les premières correspondent à des états du système réel modélisé : les configurations instables permettent de modéliser simplement le traitement des occurrences mémorisées, mais ne correspondent à aucun état du système réel (une séquence de stabilisation est généralement considérée comme une seule et même étape dans un système réel).

Par contre, il est clair que l'ordre entre $\mathtt{r}_1$ et $\mathtt{w}$ est important puisque le fait de traiter l'une des occurrences ne permet pas de traiter l'autre. S'il n'y avait par exemple pas de transition qui va de W à $\varnothing$ dans notre exemple, il pourrait être impossible de traiter les occurrences mémorisées de $\mathtt{r}_1$ suite à la consommation d'une occurrence de $\mathtt{w}$.

Nous cherchons donc à construire un modèle réduit pour la file qui tient compte de ces équivalences en réunissant les états du modèle complet qui produisent des exécutions indistinguables (vis à vis des configurations stables). Le modèle réduit de la file pour le programme des lecteurs/écrivains, noté F_r, est représenté en figure 7.2. Il ne contient plus que 33 états contre 40 pour le modèle complet de la figure 7.1. Les états marqués entre crochets remplacent plusieurs états du modèle initial.

Bien entendu, les transitions de retrait d'occurrences mémorisées doivent être adaptées à la réduction : par exemple pour l'état $[\mathtt{r}_1\mathtt{r}_2\mathtt{r}_2]$ qui représente l'ensemble d'états $\{\mathtt{r}_1\mathtt{r}_2\mathtt{r}_2, \mathtt{r}_2\mathtt{r}_1\mathtt{r}_2, \mathtt{r}_2\mathtt{r}_2\mathtt{r}_1\}$, il faut considérer que $\mathtt{r}_1$ peut être dans chacune de ces trois positions lors du retrait. Il y a donc trois transitions étiquetées $(-\mathtt{r}_1)_1$, $(-\mathtt{r}_1)_2$ et $(-\mathtt{r}_1)_3$ qui amènent dans l'état $\mathtt{r}_2\mathtt{r}_2$ et deux transitions d'étiquettes[1] $(-\mathtt{r}_2)_1$ et $(-\mathtt{r}_2)_2$ qui amènent dans l'état $[\mathtt{r}_1\mathtt{r}_2]$ qui représente $\mathtt{r}_1\mathtt{r}_2$ et $\mathtt{r}_2\mathtt{r}_1$. Ceci entraîne l'introduction de l'indéterminisme dans le modèle de la file réduit, et donc dans $R \,||\, F_r$ (le modèle obtenu en faisant le produit synchronisé de R et de F_r, comme nous l'avons décrit dans la section précédente).

[1] Il n'y a pas de transition étiquetée $(-\mathtt{r}_2)_3$ puisqu'une occurrence de $\mathtt{r}_2$ en troisième position est forcément précédée d'une autre occurrence de $\mathtt{r}_2$.

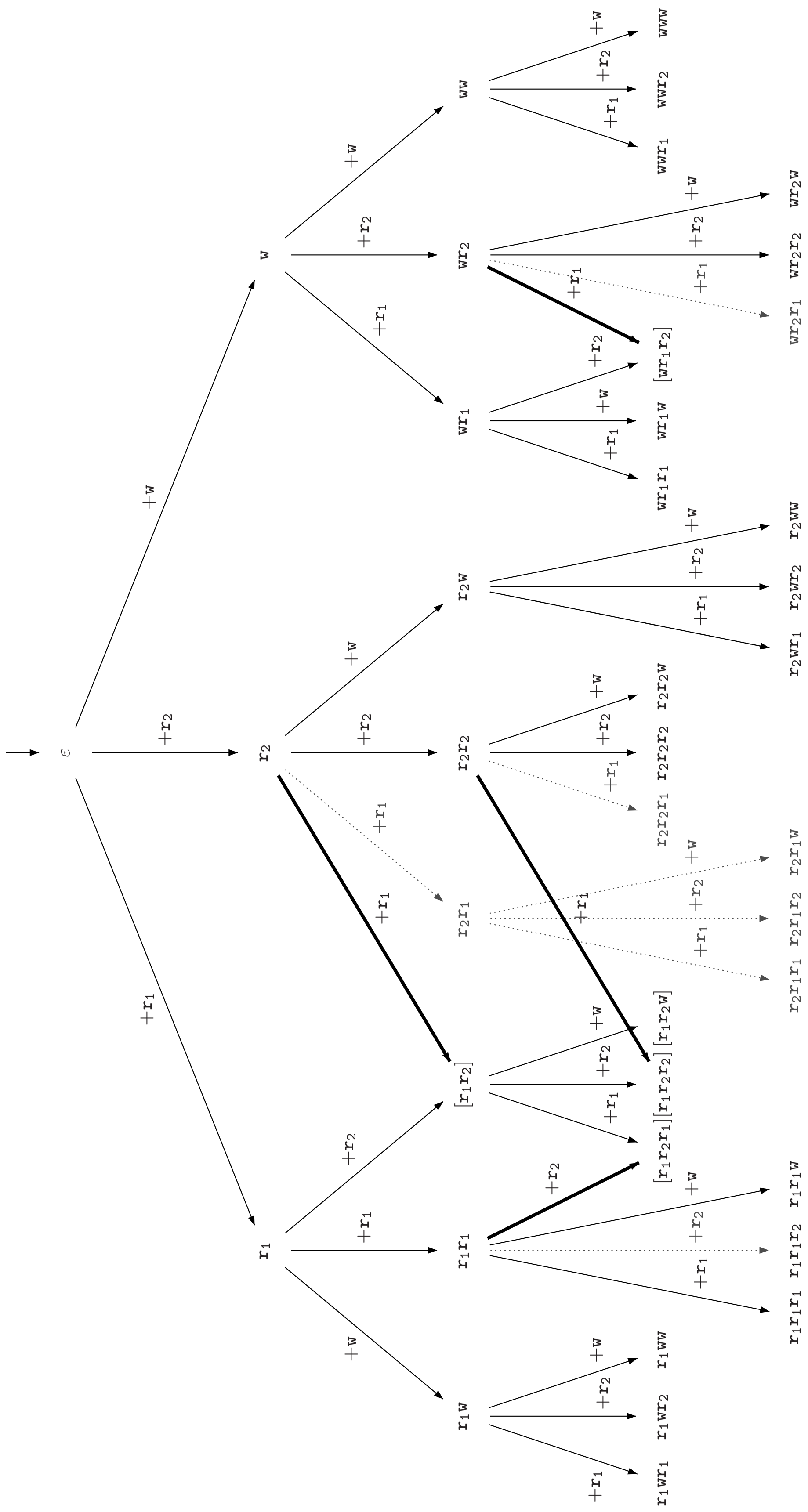


FIG. 7.2 – Modèle réduit de la file pour les lecteurs/écrivains.

Lemme 7.1
$R \,||\, F_r$ dispose d'au moins tous les comportements de $R \,||\, F$ par l'indéterminisme sur le retrait des occurrences mémorisées : $[\![R \,||\, F]\!] \subseteq [\![R \,||\, F_r]\!]$. ♦

Nous montrons dans la suite de ce chapitre comment mettre à profit cette possibilité de réduction au moyen des *méthodes d'ordre partiel*.

7.2 Les méthodes d'ordre partiel

Les *méthodes d'ordre partiel* [82, 93, 50, 48, 69, 79, 49, 46] ont été introduites pour répondre au problème de l'explosion combinatoire. Comme nous le montrons maintenant, pour la vérification de certaines propriétés basée sur l'exploration de l'espace d'états d'un système de transitions, elles permettent de ne visiter que certains états et/ou de ne franchir que certaines transitions, entraînant ainsi un gain en espace comme en temps. Le choix des transitions exécutées est basé sur la notion de *dépendance* (définition 7.2) qui permet d'exprimer l'effet de l'exécution d'une transition sur la possibilité d'en franchir une autre. Nous introduisons ici les méthodes d'ordre partiel dans le contexte de la recherche des états puits[2] d'un système de transitions fini. Il est possible de s'intéresser à d'autres types de propriétés comme par exemple la vérification de la satisfaction de formules LTL : les méthodes d'ordre partiel sont massivement utilisées, par exemple dans le model-checker SPIN [63], pour la vérification de propriétés de protocoles de communication [47, 64, 63], mais l'utilisation que nous faisons de ces méthodes n'oblige pas à s'aventurer aussi profondément dans leur description. Cette section est basée sur [46] auprès duquel le lecteur pourra trouver une référence sur les méthodes d'ordre partiel.

Dans cette section, nous prenons l'exemple de l'algorithme 7.1 qui explore l'espace d'états d'un système de transitions fini S, en profondeur d'abord. P et T y représentent respectivement

```
P ← ∅; T ← ∅
P.push q0
tant que P ≠ ∅ faire
  q ← P.pop
  si q ∉ T alors
    T.add q
    pour chaque l ∈ out(q) faire
      q' ← Succ^l(q)
      P.push q'
    fin pour
  fin si
fin tant que
```

Algorithme 7.1: Agorithme d'exploration de $RS(S)$ en profondeur d'abord.

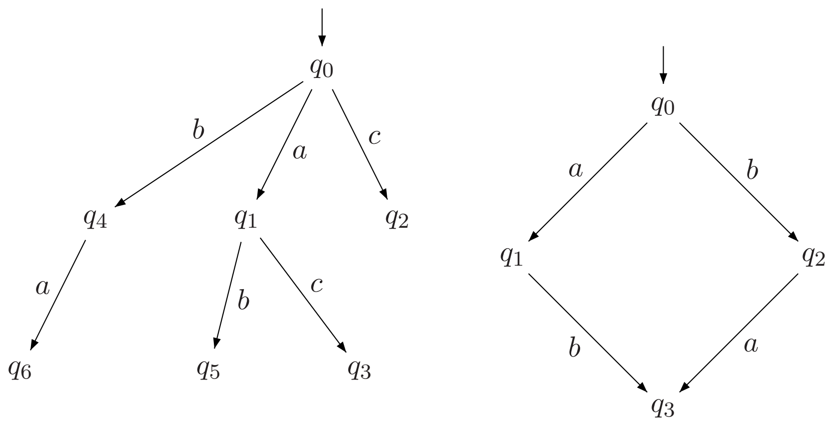
la pile des états à traiter et la table des états déjà visités. Nous nous plaçons donc maintenant dans le contexte de l'utilisation de cet algorithme pour la recherche d'états puits d'un système de transitions fini et nous montrons comment les méthodes d'ordre partiel permettent de l'optimiser.

[2]Les états q du système tels que $out(q) = \emptyset$.

7.2.1 Relation de dépendance d'un système de transitions

Définition et propriétés d'une relation de dépendance

Prenons pour exemple les systèmes de transitions de la figure 7.3. Depuis l'état q_0 de la



(a) Système de transitions S_1. (b) Système de transitions S_2.

FIG. 7.3 – Dépendance et indépendance des transitions.

figure 7.3(b), nous voyons que l'ordre d'exécution des transitions d'étiquettes a et b n'a aucune influence sur l'accessibilité de l'état q_3 : les séquences de transitions étiquetées ab et ba depuis q_0 mènent toutes deux à q_3. À l'inverse, dans le système de transitions S_1 de la figure 7.3(a), nous voyons que ce n'est pas le cas des transitions d'étiquettes a, b et c en q_0. En effet, l'ordre de franchissement des transitions a et c en q_0 est important puisque si après avoir franchi a il est toujours possible d'exécuter c, l'exécution de c *inhibe* l'exécution de a : $Succ^a(q_2) = \emptyset$. Les actions a et b ne s'inhibent pas, mais un problème différent se pose : les séquences ab et ba toutes deux exécutables depuis q_0 ne mènent pas dans le même état : q_5 pour ab et q_6 pour ba. Nous dirons qu'elles n'ont pas le même *effet*. Ainsi, les actions a et b sont *indépendantes* dans S_2 (figure 7.3(b)) et a et b, d'un côté, a et c de l'autre sont *dépendantes* dans S_1 (figure 7.3(a)). Notons au passage que b et c sont aussi dépendantes dans S_1.

Définition 7.2 (Relation de dépendance [69])
Soit $S = (Q, q_0, L, \rightarrow)$ un système de transitions étiqueté. Une relation $\mathcal{D} \subseteq L \times L$ réflexive et symétrique est une *relation de dépendance* pour S ssi quelles que soient $l, l' \in L$, alors $(l, l') \notin \mathcal{D}$ (l et l' sont *indépendantes*) si et seulement si :

1. quel que soit $q \in Q$, si $l \in out(q)$ et $q \xrightarrow{l} q'$ alors $l' \in out(q')$ ssi $l' \in out(q)$ (deux transitions indépendantes ne peuvent ni s'activer, ni s'inhiber),
2. si $l, l' \in out(q)$ alors il existe un unique état q' tel que $q \xrightarrow{l.l'} q'$ et $q \xrightarrow{l'.l} q'$ (commutativité des transitions indépendantes).

♦

Deux étiquettes $l, l' \in L$ sont *dépendantes* si $(l, l') \in \mathcal{D}$ et *indépendantes* sinon. La *relation d'indépendance* $\mathcal{I}$ est définie par $\mathcal{I} = (L \times L) \setminus \mathcal{D}$. Dans la suite, nous omettons les couples (l, l')

liés à la réflexivité et la symétrie de $\mathcal{D}$ qui doivent donc être ajoutés aux relations décrites ici pour obtenir les relations complètes. Nous n'indiquons pas de méthode générale pour détecter $\mathcal{D}$: le lecteur trouvera dans [46] une approche de ce problème, et nous fournirons une méthode de calcul de $\mathcal{D}$ pour les programmes Electre en section 7.3.1. Notons cependant que la taille de $\mathcal{D}$ est un facteur clef pour obtenir l'effet optimal des méthodes d'ordre partiel. Par conséquent, le calcul de $\mathcal{D}$ repose fortement sur le modèle considéré.

Définition 7.3 (Trace [78])
Soit $w = l_1 l_2 \dots l_i l_{i+1} \dots l_n$ une séquence d'étiquettes et $\mathcal{D}$ une relation de dépendance. La *trace* $[w]$ représente l'ensemble des séquences d'étiquettes obtenues depuis w en permutant deux étiquettes indépendantes (vis-à-vis de $\mathcal{D}$) et adjacentes : $l_1 l_2 \dots l_{i+1} l_i \dots l_n$ avec $(l_i, l_{i+1}) \notin \mathcal{D}$. ♦

Les traces permettent donc de définir des classes d'équivalence de séquences d'étiquettes relativement à $\mathcal{D}$, dans la suite, la relation $\mathcal{D}$ n'est pas toujours explicitée. Ainsi, pour le système S_1 de la figure 7.3(a), $\mathcal{D} = \{(a,b),(a,c),(b,c)\}$, et les traces depuis q_0 sont : $[ba]$, $[ab]$, $[ac]$ et $[c]$. Pour le système S_2 de la figure 7.3(b), $\mathcal{D} = \emptyset$, et la trace $[ab] = \{ab, ba\}$ capture l'indépendance de a et de b.

Lemme 7.4 (Godefroid [46])
Soit q un état de S. Si $q \xrightarrow{\sigma_1} q_1$ et $q \xrightarrow{\sigma_2} q_2$, et si $[\sigma_1] = [\sigma_2]$, alors $q_1 = q_2$. ♦

Ce résultat est fondamental puisqu'il permet d'exécuter uniquement σ_1 depuis q pour obtenir l'accessibilité de $q_1 = q_2$, au lieu de toutes les séquences de $[\sigma_1]$.

Algorithme de recherche sélective

Lors de l'exploration du système S_2 de la figure 7.3(b), il apparaît bien inutile de franchir les deux entrelacements ab et ba pour atteindre dans les deux cas le même état q_3 :le franchissement de la transition $q_2 \xrightarrow{a} q_e$ n'apporte aucune nouvelle information pour l'accessibilité de q_3. En particulier, pour la recherche d'un état puits q_p, accessible depuis q_0, par une séquence de transitions étiquetée σ, n'importe quelle séquence $\sigma' \in [\sigma]$ conduit elle aussi à q_p par le théorème 7.4. Il suffit donc d'explorer une et une seule des séquences $\sigma' \in [\sigma]$ pour trouver q_p.

Afin de parvenir à ce but, il faut modifier l'algorithme d'exploration de l'espace d'états (algorithme 7.1) pour qu'en chaque état q rencontré, seules certaines des transitions de $out(q)$, notées $T(q)$, soit franchies et non pas $out(q)$ en entier. Dans les sections 7.2.2 et 7.2.3 nous montrons comment construire $T(q)$ de façon à préserver l'accessibilité des états puits.

7.2.2 Méthodes à base d'ensembles persistants

Définition et exploration sélective

Dans cette première méthode, l'ensemble T des transitions sélectionnées est appelé *ensemble persistant*. Cela vient du fait que si depuis un état q, seules des transitions qui n'appartiennent pas à $T(q)$ sont franchies, alors toutes les transitions de $T(q)$ sont à nouveau franchissables dans les états ainsi atteints (l'exécutabilité des transitions de $T(q)$ est persistante).

Définition 7.5 (Ensemble persistant [49])
Un ensemble d'étiquettes $T(q)$ est *persistant* dans l'état q ssi

1. $T(q) \subseteq out(q)$ i.e. toute transition de $T(q)$ est franchissable depuis q,
2. et, pour toute séquence non vide de transitions :

$$q = q_1 \xrightarrow{l_1} q_2 \xrightarrow{l_2} \dots q_n \xrightarrow{l_n} q_{n+1}$$

depuis q telle que $\forall i \in \{1, \dots, n\},\ l_i \notin T(q)$, l_n est indépendante de toute transition de $T(q)$ en q_n.

♦

L'algorithme 7.2 (basé sur l'algorithme 7.1) présente une méthode d'exploration sélective de l'espace d'états d'un système de transitions S en ne franchissant dans un état q donné que les transitions qui appartiennent à l'ensemble persistant en q (noté $persistant(q)$).

```
P ← ∅; T ← ∅
P.push q0
tant que P ≠ ∅ faire
  q ← P.pop
  si q ∉ T alors
    T.add q
    pour chaque l ∈ persistant(q) faire
      q' ← Succ^l(q)
      P.push q'
    fin pour
  fin si
fin tant que
```

Algorithme 7.2: Agorithme d'exploration sélective de $RS(S)$ à base d'ensembles persistants.

Pour se convaincre que cet algorithme visite bien tous les états puits de S, il suffit de remarquer que si q_p est un état puits accessible depuis q_0 par la séquence de transitions étiquetée σ (de longueur n) dans S, alors il existe $\sigma' \in [\sigma]$ telle que t', sa première étiquette, appartient à $persistant(q_0)$. En effet, puisque $out(q_p) = \emptyset$, il est nécessaire que l'une des étiquettes de σ appartienne à $persistant(q_0)$. Sinon, par la définition 7.5, les transitions de $persistant(q_0)$ sont franchissables depuis q_p qui n'est donc pas un état puits. Ensuite, en notant l_k la première étiquette de σ qui appartient à $persistant(q_0)$, la séquence $l_k l_1 \dots l_{k-1} l_{k+1} \dots l_n$ appartient à $[\sigma]$ puisque par la définition 7.5, $(l_j, l_k) \notin \mathcal{D}$ quel que soit $j \in \{1, \dots, k\}$. Finalement, l'accessibilité de q_p depuis q_0 par σ' est obtenue par induction sur la longueur de σ en utilisant donc le fait qu'il existe une séquence dans $[\sigma]$ dont la première transition est dans $persistant(q_0)$.

La méthode des ensembles persistants permet donc de ne visiter qu'un sous-ensemble de Q et donc, de ne franchir qu'un sous-ensemble des transitions de S. L'utilisation de cette méthode laisse donc envisager un gain de temps et d'espace lors de l'exploration de l'espace d'états.

Calcul des ensembles persistants

Notons que $out(q)$ est un ensemble persistant trivial en q. Dans [46], l'auteur présente quatre méthodes pour le calcul des ensembles persistants. Nous n'en présentons ici qu'une seule à titre d'indication. Le lecteur intéressé pourra se tourner vers [82, 93, 50, 48, 49, 46] pour d'autres techniques plus raffinées.

Pour le calcul des ensembles persistants, nous utilisons le critère de [50], connu sous le nom de (sélection des) "transitions conflictuelles". Le calcul de $persistant(q)$ est initié en choisissant une étiquette $l \in out(q)$, et en posant $persistant(q) = \{l\}$. Puis, toutes étiquettes l' telles que $(l, l') \in \mathcal{D}$ avec $l \in persistant(q)$, sont ajoutées à $persistant(q)$ jusqu'à saturation. Si une étiquette l' telle que $l' \notin out(q)$ est ajoutée à $persistant(q)$, la procédure s'arrête et $persistant(q) = out(q)$. Notons que le calcul des ensembles persistants ne nécessite que des informations statiques du modèle : un ensemble persistant ne dépend pas de l'exploration en cours.

Considérons le système de transitions de la figure 7.4, dont la relation de dépendance est $\mathcal{D} = \{(b, c)\}$. Par la procédure décrite ci-dessus, et en choisissant b comme première étiquette,

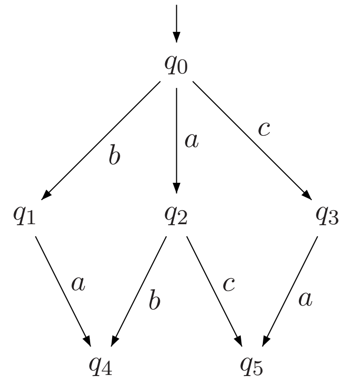


Fig. 7.4 – Système de transitions S_3.

nous avons $persistant(q_0) = \{b, c\}$. La figure 7.5 représente la partie explorée du système de transitions S_3. Les parties grisées et pointillées montrent les parties qui ont été ignorées. Ce

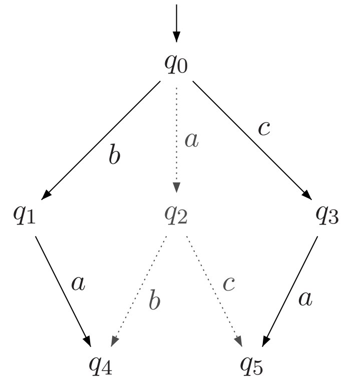


Fig. 7.5 – Exploration sélective de S_3 par l'utilisation d'ensembles persistants.

nouvel algorithme permet donc d'éviter d'explorer q_2 (économisant ainsi de l'espace dans la pile P) ainsi que trois transitions (économisant du temps). Notons que si c est l'étiquette d'initialisation de $persistant(q)$, l'ensemble obtenu pour q_0 est le même, mais si à l'inverse c'est a qui est sélectionnée, la partie explorée de S_3 est exactement la partie grisée de la figure 7.5.

7.2.3 Méthode des ensembles endormis

À l'inverse des ensembles persistants, les ensembles endormis [51, 45] ne permettent pas d'éviter certains états lors de l'exploration, mais seulement certaines transitions. Tous les états du système de transitions sont visités. Reprenons le système de transitions de la figure 7.3(b), lors de son exploration (avec l'algorithme classique), il n'est pas nécessaire de franchir la transition $q_2 \xrightarrow{a} q_3$ si q_3 a déjà été atteint par la transition $q_1 \xrightarrow{b} q_3$ pour savoir que q_3 est accessible. Les ensembles endormis ont donc pour rôle de mémoriser les transitions qu'il n'est pas nécessaire de franchir : elles sont "endormies" lors de l'exploration.

Calcul des ensembles endormis

L'ensemble endormi associé à l'état initial q_0 de S est initialement vide. Lors du franchissement d'une transition $q \xrightarrow{l} q'$, l'ensemble endormi de q' est calculé depuis celui de q de la façon suivante :

1. $endormi(q') := \{l' \in endormi(q) \,|\, (l, l') \notin \mathcal{D}\}$ ($endormi(q')$ est constitué des étiquettes de $endormi(q)$ qui sont indépendantes de l),
2. $endormi(q) := endormi(q) \cup \{l\}$ (l vient d'être franchie, donc elle doit être endormie dans les successeurs de q atteint par des transitions indépendantes de l).

À la différence des ensembles persistants, le calcul des ensembles endormis repose sur le passé de l'exploration de S en cours. En particulier, l'ensemble endormi associé à un état donné dépend de l'ordre dans lequel les successeurs d'un état donné sont visités comme l'indique le point (2) ci-dessus.

Il est clair que cette méthode de réduction est valide pour la recherche d'états puits puisque tous les états du système de transitions sont visités.

Exploration sélective à base d'ensembles endormis

L'algorithme 7.3 présente un algorithme d'exploration de l'espace d'états d'un système de transitions, optimisé par l'utilisation d'ensembles endormis. $Explore$ représente l'ensemble des transitions qui doivent être explorées depuis q. Lorsque l'état q est à nouveau visité, il est éventuellement nécessaire d'explorer des transitions qui étaient endormies lors du premier passage et qui ne le sont plus maintenant, à cause des dépendances différentes sur les différents chemins qui mènent à q. L'état q_T représente l'occurrence de q qui est mémorisée dans T.

L'utilisation de cet algorithme sur le système de transitions de la figure 7.4 permet de visiter l'intégralité de $RS(S_3)$ sans franchir toutes les transitions de S_3 (seules les transitions pleines de la figure 7.5, page 160, sont franchies).

Dans le cadre de l'exploration de l'espace d'états s'un système de transitions, la méthode des ensembles endormis permet d'économiser du temps de calcul en franchissant moins de transitions.

$P \leftarrow \emptyset$; $T \leftarrow \emptyset$; $endormi(q_0) \leftarrow \emptyset$
$P.push\ q_0$
tant que $P \neq \emptyset$ **faire**
  $q \leftarrow P.pop$
  si $q \notin T$ **alors**
    $T.add\ q$
    $Explore \leftarrow (out(q) \setminus endormi(q))$
  sinon
    $Explore \leftarrow (endormi(q_T) \setminus endormi(q))$
    $endormi(q) \leftarrow endormi(q) \cap endormi(q_T)$
    $endormi(q_T) \leftarrow endormi(q)$
  fin si
  pour chaque $l \in Explore$ **faire**
    $q' \leftarrow Succ^l(q)$
    $endormi(q') = \{l' \in endormi(q) \mid (l, l') \notin \mathcal{D}\}$
    $P.push\ q'$
    $endormi(q) = endormi(q) \cup \{l\}$
  fin pour
fin tant que

Algorithme 7.3: Agorithme d'exploration sélective de $RS(S)$ à base d'ensemble endormis.

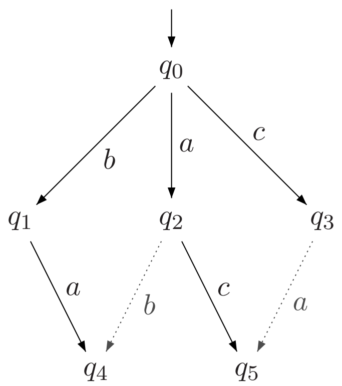


FIG. 7.6 – Exploration sélective de S_3 par l'utilisation d'ensembles endormis.

7.2.4 Remarques sur les méthodes d'ordre partiel

Dans cette section, nous avons principalement cité les travaux d'Overman [82], de Valmari [93], de MacMillan [79] et de Godefroid, Pirottin et Wolper [45, 50, 48, 49, 46]. Il faut y ajouter ceux de Katz et Peled [69, 70] sur les *faithfull decompositions* qui s'approchent des ensembles persistants, ceux de Peled [84] sur le calcul des ensembles persistants.

Nous avons présenté les techniques à base d'ensembles persistants et d'ensembles endormis comme deux méthodes entièrement disjointes. Il est possible (et souvent souhaitable) de les utiliser ensemble pour réduire encore plus l'espace d'états exploré. En effet, il arrive qu'un ensemble persistant contienne des transitions indépendantes. C'est éventuellement le cas des ensembles persistants obtenus par la méthode de sélection des "transitions conflictuelles" [50] présentée en section 7.2.2 lorsqu'une étiquette ne correspondant pas à une transition franchissable en q est ajoutée à $persistant(q)$. La méthode des ensembles endormis permet dans ce cas d'éliminer ce problème. Le lecteur trouvera dans [51, 46] une description de l'utilisation conjointe des deux méthodes.

7.3 Réduction du modèle de vérification

Nous montrons maintenant comment les méthodes d'ordre partiel nous permettent de construire un modèle réduit pour la file comme expliqué en section 7.1.2. Nous cherchons (dans cette étude) uniquement à réduire le modèle de la file. S'il est possible d'aller plus loin (voir section 7.3.3, page 177), notre choix n'est pas anodin : puisqu'il n'existe pas de model-checker pour les automates à file réactifs, il faut modéliser la file de mémorisation (statiquement), et le modèle correspondant est souvent d'une taille démesurée par rapport à celle de l'automate à file réactif. Par exemple, l'AFR des lecteurs/écrivains a 7 états, alors que le modèle de la file a 40 états si la file est de taille 3, 283 si elle est de taille 5 et 59049 états si elle peut contenir jusqu'à 10 événements. Nous donnons tout d'abord une méthode pour le calcul de la relation de dépendance directement d'après le programme ELECTRE, puis en section 7.3.2, nous montrons la construction du modèle réduit et nous étudions, d'une part les propriétés du modèle qui sont préservées par notre technique de réduction, et d'autre part la taille (le nombre d'états) de celui-ci. Enfin, dans la dernière section, nous discutons de cette construction : en particulier, nous montrons comment elle peut être améliorée.

7.3.1 Calcul de la relation de dépendance pour les programmes Electre

La relation de dépendance est l'élément clef de la méthode de réduction. Notons que cette relation n'est pas liée au système de transitions qui modélise la file, mais à l'automate à file réactif, donc au programme ELECTRE, auquel cette file correspond. C'est en effet l'ordre dans lequel les occurrences mémorisées sont consommées qui définit cette relation.

Remarque 7.6
Alors que le modèle (classique) de la file correspond à n'importe quel programme ELECTRE ayant les mêmes événements mémorisables (à mémorisations unique et multiple), ce n'est plus le cas du modèle réduit qui utilise des informations intrinsèques de l'automate à file réactif qui lui correspond. ♦

Un programme ELECTRE peut être vu sous la forme d'un arbre d'analyse syntaxique abstrait. Ainsi, le programme des lecteurs/écrivains de la figure 2.1 (page 35) est représenté sous la forme d'un arbre en figure 7.7. Un événement `e` est de *premier niveau* vis-à-vis d'un opérateur ELECTRE donné *op*, s'il existe une branche dans l'arbre d'analyse du programme menant de *op* à `e` où il ne figure aucun événement entre *op* et `e`. Intuitivement, les événements de premier niveau sont ceux dont la dépendance ou l'indépendance est définie par l'opérateur concerné. Dans le programme des lecteurs/écrivains, $\mathtt{w}$, $\mathtt{r}_1$ et $\mathtt{r}_2$ sont de permier niveau par rapport à $|$, et $\mathtt{r}_1$ et $\mathtt{r}_2$ sont de premier niveau par rapport à $|||$. Ce programme ne contient pas d'événement masqué : tous ses événements sont de premiers niveau.

Définition 7.7
La *relation de dépendance* $\mathcal{D}$ d'un programme ELECTRE P est définie par $(\mathsf{e}, \mathsf{e}') \in \mathcal{D}$ ssi :

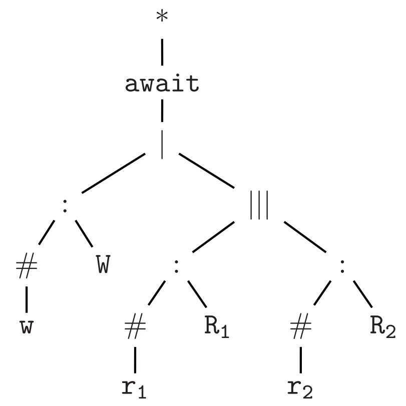


FIG. 7.7 – Arbre d'analyse syntaxique du programme ELECTRE des lecteurs/écrivains.

1. soit il existe une structure de programme $\langle A \rangle$ et une structure d'événement $\langle B \rangle$ telles que e est un événement de $\langle A \rangle$ et e′ est un événement de premier niveau de $\langle B \rangle$, et soit $\langle A \rangle$ `await` $\langle B \rangle$, soit[3] $\langle A \rangle$ `when` $\langle B \rangle$ est un sous-programme de P,
2. soit il existe deux structures d'événements $\langle A \rangle$ et $\langle B \rangle$ telles que e est un événement de $\langle A \rangle$ et e′ est un événement de $\langle B \rangle$, et $\langle A \rangle \mid \langle B \rangle$ est un sous-programme de P,
3. soit $\mathsf{e} = \mathsf{e}'$.

♦

La relation de dépendance ainsi obtenue est bien réflexive et symétrique. Dans la suite de ce chapitre, nous considérons que $\mathcal{D}$ désigne la relation de dépendance obtenue depuis un programme ELECTRE P en suivant la définition précédente.

Propriété 7.8

La relation de dépendance $\mathcal{D}$ d'un programme ELECTRE est calculée en temps linéaire en son nombre d'opérateurs. ♦

Preuve. Supposons que le programme ELECTRE est donné sous la forme de son arbre d'analyse syntaxique. Nous décorons chaque nœud de l'arbre de deux ensembles : $\mathcal{E}$ et $\mathcal{E}_1$, respectivement l'ensemble des événements et celui des événements de premier niveau du sous-arbre dont le nœud est la racine. Ces deux ensembles sont initialement vides pour chaque nœud, sauf pour les feuilles étiquettées par un événement e, auquel cas $\mathcal{E}$ et $\mathcal{E}_1$ sont initialisés avec le singleton $\{\mathsf{e}\}$. De même, initialement $\mathcal{D} = \emptyset$. Cet arbre (fini) est parcouru en profondeur d'abord, en ordre postfixe. Chaque nœud rencontré est étiqueté :

- soit par un événement e′, auquel cas, pour ce nœud, $\mathcal{E} = \mathcal{E} \cup \{\mathsf{e}'\}$ et $\mathcal{E}_1 = \{\mathsf{e}'\}$,
- sinon par un opérateur `op`, en identifiant par les exposants g et d les ensembles associés aux fils gauche et droit de l'opérateur[4] dans l'arbre :
  - si `op` est `await` ou `when`, alors $\mathcal{D} = \mathcal{D} \cup (\mathcal{E}^g \times \mathcal{E}_1^d)$,
  - si `op` est $\mid$, alors $\mathcal{D} = \mathcal{D} \cup (\mathcal{E}^g \times \mathcal{E}^d)$,

[3]L'opérateur « `await` » indique une attente forcée d'un événement : l'attente se prolonge jusqu'à ce que l'événement survienne, contrairement à « `when` ».

[4]Par convention, nous avons $\mathcal{E}^d = \mathcal{E}_1^d = \emptyset$ lorsque l'opérateur est unaire (comme "`loop`").

et quel que soit l'opérateur, $\mathcal{E} = \mathcal{E}^g \cup \mathcal{E}^d$ et $\mathcal{E}_1 = \mathcal{E}_1^g \cup \mathcal{E}_1^d$.

Il est clair que cet algorithme calcule la relation de dépendance d'un programme ELECTRE telle que décrite en définition 7.7, et en temps linéaire puisque chaque nœud de l'arbre est traité une seule fois lors du parcours postfixe (nous supposons que le temps de parcours de l'arbre est négligeable vis à vis du temps de calcul de $\mathcal{E}$, $\mathcal{E}_1$ et $\mathcal{D}$). ♦

Il reste maintenant à prouver la validité de notre relation de dépendance, où plutôt la validité de la relation d'indépendance, puisque dans le modèle de vérification par défaut, tous les événements sont considérés comme dépendants. Comme décrit en section A.2 (page 194), $\{\sigma\} \vdash P \xrightarrow{*} p \rhd P' \xrightarrow{*} p'$ indique que le programme ELECTRE p se reécrit en p' par occurrence de la séquence d'événements σ.

Théorème 7.9

Soient P un programme ELECTRE et $\sigma \in \mathcal{E}^*$ une séquence d'événements. Quel que soit $\sigma' \in [\sigma]$, si $\{\sigma\} \vdash P \xrightarrow{*} p \rhd P' \xrightarrow{*} p'$ et $\{\sigma'\} \vdash P \xrightarrow{*} p \rhd P'' \xrightarrow{*} p''$ alors $p' = p''$. ♦

Un schéma de la preuve est donné en annexe D (section D.1, page 221).

7.3.2 Construction du modèle réduit

Maintenant que nous savons comment obtenir une relation de dépendance pour un programme ELECTRE, il reste à produire un modèle réduit pour la file. Nous montrons tout d'abord quelle méthode d'ordre partiel est adaptée pour ce calcul, puis nous présentons un algorithme de construction du modèle réduit. Puis nous montrons que notre technique de réduction préserve l'accessibilité des configurations stables du système (modulo la relation d'équivalence introduite par les traces). Enfin, nous donnons quelques éléments de complexité en terme du nombre d'états du modèle réduit.

Choix de la méthode d'ordre partiel adéquate

Notre but est de produire un système de transitions réduit pour modéliser la file, où les états ne sont plus l'ensemble des contenus possibles pour celle-ci (comme en figure 7.1), mais les traces correspondant aux différents entrelacements équivalents (comme en figure 7.2). Dans le modèle réduit, chaque état est une trace, éventuellement réduite à un singleton. Si notre technique ne préserve pas les états (au sens où certains d'entre eux sont regroupés), elle doit par contre préserver tous les comportements de la file. Cette réduction peut être vue comme un *pliage* du système initial.

Le modèle réduit est calculé de la façon suivante[5] à partir de l'état ε où la file est vide :

1. à partir d'un état $[\mathtt{e_1} \dots \mathtt{e_n}]$, pour chaque événement mémorisable $\mathtt{e_{n+1}}$ (sauf les événements à mémorisation unique parmi $\mathtt{e_1}, \dots, \mathtt{e_n}$) nous calculons le successeur de $[\mathtt{e_1} \dots \mathtt{e_n}]$ par l'ajout de $\mathtt{e_{n+1}}$: $[\mathtt{e_1} \dots \mathtt{e_n} \mathtt{e_{n+1}}]$, puis :

[5] Nous n'expliquons pas comment construire les transitions de retrait d'événements. Ceci sera abordé lors de la présentation de l'algorithme 7.5 en page 167.

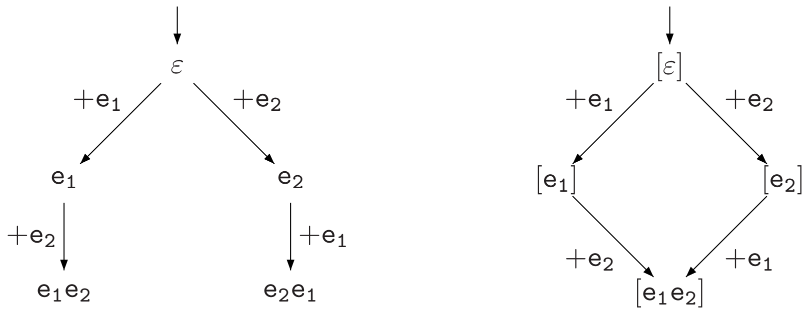
(a) si, parmi tous les états déjà créés, il existe un état $[e_{i_1} \dots e_{i_{n+1}}]$ tel que $[e_1 \dots e_{n+1}] = [e_{i_1} \dots e_{i_{n+1}}]$, alors nous plaçons une transition d'ajout de e_{n+1} de $[e_1 \dots e_n]$ jusqu'à $[e_{i_1} \dots e_{i_{n+1}}]$,

(b) sinon, un nouvel état $[e_1 \dots e_{n+1}]$ est créé et la transition d'ajout de e_{n+1} de $[e_1 \dots e_n]$ à ce nouvel état est ajoutée,

2. le point (1) est appliqué à tous les nouveaux états de taille inférieure à la borne de la file (obtenue, par exemple, au moyen de notre test décrit au chapitre 5 (page 107)).

Notre algorithme s'apparente à un algorithme d'exploration de l'espace d'états d'un système de transitions à un détail près : il construit le système au fur et à mesure qu'il le parcourt. Nous souhaitons utiliser les méthodes d'ordre partiel pour détecter de façon efficace l'existence d'états équivalents dans le point (1a), c'est-à-dire sans qu'il soit nécessaire d'explorer la table des états déjà créés pour trouver un état équivalent. En particulier, nous souhaitons savoir rapidement s'il existe un état équivalent qui a déjà été créé.

Inadéquation de la méthode des ensembles persistants. Il apparaît alors clairement que la technique des ensembles persistants ne nous permet pas d'atteindre notre but puisqu'elle ne préserve ni les comportements, ni les états. Considérons le système de transitions de la figure 7.8(a) qui modélise une file pouvant mémoriser une occurrence de chacun des événements e_1 et e_2 qui sont indépendants. Nous souhaitons produire le modèle réduit qui correspond à la



(a) Modèle complet. (b) Modèle réduit.

FIG. 7.8 – Modèles complet et réduit pour une file de 2 événements indépendants.

figure 7.8(b), mais la technique des ensembles persistants produit le système de transitions de la figure 7.9. En effet, puisque $\{+e_1\}$ est persistant[6] en ε, la transition d'ajout de e_2 est simplement ignorée. Le modèle obtenu n'est donc pas valide puisqu'il n'est plus possible de mémoriser e_2 lorsque la file est vide.

Succès de la méthode des ensembles endormis. La méthode des ensembles endormis nous permet de procéder à notre réduction. En effet, par construction, un ensemble endormi contient les événements qui conduisent dans des états déjà visités, c'est-à-dire, dans notre cas, des états équivalents au nouveau contenu de file. Reprenons pour exemple le modèle représenté

[6]Rappelons que la relation de dépendance n'est pas obtenue par analyse du système de la figure 7.8(a), mais par examen du programme ELECTRE correspondant.

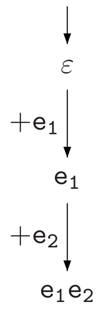


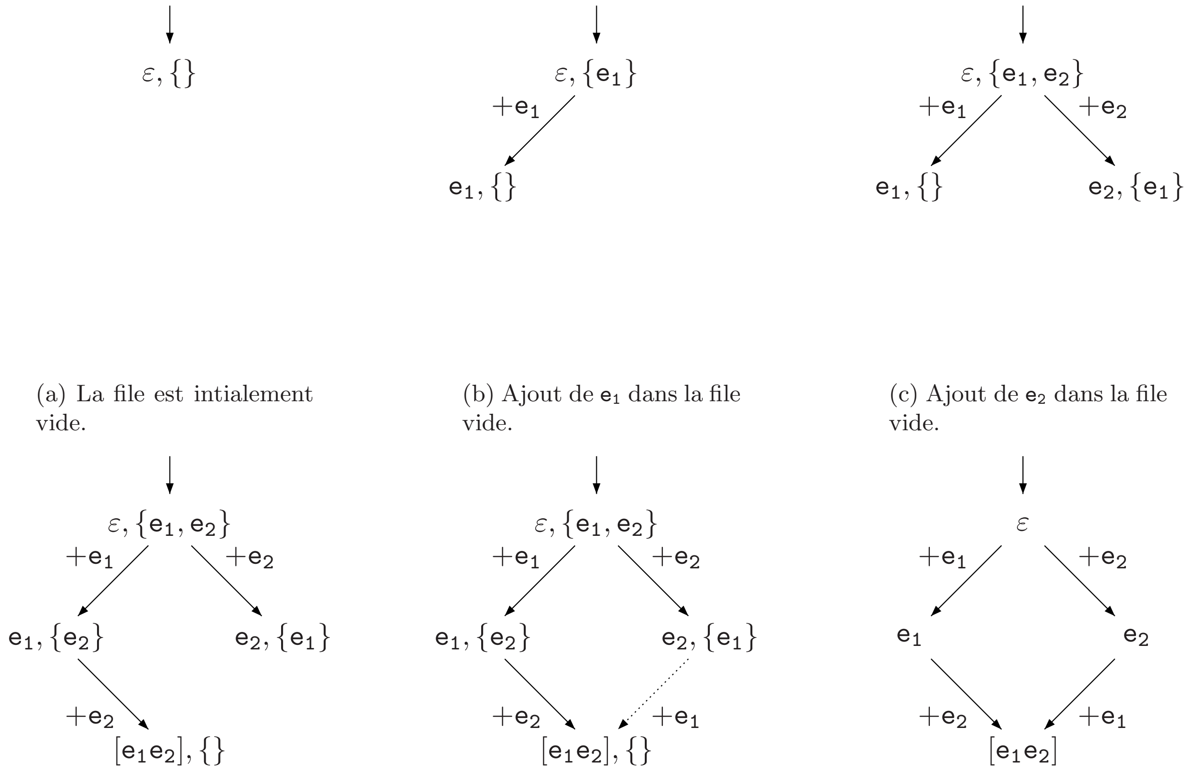
FIG. 7.9 – Modèle réduit pour une file de 2 événements indépendants par la technique des ensembles persistants.

en figure 7.8(a). La figure 7.10 représente la construction du modèle réduit (qui est représenté en figure 7.8(b)) en appliquant la méthode expliquée en début de cette section, adaptée pour calculer les ensembles endormis, comme expliqué page 161. Sur cette figure, les ensembles endormis sont indiqués entre accolades pour chaque état. Nous remarquons en figure 7.10(e) que l'ensemble endormi associé à l'état $\mathtt{e}_2$ contient $\mathtt{e}_1$. Par conséquent, nous savons que l'ajout d'une occurrence de $\mathtt{e}_1$ à la file conduit à un état équivalent à l'un des états déjà construits. Il n'est alors pas nécessaire de construire de nouvel état, mais il faut seulement ajouter la transition vers l'état existant. Notons que cette information essentielle est obtenue à très faible coût lors de la construction du modèle réduit ce qui fait la grande force de l'utilisation des techniques d'ordre partiel. Il reste ensuite à trouver l'état équivalent de façon efficace, par exemple, au moyen d'une fonction de hachage adaptée.

Algorithme de construction du modèle réduit

Nous supposons que nous voulons construire le modèle réduit d'une file de taille maximale N (connue, par exemple, suite à l'utilisation de notre test de non-bornitude présenté dans le chapitre 5, page 107). Cette construction est basée avant tout sur un algorithme classique de parcours de l'espace d'états, avec dans notre cas, la particularité de construire cet espace d'états lors de l'exploration. Notre méthode est décrite par l'algorithme 7.4. Q y représente une file FIFO utilisée pour le parcours en largeur d'abord de l'espace d'états, T est une table de hashage utilisée pour stocker les états déjà créés[7]. Les états sont identifiés par le contenu de file qu'ils représentent, en particulier, l'état initial est le mot vide ε. L'appel de la fonction $\mathsf{successeurs}(q)$ (décrite dans l'algorithme 7.5, page 170) calcule les états successeurs de q, trouve les états déjà créés et équivalents, construits les transitions d'ajout et de retrait des événements de la file et renvoit finalement la liste des états nouvellement créés. Il est par conséquent absolument nécessaire que les nouveaux états soient ajoutés à T dès leur création, et non pas lors de leur exploration pour que toutes les équivalences soient détectées. La construction des transitions de retrait des occurrences mémorisées est relativement complexe puisqu'il faut tenir compte à la fois du pliage des états, et de la préservation des comportements du modèle complet. Bien entendu, une seule occurrence de chaque événement peut être consommée en chaque état. Ensuite, lorsque sa position est déterminée, il faut trouver les indices minimaux et maximaux où peut se trouver

[7] Les instructions *push*, *pop* désignent les opérations classiques sur les piles, et *add* est l'opération d'ajout à une table de hashage.



(a) La file est intialement vide.

(b) Ajout de $\mathtt{e}_1$ dans la file vide.

(c) Ajout de $\mathtt{e}_2$ dans la file vide.

(d) Ajout de $\mathtt{e}_2$ lorsque la file contient $\mathtt{e}_1$.

(e) Ajout de $\mathtt{e}_1$ lorsque la file contient $\mathtt{e}_2$: traitement de l'ensemble endormi.

(f) Modèle réduit obtenu.

FIG. 7.10 – Construction du modèle réduit pour une file de 2 événements indépendants par la technique des ensembles endormis.

cette occurrence dans la file (par permutation, conformément à la définition 7.3). Il ne reste alors plus qu'à construire les transitions de retrait correspondantes.

Notons que dans le cas de la construction de ce système de transitions réduit, un état n'est jamais visité plus d'une fois, et que chaque état admet un successeur pour chaque événement du système. Il n'est donc pas nécessaire de s'inquiéter du rôle joué par l'ensemble endormi lors des visites successives d'un même état, comme c'est le cas pour l'algorithme 7.3.

Définition 7.10 (Modèle de file réduit : F_r)
Nous appelons *modèle de file réduit*, noté F_r, le système de transitions fini produit par l'algorithme 7.4. ♦

Propriétés du modèle de file réduit pour la vérification

Notre méthode de réduction ne préserve pas l'ordre entre les occurrences d'événements mémorisées indépendantes. Il est donc nécessaire de bien en analyser les conséquences sur la vérification des modèles réduits obtenus. Plus particulièrement, il faut déterminer quelles sont les propriétés qui sont préservées par la réduction.

$Q \leftarrow \emptyset$; $T \leftarrow \emptyset$; $endormi(\varepsilon) \leftarrow \emptyset$
$Q.push\ \varepsilon$; $T.add\ \varepsilon$
tant que $Q \neq \emptyset$ **faire**
  $q \leftarrow Q.pop$
  $L \leftarrow \mathsf{successeurs}(q)$
  pour chaque $q' \in L$ **faire**
    $Q.push\ q'$; $T.add\ q'$
  fin pour
fin tant que

Algorithme 7.4: Algorithme de construction du modèle réduit de la file.

Théorème 7.11

$$[\![R \,||\, F]\!] = [\![R \,||\, F_r]\!]$$

♦

Preuve. Il suffit de constater que comme toutes les exécutions du système sont permises (il n'y a pas d'environnement), la perte de l'ordre entre les occurrences mémorisées laisse le nombre de comportement invariant : seul du non-déterminisme est ajouté. ♦

En plus de préserver les exécutions, notre méthode de réduction préserve l'accessibilité des configurations.

Théorème 7.12
Une configuration (q, w) est accessible dans $R \,||\, F$ si et seulement si $(q, [w])$ est accessible dans $R \,||\, F_r$. ♦

Preuve. L'argument pour cette preuve est le même pour le théorème précédent. ♦

Nous obtenons par conséquent le résultat de vérification suivant :

Corollaire 7.13
Soit ϕ une formule de logique temporelle (Ltl, Ctl, etc.) dont les propriétés atomiques ne font pas intervenir l'ordre entre les occurrences mémorisées. $R \,||\, F \models \phi$ si et seulement si $R \,||\, F_r \models \phi$. ♦

Cependant, lorsque le système est embarqué dans un environnement E (définition 2.21, page 51), le résultat est moins fort car certaines exécutions qui avaient été empêchées par l'environnement sont réintroduites lors de la réduction puisque l'ordre entre les occurrences mémorisées est perdu. Notons $\pi_{\overline{?}}$ le morphisme de $(A_R, .)$ dans $(A_R \setminus (\{?\} \times \Sigma_M), .)$ défini par :

$$\pi_{\overline{?}}(a) = \begin{cases} \varepsilon & \text{si } a = ?\mathsf{e},\ \mathsf{e} \in \Sigma_M \\ a & \text{sinon} \end{cases}$$

```
/* Calculs des successeurs d'un état q */
/* si q est de la taille maximale fixée, il n'a pas de nouveaux successeurs par mémorisation */
si |q| < N alors
   /* Traitement de l'ensemble endormi */
   pour chaque e ∈ endormi(q) faire
      q' ← T.equivalent(qe)
      ajouter la transition q --+e--> q'
   fin pour

   /* Calcul des nouveaux successeurs */
   pour chaque e ∈ [(E_{M*} ∪ (E_{M1} \ Σ_q)) \ endormi(q)] faire
      créer q.e
      endormi(q.e) = {e' ∈ endormi(q) | (e, e') ∉ D}
      endormi(q) = endormi(q) ∪ {e}
      ajouter la transition q --+e--> q.e
   fin pour

   /* Traitement des événements à mémorisation unique qui sont déjà dans q */
   pour chaque e ∈ (Σ_q ∩ E_{M1}) faire
      ajouter la transition q --+e--> q
   fin pour
sinon
   /* |q| = N : les mémorisations sont «sans effet» */
   pour chaque e ∈ Σ_q faire
      ajouter la transition q --+e--> q
   fin pour
fin si

/* Boucle pour la synchronisation avec les transitions de l'automate à file réactif sans effet
sur la file : τ est un événement spécial ajouté au modèle de la file */
ajouter la transition q --τ--> q

/* Transitions de consommation */
pour chaque e ∈ Σ_q faire
   i_e ← min{i ∈ {1, ..., |q|} | q_i = e}
   i_- ← min ({i ∈ {1, ..., i_e − 1} | ∀j ∈ {i, ..., i_e − 1}, (q_j, e) ∉ D} ∪ {i_e})
   i_+ ← max ({i ∈ {i_e + 1, ..., |q|} | ∀j ∈ {i_e + 1, ..., i}, (q_j, e) ∉ D} ∪ {i_e})
   pour j ← i_- à i_+ faire
      q' ← inverser q_j et q_{i_e} dans q
      q'' ← T.equivalent(q'_1 ... q'_{(j−1)} q'_{(j+1)} ... q_{|q|})
      ajouter la transition q --(−e)_j--> q''
   fin pour
fin pour

Renvoyer la liste des nouveaux états créés.
```

Algorithme 7.5: Algorithme de calcul des successeurs de q.

Ce morphisme abstrait les exécutions de $R \,||\, F \,||\, E$ (ou $R \,||\, F_r \,||\, E$) en rejettant les séquences de stabilisation. Il est aisément étendu à un ensemble de séquences.

Théorème 7.14

$$\pi_{\overline{?}}(\llbracket\, R \,||\, F \,||\, E \,\rrbracket) = \pi_{\overline{?}}(\llbracket\, R \,||\, F_r \,||\, E \,\rrbracket)$$

♦

Preuve.

– Nous montrons l'inclusion $\pi_{\overline{?}}(\llbracket\, R \,||\, F \,||\, E \,\rrbracket) \subseteq \pi_{\overline{?}}(\llbracket\, R \,||\, F_r \,||\, E \,\rrbracket)$. Soit $\sigma \in \llbracket\, R \,||\, F \,||\, E \,\rrbracket$, et σ_i son préfixe aboutissant dans sa $i^{ième}$ configuration stable : $\langle q_R^i, w_i, q_E^i \rangle$. Nous montrons la propriété **(P)** suivante par induction sur i :

il existe $\sigma' \in \llbracket\, R \,||\, F_r \,||\, E \,\rrbracket$ telle que :

1. σ'_i aboutit en $\langle q_R^i, [w_i], q_E^i \rangle$ (qui est stable),
2. et $\pi_{\overline{?}}(\sigma'_i) = \pi_{\overline{?}}(\sigma_i)$.

– **Si i = 0.** Les deux configurations initiales $\langle q_R^0, \varepsilon, q_E^0 \rangle$ pour $R \,||\, F \,||\, E$, et $\langle q_R^0, [\varepsilon], q_E^0 \rangle$ pour $R \,||\, F_r \,||\, E$ satisfont le point (1) de **(P)**. Sachant que $i = 0$, le point (2) est trivialement vérifié.

– **Si i > 0.** Supposons que **(P)** soit vérifiée jusqu'au rang i. Alors, deux cas se présentent suivant la transition franchie :

  – **Après la transition $\langle \mathbf{q_R^i}, \mathbf{w_i}, \mathbf{q_E^i} \rangle \xrightarrow{a} \langle \mathbf{q_R^{i+1}}, \mathbf{w_{i+1}}, \mathbf{q_E^{i+1}} \rangle$, $\mathbf{R \,||\, F \,||\, E}$ atteint la configuration stable $\langle \mathbf{q_R^{i+1}}, \mathbf{w_{i+1}}, \mathbf{q_E^{i+1}} \rangle$.** Par hypothèse d'induction, $\langle q_R^i, [w_i], q_E^i \rangle$ est stable, et $R \,||\, F_r \,||\, E$ franchit donc la même transition. Soit cette transition laisse la file invariante : $w_{i+1} = w_i$. Sinon, il y a mémorisation d'un événement $\mathtt{e}$. Rappelons que par la définition 7.3 (page 158), $[w_i].\mathtt{e} \in [w_i.\mathtt{e}]$ et $w_{i+1} = w_i.\mathtt{e}$. Donc dans les deux cas, $R \,||\, F_r \,||\, E$ atteint la configuration : $\langle q_R^{i+1}, [w_{i+1}], q_E^{i+1} \rangle$.
  Finalement, $\pi_{\overline{?}}(\sigma_{i+1}) = \pi_{\overline{?}}(\sigma_i.a) = \pi_{\overline{?}}(\sigma_i).\pi_{\overline{?}}(a)$, et par hypothèses d'induction : $\pi_{\overline{?}}(\sigma_i) = \pi_{\overline{?}}(\sigma'_i)$. Donc, $\pi_{\overline{?}}(\sigma_{i+1}) = \pi_{\overline{?}}(\sigma'_{i+1})$.

  – **Après la transition $\langle \mathbf{q_R^i}, \mathbf{w_i}, \mathbf{q_E^i} \rangle \xrightarrow{a} \langle \mathbf{q_R'^i}, \mathbf{w_i}, \mathbf{q_E^{i+1}} \rangle$, $\mathbf{R \,||\, F \,||\, E}$ atteint la configuration instable $\langle \mathbf{q_R'^i}, \mathbf{w_i}, \mathbf{q_E^{i+1}} \rangle$, puis il franchit une séquence de stabilisation $\sigma_?$.** (Notons que par le lemme 2.14, page 46, a est nécessairement une transition de traitement immédiat, qui laisse donc la file w invariante).

$$\langle q_R^i, w_i, q_E^i \rangle \xrightarrow{a} \langle q_R'^i, w_i, q_E^{i+1} \rangle \xrightarrow{\sigma_?} \langle q_R^{i+1}, w_{i+1}, q_E^{i+1} \rangle$$

  De même, $R \,||\, F_r \,||\, E$ franchit une première transition d'action a (la configuration $\langle q_R^i, [w_i], q_E^i \rangle$ est stable par hypothèse d'induction) et atteint la configuration instable $\langle q_R'^i, [w_i], q_E^{i+1} \rangle$. Ensuite, il franchit une séquence de stabilisation $\sigma'_?$.
  Par définition des traces (définition 7.3, page 158), $\sigma_?$ et $\sigma'_?$ consomment les mêmes événements et en même nombre, seul l'ordre diffère. De plus, toujours par la définition

des traces, les différences d'ordre entre $\sigma_?$ et $\sigma'_?$ correspondent à des permutations d'événements adjacents et indépendants : $\sigma_? \in [\sigma'_?]$. Par le lemme 7.4 (page 158) et le théorème 7.9 (page 165) $R \,||\, F_r \,||\, E$ atteint donc finalement la configuration stable : $\langle q_R^{i+1}, [w_{i+1}], q_E^{i+1} \rangle$. Enfin, puisque $\pi_{\overline{?}}(\sigma_?) = \varepsilon = \pi_{\overline{?}}(\sigma'_?)$, nous en déduisons la satisfaction du point (2) de la propriété **(P)**.

– L'inclusion inverse : $\pi_{\overline{?}}(\llbracket\, R \,||\, F \,||\, E \,\rrbracket) \supseteq \pi_{\overline{?}}(\llbracket\, R \,||\, F_r \,||\, E \,\rrbracket)$ est prouvée de la même façon.

♦

Par le théorème précédent, nous avons la conservation des exécutions du système, aux séquences de stabilisation près. Cette dernière restriction est liée au fait que l'accessibilité des configurations instables n'est pas préservée. Cependant, l'accessibilité des configurations stables est, elle, préservée.

Théorème 7.15
Une configuration stable $\langle q_R, w, q_E \rangle$ est accessible dans $R \,||\, F \,||\, E$ si et seulement si $\langle q_R, [w], q_E \rangle$ est accessible dans $R \,||\, F_r \,||\, E$. ♦

Preuve. Nous montrons que si une configuration stable $\langle q_R, w, q_E \rangle$ est accessible dans $R \,||\, F \,||\, E$, alors la configuration stable $\langle q_R, [w], q_E \rangle$ est accessible dans $R \,||\, F_r \,||\, E$. La réciproque se montre en appliquant la même démarche.

Notons que les configurations initiales $\langle q_R^0, \varepsilon, q_E^0 \rangle$ et $\langle q_R^0, [\varepsilon], q_E^0 \rangle$, de $R \,||\, F \,||\, E$ et $R \,||\, F_r \,||\, E$ respectivement, sont accessibles.

Considérons que $\langle q_R, w, q_E \rangle$ et $\langle q_R, [w], q_E \rangle$ sont stables et accessibles dans $R \,||\, F \,||\, E$ et $R \,||\, F_r \,||\, E$ respectivement.

– **Soit $R \,||\, F \,||\, E$ atteint la configuration stable $\langle q'_R, w', q'_E \rangle$ en une seule transition $\langle q_R, w, q_E \rangle \xrightarrow{a} \langle q'_R, w', q'_E \rangle$.** La transition a correspond donc soit à un traitement immédiat, soit à une mémorisation. Par hypothèse d'induction, $\langle q_R, [w], q_E \rangle$ est stable, et l'environnement E impose la même contrainte. Si a est une transition de traitement immédiat, $w' = w$ et $R \,||\, F_r \,||\, E$ atteint bien la configuration $\langle q'_R, [w'], q'_E \rangle$. À l'inverse, si a est une transition de mémorisation de e, $w' = w.\mathsf{e}$, et par la définition 7.3 (page 158), $[w].\mathsf{e} = [w.\mathsf{e}] = [w']$, donc $R \,||\, F_r \,||\, E$ atteint la configuration stable $\langle q'_R, [w'], q'_E \rangle$.
– **Soit $R \,||\, F \,||\, E$ atteint la configuration instable $\langle q'_R, w, q'_E \rangle$ puis franchit la séquence de stabilisation $\sigma_?$ pour atteindre la configuration $\langle q''_R, w', q'_E \rangle$.**

$$\langle q_R, w, q_E \rangle \xrightarrow{a} \langle q'_R, w, q'_E \rangle \xrightarrow{\sigma_?}{}^{*} \langle q''_R, w', q'_E \rangle$$

(Par le lemme 2.14, page 46, a est nécessairement une transition de traitement immédiat, qui laisse donc la file w invariante).

Par hypothèse d'induction, $R \,||\, F_r \,||\, E$ franchit la même transition a que $R \,||\, F \,||\, E$, et atteint ainsi la configuration instable $\langle q'_R, [w], q'_E \rangle$. Puis, il franchit la séquence de stabilisation $\sigma'_?$ qui, par la définition des traces (définition 7.3, page 158), diffère de $\sigma_?$ uniquement par l'ordre du traitement des occurrences mémorisées. En utilisant le lemme 7.4 (page 158) et le théorème 7.9 (page 165), nous obtenons :

$$\langle q_R, [w], q_E\rangle \xrightarrow{a} \langle q'_R, [w], q'_E\rangle \xrightarrow{\sigma'_?}{}^* \langle q''_R, [w'], q'_E\rangle$$

ce qui conclut sur l'accessibilité de la configuration stable $\langle q'_R, [w'], q'_E\rangle$.

♦

Notons que seules les configurations stables décrivent des états réels du système modélisé. Les deux résultats précédents permettent donc de vérifier l'essentiel des propriétés du système sur sa représentation réduite.

Corollaire 7.16
Soit ϕ une formule de logique temporelle (LTL, CTL, etc.) dont les propriétés atomiques sont exprimées sur les configurations stables de $R\,||\,F\,||\,E$ et qui ne font pas intervenir l'ordre entre les occurrences mémorisées. $R\,||\,F\,||\,E \vDash \phi$ si et seulement si $R\,||\,F_r\,||\,E \vDash \phi$. ♦

Pour des propriétés plus générales, exprimées sur l'ensemble des configurations (stables et instables) par exemple, notre réduction préserve la satisfaction des propriétés de sûreté : toute propriété de sûreté vérifiée sur $R\,||\,F_r\,||\,E$ l'est aussi sur $R\,||\,F\,||\,E$.

Éléments de complexité pour le modèle réduit

Le gain apporté par notre méthode de réduction se traduit par une plus faible utilisation de l'espace mémoire (parfois de façon spectaculaire). Cependant, l'utilisation de cette technique ne permet pas toujours d'obtenir un modèle assez petit, en particulier lorsque la relation de dépendance est forte. Il est donc intéressant de connaître la taille du modèle réduit avant de le calculer, d'autant plus que ce calcul peut être couteux : le passage de n à $n+1$ événements mémorisables se traduit par une augmentation factorielle du nombre de contenus possibles pour la file (pour n événements à mémorisation unique, il y a $n!$ contenus de taille n dans la modèle complet).

Approche théorique. Nous notons $\eta_1 = Card(\mathcal{E}_{M_1})$ le nombre d'événements à mémorisation unique et $\eta_* = Card(\mathcal{E}_{M_*})$ le nombre d'événements à mémorisation multiple. Nous appelons *états de rang n* les états du modèle de la file qui sont formés de n occurrences d'événements, et nous notons $rang(n)$ le nombre d'états de rang n dans ce modèle. Nous cherchons par la suite à caractériser le nombre d'états de rang n, la taille globale du modèle étant alors donnée par la formule :

$$nb(N) = \sum_{i=0}^{N} rang(i) \tag{7.1}$$

où N est la *borne* de la file : le nombre maximal d'occurrences qu'elle peut contenir simultanément.

$\boxed{\text{Cas au pire.}}$ Nous nous intéressons en premier lieu au pire des cas : lorsque tous les événements sont dépendants (c'est à dire $\mathcal{D} = \mathcal{E} \times \mathcal{E}$). Voici de façon informelle comment nous pouvons décrire le nombre d'états :

- $n = 0$: le seul état ne contenant pas d'événement est ε,

$$rang(0) = 1$$

- $n = 1$: ces états sont obtenus depuis le précédent en ajoutant n'importe quel événement mémorisable :

$$rang(1) = \eta_1 + \eta_*$$

- $n = 2$: ces états sont composés :
  - soit exclusivement d'événements à mémorisation unique, alors le premier est l'un des η_1 événements de ce type, alors qu'il ne reste que $(\eta_1 - 1)$ choix possibles pour le second, donc finalement $\eta_1 \times (\eta_1 - 1)$ possibilités,
  - soit d'un événement à mémorisation unique, puis d'un événement à mémorisation multiple, ce qui laisse $\eta_1 \times \eta_*$ possibilités,
  - soit l'inverse : un événement à mémorisation multiple puis un événement à mémorisation unique : $\eta_* \times \eta_1$ états,
  - soit finalement exclusivement d'événements à mémorisation multiple, ce qui donne $\eta_* \times \eta_*$ possibilités, et finalement :

$$rang(2) = [\eta_1 \times (\eta_1 - 1)] + 2 \times [\eta_1 \times \eta_*] + [\eta_* \times \eta_*]$$

- $n = 3$: de la même façon que pour le cas précédent, ces états contiennent :
  - soit uniquement des événements à mémorisation unique. Par définition, il y a donc η_1 choix possibles pour le premier événement, puis $(\eta_1 - 1)$ pour le deuxième et enfin $(\eta_1 - 2)$ pour le troisième, ce qui donne $\eta_1 \times (\eta_1 - 1) \times (\eta_1 - 2)$ permutations,
  - soit un événement à mémorisation unique, puis un autre, et finalement un événement à mémorisation multiple, donc $\eta_1 \times (\eta_1 - 1) \times \eta_*$ possibilités,
  - soit un événement à mémorisation unique, puis un événement à mémorisation multiple, et finalement un événement à mémorisation unique, ce qui donne à nouveau $\eta_1 \times \eta_* \times (\eta_1 - 1)$ états possibles,
  - soit d'abord un événement à mémorisation multiple puis deux événements à mémorisation unique et par conséquent $\eta_* \times \eta_1 \times (\eta_1 - 1)$ possibilités,
  - ensuite, comme dans les trois cas précédents, nous ajoutons les permutations de deux événements à mémorisation multiple et d'un événement à mémorisation unique, ce qui donne dans les trois cas, $\eta_* \times \eta_* \times \eta_1$ permutations,
  - enfin, il reste les états qui ne contiennent aucun événement à mémorisation unique, il y en a : $\eta_* \times \eta_* \times \eta_*$, ce qui conduit à :

$$rang(3) = [\eta_1 \times (\eta_1 - 1) \times (\eta_1 - 2)] + 3 \times [\eta_1 \times (\eta_1 - 1) \times \eta_*] + 3 \times [\eta_1 \times \eta_* \times \eta_*] + [\eta_* \times \eta_* \times \eta_*]$$

et ainsi de suite...

Lemme 7.17

Dans le pire des cas (tous les événements dépendants) :

$$rang(n) = (\eta_*)^n + n \times \sum_{i=0}^{n-2} \left[\left(\prod_{j=0}^{i} (\eta_1 - j) \right) \times (\eta_*)^{(n-(i+1))} \right] + \prod_{i=0}^{n-1} (\eta_1 - i)$$

♦

La preuve de ce lemme est donnée en annexe D, section D.2, page 222. Ainsi, dans le cas où $\eta_* = 0$, c'est-à-dire que tous les événements sont à mémorisation unique, $rang(n) = \mathsf{A}_n^{\eta_1} = \frac{\eta_1!}{(\eta_1 - n)!}$ est un arrangement de n événements parmi η_1.

Cas au mieux. Dans le meilleur des cas, la relation de dépendance correspond à l'identité et l'ordre entre les événements n'a plus aucune importance. Par convenance, nous considérons donc que les états sont composés d'un certain nombre d'événements à mémorisation unique rangés en ordre lexicographique puis d'événements à mémorisation multiple, eux aussi placés en ordre lexicographique. Comme pour le cas précédent, nous commençons par une description informelle de $rang(n)$.

Considérons tout d'abord les états de taille n qui sont exclusivement composés d'événements à mémorisation multiple. Notons $p_1, \ldots, p_{\eta_*}$ les nombres d'occurrences de chacun des événements dans un état donné. À tout moment, nous avons donc : $p_1 + \cdots + p_{\eta_*} = n$. Dans chacun de ces états, il y a (p_1) entre 0 et n occurrences du premier événement mémorisable, puis (p_2) entre 0 et $n - p_1$ occurrences du deuxième événement, et ainsi de suite, jusqu'à l'avant-dernier événement qui figure (p_{η_*-1}) entre 0 et $n - \sum_{i=1}^{n-2} p_i$ fois, le nombre d'occurrences du $n^{ième}$ événement mémorisable étant fixé par la relation $p_1 + \cdots + p_{\eta_*} = n$, donc par $p_1, \ldots, p_{\eta_*-1}$. Ceci conduit donc à $\sum_{p_1=0}^{n} \sum_{p_2=0}^{(n-p_1)} \cdots \sum_{p_{n-1}}^{\left(n - \sum_{i=1}^{n-2} p_i\right)} 1$ possibilités.

Maintenant, il est aisé de tenir compte des événements à mémorisation unique. En effet, l'ensemble des états de rang n disposant d'un événement à mémorisation unique (parmi η_1) puis de $n-1$ événements à mémorisation multiple est obtenu en ajoutant l'un des événements à mémorisation unique (seulement en tête puisque l'ordre n'intervient pas) aux états de rang $n-1$ ne contenant que des événements à mémorisation multiple. Nous procédons de la même façon pour deux événements à mémorisation unique, puis trois, etc.

Lemme 7.18

Dans le meilleur des cas (tous les événements sont indépendants) :

$$rang(n) = \sum_{i=0}^{n} \left[\left(\prod_{j=0}^{i-1} (\eta_1 - j) \right) \times \left(\sum_{p_1=0}^{(n-i)} \sum_{p_2=0}^{(n-i-p_1)} \cdots \sum_{p_{(\eta_*-1)}=0}^{\left(n-i-\sum_{k=1}^{(\eta_*-2)} p_k\right)} 1 \right) \right]$$

♦

La preuve de ce lemme est donnée en annexe D, section D.3, page 223. Notons que lorsque tous les événements sont à mémorisation unique (c'est à dire $\eta_* = 0$), $rang(n) = \mathsf{C}_n^{\eta_1} = \frac{\eta_1}{n! \times (\eta_1 - n)!}$, ce qui correspond bien à l'intuition : dans le meilleur des cas, le nombre d'états de rang n est donné par le nombre de listes non ordonnées de longueur n.

Cas en moyenne. Le cas en moyenne est beaucoup plus complexe que les précédents. En effet, alors que dans les premiers cas la relation de dépendance est uniforme sur l'ensemble des événements : ils sont tous dépendants entre eux, ou au contraire tous indépendants, il est maintenant nécessaire de considérer le cas de chaque événement séparément.

Nous n'avons pas pu déterminer la complexité en moyenne dans le cas général à cause de la difficulté d'exprimer la dépendance entre les événements et plus particulièrement, parce que cette relation n'est généralement pas transitive. En effet, considérons trois événements $\mathtt{e}_1$, $\mathtt{e}_2$ et $\mathtt{e}_3$ et la relation de dépendance[8] $\mathcal{D} = \{(\mathtt{e}_1, \mathtt{e}_2), (\mathtt{e}_1, \mathtt{e}_3)\}$. Les traces formées par les séquences contenant une occurrence de chacun de ces événements ne représentent pas le même nombre de séquences totalement ordonnées contrairement à ce qui se produit pour les cas au pire et au mieux. Par exemple, $[\mathtt{e}_1\mathtt{e}_2\mathtt{e}_3] = \{\mathtt{e}_1\mathtt{e}_2\mathtt{e}_3, \mathtt{e}_1\mathtt{e}_3\mathtt{e}_2\}$ alors que $[\mathtt{e}_2\mathtt{e}_1\mathtt{e}_3] = \{\mathtt{e}_2\mathtt{e}_1\mathtt{e}_3\}$. Il faut donc faire intervenir l'ordre entre les occurrences d'événements, ce qui complique grandement le problème.

Nous proposons donc, dans la section suivante, d'étudier, de façon informelle, l'effet du nombre d'événements mémorisables et le ratio de dépendance entre ceux-ci, sur la taille du modèle réduit obtenu.

Aperçu de la complexité sur quelques exemples. Pour terminer cette section sur la complexité, nous montrons de façon plus informelle l'influence du nombre d'événements et du taux de dépendance sur la taille du modèle réduit. Ces deux paramètres déterminent en effet le nombre d'états du système de transitions, et donc, l'efficacité de notre technique de réduction. Dans la suite, les programmes Electre utilisés sont donnés à titre d'exemple et il existent de nombreux autres programmes qui remplissent les mêmes critères. Le tableau 7.1 présente l'effet d'une augmentation du nombre d'événements (tous indépendants) sur le nombre d'états des modèles complets et réduits. Dans le modèle complet, ce facteur fait croître le nombre d'états de façon exponentielle, alors que dans le modèle réduit, cela se traduit par une croissance géométrique. Notons que la réduction a d'autant plus d'effet que le nombre d'événements (tous indépendants) est grand. Ceci s'explique simplement par le fait que la trace $[\mathtt{e}_1\mathtt{e}_2\mathtt{e}_3\mathtt{e}_4\mathtt{e}_5]$ remplace alors $5! = 120$ états dans le modèle réduit correspondant au programme à 5 événements.

Nombre d'événements	Programme	Nb états (complet)	Nb états (réduit)	% réduction
1	`await` $\mathtt{e}_1$	2	2	0%
2	`await` $\{\mathtt{e}_1 \parallel \mathtt{e}_2\}$	5	4	20%
3	`await` $\{\mathtt{e}_1 \parallel \mathtt{e}_2 \parallel \mathtt{e}_3\}$	16	8	50%
4	`await` $\{\mathtt{e}_1 \parallel \mathtt{e}_2 \parallel \mathtt{e}_3 \parallel \mathtt{e}_4\}$	65	16	75%
5	`await` $\{\mathtt{e}_1 \parallel \mathtt{e}_2 \parallel \mathtt{e}_3 \parallel \mathtt{e}_4 \parallel \mathtt{e}_5\}$	326	32	90%

Tab. 7.1 – Pourcentage d'états supprimés en fonction du nombre d'événements indépendants.

[8] Nous avons réduit $\mathcal{D}$ à son expression minimale, il faut bien sûr y ajouter les couples obtenus par symétrie et par réflexion.

Le tableau 7.2 montre l'influence du taux de dépendance sur l'efficacité de la méthode de réduction. Le taux d'indépendance est simplement obtenu par $\frac{Card(\mathcal{D})}{Card(\mathcal{E}_M \times \mathcal{E}_M)}$ (en faisant abstraction des éléments réflexifs). Nous faisons varier le taux de dépendance en remplaçant progressivement les opérateurs de composition parallèle qui organisent les attentes d'événements, par des opérateurs de composition exclusive. L'importance du pourcentage de dépendance sur l'efficacité de la réduction apparaît clairement sur ce tableau, et n'a rien de surprenant.

% événements dépendants	Programme	Nb états (complet)	Nb états (réduit)	% réduction
0%	`await` $\{e_1 \parallel e_2 \parallel e_3 \parallel e_4 \parallel e_5\}$	326	32	90%
40%	`await` $\{e_1 \mid \{e_2 \parallel e_3 \parallel e_4 \parallel e_5\}\}$	326	97	70%
70%	`await` $\{e_1 \mid e_2 \mid \{e_3 \parallel e_4 \parallel e_5\}\}$	326	190	42%
90%	`await` $\{e_1 \mid e_2 \mid e_3 \mid \{e_4 \parallel e_5\}\}$	326	277	15%
100%	`await` $\{e_1 \mid e_2 \mid e_3 \mid e_4 \mid e_5\}$	326	326	0%

TAB. 7.2 – Pourcentage d'états supprimés en fonction du taux de dépendance.

Finalement, alors que le nombre d'états augmente de façon géométrique avec le nombre d'événements, le taux de dépendance influe plus fortement (de façon moins favorable) sur la taille du modèle réduit.

7.3.3 Perspectives pour notre méthode de réduction

Implantation et utilisation de notre méthode de réduction

Nous avons implanté notre technique de réduction, et l'outil obtenu a été utilisé dans le cadre de la vérification d'une application de contrôle d'un réacteur d'avion [16]. Cette application est composée de 6 programmes

- Pour quatre d'entre eux, tous les événements sont dépendants. Notre méthode de réduction n'a donc pas pu être utilisée.
- Pour les deux autres programmes, le taux de dépendance est, au contraire, très faible, et nous avons pu tester la capacité de notre méthode pour la vérification d'applications réelles. Le tableau 7.3 résume les caractéristiques principales de ces deux programmes.
  Les événements sont tous à mémorisation unique. Le faible taux de dépendance (le calcul de cette valeur est expliqué page 176), autour de 3% dans les deux cas, permet d'obtenir une très forte réduction.

Prog.	# opérateurs ELECTRE	# événements mémorisables	taux de dépendance	# états non-réduit	# états réduit	% réduction
1	60	10	3.33%	9.864.101	2.048	99%
2	49	15	2.86%	$\simeq 3,55 \times 10^{12}$	133.120	99%

TAB. 7.3 – Résultats de réduction pour la vérification d'une application industrielle.

Au-delà de notre méthode de réduction

Notre méthode s'inspire du fait que l'ordre de traitement de certains événements n'influe pas sur l'accessibilité des configurations stables du système. L'utilisation du modèle réduit construit par notre méthode permet par exemple de vérifier l'ensemble des propriétés de sûreté qui s'expriment sur les configurations stables du système (en effet, le système réduit possède au moins tous les comportements du système complet, comme nous l'indiquons dans le corollaire 7.13, page 169). Lorsque la propriété à vérifier dépend de l'ordre des occurrences mémorisées, par exemple «Toute occurrence mémorisée de $\mathtt{r}_2$ est précédée d'au moins une occurrence mémorisée de $\mathtt{r}_1$», il faut conserver l'ordre entre les occurrences de $\mathtt{r}_1$ et $\mathtt{r}_2$, c'est à dire considérer que la relation de dépendance est $\mathcal{D}' = \mathcal{D} \cup \{(\mathtt{r}_1, \mathtt{r}_2), (\mathtt{r}_2, \mathtt{r}_1)\}$ où $\mathcal{D}$ est la relation de dépendance calculée sur le programme Electre. Bien entendu, cela est réalisé au détriment de la taille du modèle réduit.

Finalement, nous avons rempli notre objectif : réduire le modèle de vérification de la file. Notons que dans le cas des système à file réactifs embarqués, cela se traduit par une augmentation du temps de vérification puisque si le nombre de configurations du modèle global est (fortement) diminué, le nombre de ses comportements ont eux (fortement) augmenté. Il faut donc envisager d'utiliser les méthodes d'ordre partiel pour s'attaquer au nombre de comportements du système, typiquement en introduisant le calcul des ensembles persistants lors de la vérification afin de n'explorer que les chemins nécessaires pour conclure sur la satisfaction d'une propriété donnée. Or, cela n'est possible que dans le cadre de la création d'un outil de vérification dédié à Electre ou plus généralement aux automates à file réactifs. Un autre argument plaide en ce sens : bien qu'il soit actuellement impossible de faire autrement, le fait de construire entièrement et statiquement le modèle de vérification de la file est abbérant puisqu'en général, seule une infime partie de celui-ci est accessible. Le fait que notre méthode n'ait pas pu être utilisée avec succès pour la vérification de l'application évoquée en section 7.3.3 plaide d'ailleurs dans ce sens. Notre méthode pourrait aussi bien être utilisée dans le cadre d'une méthode de vérification qui génère l'espace des états qu'elle explore dynamiquement.

Dans cette partie, nous avons introduit le langage $\mathcal{H}$-Electre pour la spécification temporelle des applications temps-réel. Le modéle obtenu alors, un SFR Hybride Linéaire, a la puissance d'une machine de Turing, et par conséquent, le problème de la bornitude est indécidable. Nous avons donc adapté notre test de non-bornitude au contexte temporisé. Cependant, même lorsque le SFR Hybride Linéaire est borné, nous n'avons pas la certitude que notre test termine.

Ensuite, nous avons montré une méthode pour la réduction du modèle de vérification des programmes Electre. En effet, même lorsqu'un SFR Embarqué, ou Hybride Linéaire, est borné, sa taille est bien souvent trop grosse pour qu'il puisse être vérifié. Notre méthode permet, quand elle s'applique, d'obtenir une réduction drastique : elle a d'ailleurs été utilisée avec succès pour la vérification d'une application de contrôle d'un réacteur d'avion.

Conclusion

Bilan. La conception et la vérification des systèmes réactifs avec mémorisation se heurtait, jusqu'à présent, au problème de la bornitude du dispositif de mémorisation.

- Du point de vue conceptuel, le problème est principalement lié à l'implantation du système considéré qui *doit* être borné : tout dispositif physique de mémorisation est de taille bornée.
- Du point de vue de la vérification, les méthodes utilisées pour les modèles finis (i.e. lorsque le modèle est borné) ou infinis (i.e. lorsque le modèle n'est pas borné) diffèrent. De plus, bien que la vérification des modèles infinis soit un domaine de recherche particulièrement actif actuellement, les méthodes existantes s'appliquent à des classes particulières de modèles.

Notre travail apporte une réponse au problème de la bornitude pour les *systèmes à file réactifs (SFR)*, un modèle pour les systèmes réactifs avec mémorisation, et pour les *programmes* ELECTRE, un langage pour la spécification des SFR. Les systèmes à file réactifs modélisent les réactions du système aux sollicitations de son environnement (représentées par des événements discrets), sans expliciter la façon dont ces dernières se produisent. Or, la bornitude du système modélisé, aussi bien que les autres propriétés de celui-ci, dépendent de la contrainte imposée par l'environnement. Nous avons comblé cette lacune en introduisant la spécification, en sus, de l'*environnement* des systèmes à file réactifs, sous la forme d'un système de transitions étiqueté, ce qui conduit aux *systèmes à file réactifs embarqués (SFR Embarqués)*.

Nous avons principalement montré que :

- La bornitude des SFR Embarqués est indécidable pour les systèmes possédant 2 événements mémorisables et un environnement périodique (la grande majorité des SFR Embarqués sont plus expressifs).
- L'accessibilité est décidable pour les *Lossy SFR Embarqués* (i.e. les SFR Embarqués dont la file de mémorisation n'est pas fiable) dont l'environnement est bien structuré.

Ces deux résultats apportent une réponse aux points de vue de la conception et de la vérification de SFR Embarqués : il est impossible de déterminer, en général, si un système est borné. Par contre, même s'il n'est pas borné, il est possible de déduire qu'il satisfait les propriétés de sûreté (i.e. le système ne se trouve jamais dans une «mauvaise situation»), qui sont satisfaites par le modèle avec perte correspondant.

Cependant, du point de vue conceptuel, cette réponse n'est qu'à demi satisfaisante : en pratique, il est nécessaire de s'assurer que le système considéré est borné pour pouvoir l'implanter. Nous avons donc proposé un *test de non-bornitude* qui indique si le système est borné, ou s'il admet un comportement infini périodique qui le rend non borné. Cette méthode de décision partielle a été implantée dans l'outil TETU.

Ensuite, nous nous sommes interessés à des aspects plus pratiques de la vérification. En particulier, nous considérons les *applications temps-réel* : des systèmes réactifs dont la correction dépend particulièrement des délais nécessaires à l'application pour réagir aux sollicitations de son environnement. Nous avons introduit la spécification temporelle de l'environnement des programmes Electre sous la forme de la durée des modules et de la fréquence d'occurrence des événements qui le composent. Ceci nous conduit aux *systèmes à file réactifs hybride linéaires (SFR Hybrides Linéaires)*. Tout environnement périodique pouvant être exprimé dans ce formalisme, le problème de la bornitude n'est évidemment pas décidable pour les SFR Hybrides Linéaires avec (au moins) 2 événements mémorisables. Par conséquent, nous avons adapté notre test de non-bornitude à ce contexte.

Finalement, lorsque le modèle est borné, il n'est pas pour autant possible de le vérifier pratiquement, à cause de sa taille. Nous avons donc défini une méthode de réduction, qui, lorsqu'elle s'applique, conduit à un gain fortement significatif. Notre technique a été implantée puis utilisée avec succès dans le cadre de la vérification d'une application de taille réelle, avec, pour les programmes le permettant, un gain dépassant 99% de la taille du modèle sans réduction.

Perspectives. Il reste cependant un certain nombre de problèmes à résoudre. Tout d'abord, les problèmes de la décidabilité du model-checking des logiques Ltl et Ctl pour les Lossy SFR Embarqués avec 2 (et 3 pour Ctl) événements mémorisables. À notre connaissance, ces problèmes sont actuellement ouverts pour la classe de systèmes plus commune des Lossy machines à compteurs.

Ensuite, de nombreuses voies «plus pratiques» restent à explorer :

- Nous avons considéré des spécifications temporelles simples, et nous souhaitons, à l'avenir, en considérer de plus expressives. Par exemple, en liant les occurrences d'événements à l'exécution des modules afin de représenter les communications internes à l'application modélisée, ou encore la possibilité de lier les occurrences d'événements les unes aux autres. C'est par exemple le cas des événements signalant les niveaux haut et bas d'un réservoir, dont les occurrences, haut puis bas, sont liées par les lois physiques de la dynamique des fluides, et fonction des modes d'alimentation et de vidange du réservoir.
- En considérant ce contexte plus général, nous souhaitons implanter le test de non-bornitude pour les SFR Hybrides Linéaires dans notre outil TeTU afin d'étudier la bornitude d'applications réelles.
- Dans l'optique de la conception des applications temps-réel, il est aussi interessant de considérer que certaines spécifications temporelles sont paramétrées. Ceci dans le but de calculer la durée maximale d'un module, ou la fréquence d'occurrence minimale d'un événement, pour que la file reste bornée, ne dépassant pas une taille fixée.
- Notre étude est influencée par le langage Electre, mais nous espérons utiliser nos résultats pour d'autres langages. En particulier, pour les langages synchrones, la bornitude du gestionnaire d'événement (qui aplanit les différences dans les rythmes d'évolution de l'application et de son environnement) est un problème clef pour leur implantation. Nous espérons pouvoir utiliser notre test en ce sens. De plus, la sémantique Fiffo est très proche de la gestion des files dans le langage Sdl [65], utilisé pour la spécification

des applications réactives ou temps-réel. Nous proposons d'étudier l'adéquation des SFR Embarqués pour donner la sémantique des spécifications Sdl.

– Enfin, nous avons discuté l'inefficacité du processus de vérification actuellement utilisée pour les programmes Electre. La construction statique du modèle pour la file de mémorisation est la source principale d'explosion combinatoire, tout en créant un formidable gaspillage puisqu'en général, seule une infime partie du modèle est utilisée. Il serait donc préférable de construire le modèle dynamiquement : pendant la vérification. Ceci impose d'implanter un outil de vérification spécialisé pour les systèmes à file réactifs. L'ensemble des techniques présentées dans notre thèse pourraient alors y être incluses.

Bibliographie

[1] P. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 160–170. IEEE Computer Society Press, 1993.

[2] P. Abdulla and B. Jonsson. Undecidable verification problems for programs with unreliable channels. *Lecture Notes in Computer Science*, 820 :316– ? ?, 1994.

[3] P. A. Abdulla, K. Čerāns, B. Jonsson, and T. Yih-Kuen. General decidability theorems for infinite-state systems. In *Proceedings, 11*$^{\text{th}}$ *Annual IEEE Symposium on Logic in Computer Science (LICS'96)*, pages 313–321, New Brunswick, NJ, USA, July 1996.

[4] P. A. Abdulla, K. Čerāns, B. Jonsson, and T. Yih-Kuen. Algorithmic analysis of programs with well quasi-ordered domains. *INFCTRL : Information and Computation (formerly Information and Control)*, 160, 2000.

[5] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proc. IEEE 5th Symp. Logic in Computer Science*, 1990.

[6] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1) :3–34, 6 February 1995.

[7] R. Alur, T.A. Henzinger, and P-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3) :181–201, March 1996.

[8] R. Alur, T.A. Henzinger, and P.S. Ho. Automatic symbolic verification of embedded systems. In *Proc. 14th annual Real-Time Systems Symposium*, 1993.

[9] A. Arnold. *Systemes de Transitions Finis et Semantique de Processus Communicants*. Masson, 1992.

[10] M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Acta Informatica*, 20(3) :207–226, December 1983.

[11] A. Benveniste and G. Berry. The synchronous approach to reactive and real–time systems. *IEEE Trans. Autom. Control*, 9(79) :1270–1282, September 1991.

[12] G. Berry. Real time programming : Special purpose or general purpose languages. In *IFIP World Computer Congress*, pages 11–17, San Francisco, 1989.

[13] G. Berry, P. Couronné, and G. Gonthier. Synchronous programming of reactive systems, an introduction to ESTEREL. In K. Fuchi and M. Nivat, editors, *Programming of Future Generation Computers*. Elsevier Science Publisher B.V., North Holland, 1988.

[14] V. Bertin, M. Poize, J. Pulou, and J. Sifakis. Towards validated real-time software. In *Proceedings of the 12 th Euromicro Conference on Real Time Systems*, pages 157–164, Stockholm, June 2000.

[15] F. Bertrand. Détection et traitement des erreurs syntaxiques et sémantiques dans le langage temps-réel ELECTRE. DEA AIA, July 1992.

[16] P. Boisieau. *Vérification et exécution d'applications temps-réel industrielles avec ELECTRE*. PhD thesis, École Centrale de Nantes, 1999.

[17] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata : Application to model-checking. In *Proc. 8th Int. Conf. Concurrency Theory (CONCUR'97), Warsaw, Poland, Jul. 1997*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.

[18] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2) :323–342, April 1983.

[19] O. Burkart and J. Esparza. More infinite results. *BEATCS : Bulletin of the European Association for Theoretical Computer Science*, 62, 1997.

[20] F. Cassez. *Compilation et vérification de programmes ELECTRE*. PhD thesis, Thèse de l'Université de Nantes et de l'École Centrale de Nantes, FRANCE, March 1993.

[21] F. Cassez, F. Herbreteau, and O. Roux. Reactive systems with unbounded event memorization. In *Conférence Africaine de Recherche en Informatique (CARI'00)*, Antananarivo, Madagascar, October 2000.

[22] F. Cassez and K.G. Larsen. The impressive power of stopwatches. In *International Conference on Concurrency Theory*, pages 138–152, 2000.

[23] F. Cassez and O. Roux. Compilation of the ELECTRE reactive language into finite transition systems. *Theoretical Computer Science*, 146, 1995.

[24] G. Cécé, A. Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1) :20–31, 1996.

[25] E.M. Clarke and E.A. Emerson. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In D. Kozen, editor, *Proceedings of the Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71, Yorktown Heights, New York, May 1981. Springer-Verlag.

[26] E. Closse, M. Poize, J. Pulou, J. Sifakis, P. Venter, D. Weil, and S. Yovine. TAXYS : A tool for the development and verification of real-time embedded systems. In *Proceedings of the 13th Conference on Computer Aided Verification*, pages 391–395, Paris, July 2001.

[27] P. J. Courtois, F. Heymans, and D. L. Parnas. Concurrent control with "readers" and "writers". *Communications of the ACM*, 14(10) :667–668, October 1971.

[28] C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *Proc. 25th Int. Coll. Automata, Languages, and Programming (ICALP'98), Aalborg, Denmark, July 1998*, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115. Springer, 1998.

[29] C. Dufourd, P. Jančar, and Ph. Schnoebelen. Boundedness of reset p/t nets. In *Proc. 26th Int. Coll. Automata, Languages, and Programming (ICALP'99), Prague, Czech Rep.*, volume 1644 of *Lecture Notes in Computer Science*, pages 301–310. Springer, July 1999.

[30] B. Dutertre, M. Le Borgne, A. Benveniste, and P. Le Guernic. Programming and proving discrete events systems with SIGNAL. Réunion annuelle du G.R. Automatique du C.N.R.S., September 1992.

[31] J.P. Elloy and O. Roux. ELECTRE : A language for control structuring in real time. *The Computer Journal*, 28(5), July 1985.

[32] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Formal models and semantics*, volume B of *Handbook of Theoretical Computer Science*, pages 995–1072. Elsevier, 1994.

[33] E.A. Emerson and J. Srinivasan. Branching time temporal logic. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models of Concurrency*, number vol. 354 in Lecture Notes in Computer Science, pages 123–172. Springer, Heidelberg, Germany, 1989.

[34] F. Boussinot and R. De Simone. The ESTEREL language. *Proc. IEEE*, 79(9) :1293–1304, September 1991.

[35] A. Finkel. *Structuration des Systèmes de Transitions. Applications au Contrôle du Parallélisme par File FIFO.* Thèse de docteur d'état, Université de Paris-Sud, Orsay, France, June 1986.

[36] A. Finkel. A generalization of the procedure of Karp and Miller to well structured transition systems. In Thomas Ottmann, editor, *Proceedings of the 14th International Colloquium on Automata, Languages, and Programming*, volume 267 of *LNCS*, pages 499–508, Karlsruhe, FRG, July 1987. Berlin : Springer.

[37] A. Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89(2) :144–179, December 1990.

[38] A. Finkel. Decidability of the termination problem for completely specified protocols. *Distributed Computing*, 7(3) :129–135, 1994.

[39] A. Finkel and O. Marcé. Verification of infinite regular communicating automata. Technical report, LSV, ENS Cachan, France, 1996.

[40] A. Finkel and P. McKenzie. Verifying identical communicating processes is undecidable. *Theoretical Computer Science*, 174(1–2) :217–230, March 1997.

[41] A. Finkel and Ph. Schnoebelen. Well structured transition systems everywhere ! *Theoretical Computer Science*, 256(1–2) :63–92, 2001.

[42] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *Proc. 2nd Int. Workshop on Verification of Infinite State Systems (INFINITY'97), Bologna, Italy, July 1997*, volume 9 of *Electronic Notes in Theor. Comp. Sci.*, pages 30–40. Elsevier Science, 1997.

[43] E. Fleury. *Automates temporisés avec mises à jour.* PhD thesis, École Normale Supérieure de Cachan, 2001. à paraître.

[44] T. Gautier, P. le Guernic, and L. Besnard. SIGNAL : A declarative language for synchronous programming of real-time systems. In G. Kahn, editor, *Functional Programming Languages and Computer Architecture*, pages 257–277. Springer-Verlag, Berlin, DE, 1987.

[45] P. Godefroid. Using partial orders to improve automatic verification methods. In E.M. Clarke, editor, *Proceedings of the 2nd International Conference on Computer-Aided Verification (CAV '90), Rutgers, New Jersey, 1990*, number 531 in Lecture Notes in Computer Science, pages 176–185, Berlin-Heidelberg-New York, 1991. Springer.

[46] P. Godefroid. *Partial-order methods for the verification of concurrent systems : an approach to the state-explosion problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer-Verlag Inc., New York, NY, USA, 1996.

[47] P. Godefroid, G.J. Holzmann, and D. Pirottin. State space caching revisited. In *Proc. 4th International Computer Aided Verification Conference*, pages 178–191, 1992.

[48] P. Godefroid and Wolper. P. A partial approach to model checking. In *Proceedings, Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 406–415, Amsterdam, The Netherlands, 15–18 July 1991. IEEE Computer Society Press.

[49] P. Godefroid and D. Pirottin. Refining dependencies improves partial-order verification methods (extended abstract). In *Proceedings of the 5th International Conference on Computer Aided Verification, Greece*, number 697 in Lecture Notes in Computer Science, pages 409–423, Berlin-Heidelberg-New York, 1993. Springer.

[50] P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In Kim G. Larsen and Arne Skou, editors, *Proceedings of Computer Aided Verification (CAV '91)*, volume 575 of *LNCS*, pages 332–342, Berlin, Germany, July 1992. Springer.

[51] P. Godefroid and P. Wolper. Partial-order methods for temporal verification. *CONCUR '93 Proceedings Lecture Notes in Computer Science*, 715 :233–246, August 1993.

[52] N. Halbwachs. Delay analysis in synchronous programs. In *Proc. 5th Conference on Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, 1993.

[53] N. Halbwachs. *Synchronous programming of reactive systems*. Kluwer Academic Pub., 1993.

[54] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9), September 1991.

[55] N. Halbwachs, P. Caspi, P. Raymond, and D. Pillaud. Programmation et vérification des systèmes réactifs : le langage LUSTRE. *Technique et Science Informatiques*, 10(2) :139–158, 1991.

[56] D. Harel. Statecharts : A visual formulation for complex systems. *Science of Computer Programming*, 8(3) :231–274, June 1987.

[57] D. Harel and A. Pnueli. On the development of reactive systems. In K.R. Apt, editor, *Logic and Models of Concurrent Systems, NATO, ASI-13*, pages 477–498, Berlin, 1985. Springer.

[58] T.A. Henzinger. The theory of hybrid automata. In *Proceedings, 11$^{\text{th}}$ Annual IEEE Symposium on Logic in Computer Science*, pages 278–292, New Brunswick, New Jersey, 27–30 July 1996. IEEE Computer Society Press.

[59] T.A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata ? In ACM, editor, *Proceedings of the twenty-seventh annual ACM Symposium on Theory of Computing : Las Vegas, Nevada*, pages 373–382, New York, NY, USA, 1995. ACM Press.

[60] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *Proceedings, Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 394–406, Santa Cruz, California, 22–25 June 1992. IEEE Computer Society Press.

[61] F. Herbreteau, F. Cassez, and O. Roux. Application of partial-order methods to reactive programs with event memorization. *Journal of Real-time Systems*, 2000.

[62] G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society (3)*, 2(7) :326–336, September 1952.

[63] G.J. Holzmann. The Spin model checker. *IEEE Transactions on Software Engineering*, 23(5) :279–95, May 1997.

[64] G.J. Holzmann, P. Godefroid, and D. Pirottin. Coverage preserving reduction strategies for reachability analysis. In *Proc. 12th Int. Conf on Protocol Specification, Testing, and Verification, INWG/IFIP*, Orlando, Fl., June 1992.

[65] International Telecommunication Union (ITU). Specification and description language (SDL). Recommendation Z.100, November 1999.

[66] Th. Jéron and C. Jard. Testing for unboundedness of fifo channels. *Theoretical Computer Science*, 113(1) :93–117, 24 May 1993.

[67] M. Jourdan, F. Maraninchi, and A. Olivero. Verifying quantitative real-time properties of synchronous programs. In *Proc. 5th Conference on Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, 1993.

[68] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4 :373–395, 1984.

[69] S. Katz and D. Peled. Defining conditional independence using collapses. *Theoretical Computer Science*, 101(2) :337–359, July 1992.

[70] S. Katz and D. Peled. Verification of distributed programs using representative interleaving sequences. *Distributed Computing*, 6 :107–120, 1992.

[71] M. Kourjanski and P. Varaiya. Stability of hybrid systems. *Lecture Notes in Computer Science*, 1066 :413– ? ?, 1996.

[72] O. Kushnarenko and Ph. Schnoebelen. A model for recursive-parallel programs. In *Proc. 1st Int. Workshop on Verification of Infinite State Systems (INFINITY'96)*, Pisa, Italy, August 1996. Elsevier.

[73] L. Lamport. What good is temporal logic ? In R. E. A. Mason, editor, *Proceedings of the IFIP Congress on Information Processing*, pages 657–667, Amsterdam, 1983. North-Holland.

[74] V. Lecompte. *Vérification automatique de programmes ESTEREL*. PhD thesis, Université de Paris VII, FRANCE, 1989.

[75] F. Maraninchi. Argonaute : Graphical descripton, semantics and verification of reactive systems by using a process algebra. In J. Sifakis, editor, *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 38–53, Berlin, June 1990. Springer.

[76] A. Mateescu and A. Salomaa. Formal languages : an introduction and a synopsis. In G. Rozenberg and A. Salomaa, editors, *Word, Language, Grammar*, number vol. 1 in Handbook of Formal Languages, pages 1–39. Springer, 1997.

[77] R. Mayr. Undecidable problems in unreliable computations. In *International Symposium on Latin American Theoretical Informatics (LATIN'2000)*, volume 1776 of *Lecture Notes in Computer Science*, Punta del Este, Uruguay, 2000. Springer-Verlag. Une version longue de cet article doit paraître dans Theoretical Computer Science.

[78] A. Mazurkiewicz. Trace theory. In *Petri Nets : Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part II : Proceedings of an Advanced Course*, volume 255 of *Lecture Notes in Computer Science*, pages 279–324, 1986.

[79] K. Mc Millan. Using unfolding to avoid the state explosion problem in the verification of asynchronous circuits. In *Proc. 4th International Computer Aided Verification Conference*, Montreal, June 1992.

[80] M. L. Minsky. *Computation : Finite and Infinite Machines*. Prentice Hall, London, 1 edition, 1967.

[81] F. Moller. Infinite results. In U. Montanari and V. Sassone, editors, *CONCUR '96 : Concurrency Theory, 7th International Conference*, volume 1119 of *Lecture Notes in Computer Science*, pages 195–216, Pisa, Italy, 26–29 August 1996. Springer-Verlag.

[82] W.T. Overman. *Verification of concurrent systems : function and timing*. PhD thesis, Universisty of California, Los Angeles, 1981.

[83] R.J. Parikh. On context-free languages. *Journal of the ACM*, 13(4) :570–581, October 1966.

[84] D. Peled. All from one, one from all : on model checking using representatives. In *Proceedings of the 5th International Conference on Computer Aided Verification, Greece*, number 697 in Lecture Notes in Computer Science, pages 409–423, Berlin-Heidelberg-New York, 1993. Springer.

[85] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, University of Aarhus, Denmark, 1981.

[86] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*, pages 46–57, Providence, Rhode Island, October 31–November 2 1977. IEEE, IEEE Computer Society Press.

[87] A. Pnueli. Specification and development of reactive systems. *Information Processing*, 1986.

[88] O. Roux, D. Creusot, F. Cassez, and J.P. Elloy. Le langage réactif asynchrone ELECTRE. *Technique et Science Informatiques (TSI)*, 11(5) :35–66, 1992.

[89] V. Rusu. *Vérification temporelle de programmes ELECTRE*. PhD thesis, École Centrale de Nantes, 1996.

[90] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logic. *Journal of the Association for Computing Machinery*, 32(3) :733–749, July 1985.

[91] G. Sutre. Vérification de propriétés sur les automates à file réactifs produits par compilation de programmes ELECTRE. Mémoire de DEA, September 1997.

[92] G. Sutre, A. Finkel, O. Roux, and F. Cassez. Effective recognizability and model checking of reactive fiffo automata. In *Proc. 7th Int. Conf. Algebraic Methodology and Software Technology (AMAST'98), Amazonia, Brazil, Jan. 1999*, volume 1548 of *Lecture Notes in Computer Science*, pages 106–123. Springer, 1999.

[93] A. Valmari. Stubborn sets for reduced state space generation. *LNCS 483 : Advances in Petri Nets'90*, 1991.

[94] B. Vauquelin and P. Franchi-Zannettacci. Automates a file. *Theoretical Computer Science*, 11(2) :221–225, June 1980. Note.

Annexe A

Compléments sur le langage Electre

En complément de la description du langage de la section 2.1, page 34, nous donnons ici sa grammaire (section A.1) et les règles de récriture qui permettent la compilation des programmes (section A.2). Il reste néanmoins certains points importants du langage que nous tenons pour implicites car le but de ce mémoire n'est pas d'épiloguer sur ELECTRE. Par exemple, un événement donné qui apparaît plusieurs fois dans un même programme doit toujours être qualifié de la même façon, et le programme : `await e : loop [ >A when #e]`, n'est donc pas correct. Cette sémantique "intuitive" du langage a été étudiée dans [15]. Le compilateur ELECTRE est bien sûr muni d'un vérificateur de la correction sémantique des programmes.

A.1 Grammaire du langage Electre

Nous notons en majuscules $P, C, R, G, \ldots$ les non terminaux de la grammaire. Les terminaux représentés ici sous la forme *id* sont des suites finies de caractères alphanumériques qui représentent un nom d'événement ou de module suivant les cas. Le non terminal initial est P.

$$
\begin{array}{lll}
P & ::= & C \\
 & | & C \parallel P \\
C & ::= & R \\
 & | & R \; ; C \\
 & | & R \; \texttt{await} \; I \\
 & | & R \; \texttt{when} \; I \\
R & ::= & G \\
 & | & \texttt{loop} \; G \\
G & ::= & M \\
 & | & [\, P \,] \\
I & ::= & K \\
 & | & K : C \\
K & ::= & E \\
 & | & \{\, J \,\}
\end{array}
$$

$$
\begin{array}{lcl}
J & ::= & I \\
 & | & I \mid J \\
 & | & I \mid\mid J \\
 & | & I \mid\mid\mid J \\
E & ::= & id \\
 & | & @\,id \\
 & | & \#\,id \\
 & | & \$\,id \\
M & ::= & id \\
 & | & >id \\
 & | & !\,id \\
 & | & \texttt{idle}
\end{array}
$$

Notons que le mot clef `idle` n'est pas nécessaire : il peut être oublié et le programme ainsi obtenu reste syntaxiquement correct. Par exemple : `await e` est équivalent à : `idle await e`.

A.2 Sémantique opérationnelle du langage Electre

Les règles de récriture du langage ELECTRE sont notées suivant le formalisme SOS introduit dans [85] qui s'interprète de la façon suivante :

$$\frac{\{\texttt{e}\} \vdash A_1 \xrightarrow{*} a_1 \rhd A'_1 \xrightarrow{*} a'_1 \wedge \cdots \wedge A_n \xrightarrow{*} a_n \rhd A'_n \xrightarrow{*} a'_n}{\{\texttt{e}\} \vdash B_1 \xrightarrow{*} b_1 \rhd B'_1 \xrightarrow{*} b'_1 \wedge \cdots \wedge B_k \xrightarrow{*} b_k \rhd B'_k \xrightarrow{*} b'_k}$$

"Dans le contexte de l'occurrence d'un événement `e`, et sachant que la dérivation $A_1 \xrightarrow{*} a_1$ se récrit en $A'_1 \xrightarrow{*} a'_1, \ldots,$ et la dérivation $A_n \xrightarrow{*} a_n$ se récrit en $A'_n \xrightarrow{*} a'_n$, nous déduisons que la dérivation $B_1 \xrightarrow{*} b_1$ se récrit en $B'_1 \xrightarrow{*} b'_1, \ldots,$ et la dérivation $B_k \xrightarrow{*} b_k$ se récrit en $B'_k \xrightarrow{*} b'_k$."

Une dérivation $A_1 \xrightarrow{*} a_1$ constitue simplement un chemin d'un non terminal A_1 de la grammaire considérée à un terminal a_1. Une dérivation est directe, et notée $A_1 \rightarrow a_1$ lorsqu'elle n'implique qu'une seule des règles de la grammaire. Dans notre cas, nous considérons par la suite la grammaire d'ELECTRE décrite en section A.1, page 193.

Règle i : $P ::= C$

$$\text{(i.1)} \quad \frac{\{\texttt{e}\} \vdash C \xrightarrow{*} m \rhd C' \xrightarrow{*} \texttt{nil}}{\{\texttt{e}\} \vdash P \xrightarrow{*} m \rhd P' \xrightarrow{*} \texttt{nil}}$$

$$\text{(i.2)} \quad \frac{\{\texttt{e}\} \vdash C \xrightarrow{*} m \rhd C' \xrightarrow{*} m'}{\{\texttt{e}\} \vdash P \xrightarrow{*} m \rhd P' \xrightarrow{*} m'}$$

Règle ii : $P_1 ::= C \mid\mid P_2$

$$\text{(ii.1)} \quad \frac{\{\texttt{e}\} \vdash C \xrightarrow{*} m \rhd C' \xrightarrow{*} m' \wedge P_2 \xrightarrow{*} p \rhd P'_2 \xrightarrow{*} p'}{\{\texttt{e}\} \vdash P_1 \xrightarrow{*} m \mid\mid p \rhd P'_1 \xrightarrow{*} m' \mid\mid p'}$$

$$\text{(ii.2)}\quad \frac{\{\mathtt{e}\} \vdash C \xrightarrow{*} m \rhd C' \xrightarrow{*} \mathtt{nil} \wedge P_2 \xrightarrow{*} p \rhd P_2' \xrightarrow{*} p'}{\{\mathtt{e}\} \vdash P_1 \xrightarrow{*} m \mid\mid p \rhd P_1' \xrightarrow{*} p'}$$

$$\text{(ii.3)}\quad \frac{\{\mathtt{e}\} \vdash C \xrightarrow{*} m \rhd C' \xrightarrow{*} m' \wedge P_2 \xrightarrow{*} p \rhd P_2' \xrightarrow{*} \mathtt{nil}}{\{\mathtt{e}\} \vdash P_1 \xrightarrow{*} m \mid\mid p \rhd P_1' \xrightarrow{*} m'}$$

$$\text{(ii.4)}\quad \frac{\{\mathtt{e}\} \vdash C \xrightarrow{*} m \rhd C' \xrightarrow{*} \mathtt{nil} \wedge P_2 \xrightarrow{*} p \rhd P_2' \xrightarrow{*} \mathtt{nil}}{\{\mathtt{e}\} \vdash P_1 \xrightarrow{*} m \mid\mid p \rhd P_1' \xrightarrow{*} \mathtt{nil}}$$

Règle iii : $C ::= R$

$$\text{(iii.1)}\quad \frac{\{\mathtt{e}\} \vdash R \xrightarrow{*} r \rhd R' \xrightarrow{*} r'}{\{\mathtt{e}\} \vdash C \xrightarrow{*} r \rhd C' \xrightarrow{*} r'}$$

$$\text{(iii.2)}\quad \frac{\{\mathtt{e}\} \vdash R \xrightarrow{*} r \rhd R' \xrightarrow{*} \mathtt{nil}}{\{\mathtt{e}\} \vdash C \xrightarrow{*} r \rhd C' \xrightarrow{*} \mathtt{nil}}$$

Règle iv : $C_1 ::= R \; ; C_2$

Notons que dans ce cas, et conformément à la notion de séquentialité exprimée par l'opérateur " ; ", seule la transformation de R est possible.

$$\text{(iv.1)}\quad \frac{\{\mathtt{e}\} \vdash R \xrightarrow{*} r \rhd R' \xrightarrow{*} r' \wedge C_2 \xrightarrow{*} c \rhd C_2' \rightarrow c'}{\{\mathtt{e}\} \vdash C_1 \xrightarrow{*} r \; ; c \rhd C_1' \xrightarrow{*} r' \; ; c}$$

$$\text{(iv.2)}\quad \frac{\{\mathtt{e}\} \vdash R \xrightarrow{*} r \rhd R' \xrightarrow{*} \mathtt{nil} \wedge C_2 \xrightarrow{*} c \rhd C_2' \rightarrow c'}{\{\mathtt{e}\} \vdash C_1 \xrightarrow{*} r \; ; c \rhd C_1' \xrightarrow{*} c'}$$

$$\text{(iv.3)}\quad \frac{\{\mathtt{e}\} \vdash R \xrightarrow{*} r \rhd R' \xrightarrow{*} r' \wedge C_2 \xrightarrow{*} c \rhd C_2' \rightarrow \mathtt{nil}}{\{\mathtt{e}\} \vdash C_1 \xrightarrow{*} r \; ; c \rhd C_1' \xrightarrow{*} r' \; ; c}$$

Règle v : $C ::= R$ `await` I

La transformation du non-terminal I en C dans la règle (v4) s'explique par la règle (xii3).

$$\text{(v.1)}\quad \frac{\{\mathtt{e}\} \vdash R \xrightarrow{*} m \rhd R' \xrightarrow{*} m' \wedge I \xrightarrow{*} i \rhd I' \xrightarrow{*} i'}{\{\mathtt{e}\} \vdash C \xrightarrow{*} m \;\mathtt{await}\; i \rhd C' \xrightarrow{*} m' \;\mathtt{await}\; i'}$$

$$\text{(v.2)}\quad \frac{\{\mathtt{e}\} \vdash R \xrightarrow{*} m \rhd R' \xrightarrow{*} \mathtt{nil} \wedge I \xrightarrow{*} i \rhd I' \xrightarrow{*} i'}{\{\mathtt{e}\} \vdash C \xrightarrow{*} m \;\mathtt{await}\; i \rhd C' \xrightarrow{*} \;\mathtt{await}\; i'}$$

$$\text{(v.3)}\quad \frac{\{\mathtt{e}\} \vdash R \xrightarrow{*} m \rhd R' \xrightarrow{*} m' \wedge I \xrightarrow{*} i \rhd I' \xrightarrow{*} \mathtt{nil}}{\{\mathtt{e}\} \vdash C \xrightarrow{*} m \;\mathtt{await}\; i \rhd C' \xrightarrow{*} \mathtt{nil}}$$

$$\text{(v.4)}\quad \frac{\{\mathtt{e}\} \vdash R \xrightarrow{*} m \rhd R' \xrightarrow{*} m' \wedge I \xrightarrow{*} i \rhd C'' \xrightarrow{*} p'}{\{\mathtt{e}\} \vdash C \xrightarrow{*} m \;\mathtt{await}\; i \rhd C' \xrightarrow{*} p'}$$

Règle vi : $C ::= R$ `when` I

$$\text{(vi.1)}\quad \frac{\{\mathtt{e}\} \vdash R \xrightarrow{*} m \rhd R' \xrightarrow{*} m' \wedge I \xrightarrow{*} i \rhd I' \xrightarrow{*} i'}{\{\mathtt{e}\} \vdash C \xrightarrow{*} m \;\mathtt{when}\; i \rhd C' \xrightarrow{*} m' \;\mathtt{when}\; i'}$$

$$\text{(vi.2)}\quad \frac{\{\mathtt{e}\} \vdash R \xrightarrow{*} m \rhd R' \xrightarrow{*} \mathtt{nil} \wedge I \xrightarrow{*} i \rhd I' \xrightarrow{*} i'}{\{\mathtt{e}\} \vdash C \xrightarrow{*} m \;\mathtt{when}\; i \rhd C' \xrightarrow{*} \mathtt{nil}}$$

La comparaison des règles (v2) et (vi2) met en valeur la différence entre les opérateurs `await` et `when`.

$$\text{(vi.3)}\quad \frac{\{\mathtt{e}\} \vdash R \xrightarrow{*} m \rhd R' \xrightarrow{*} m' \wedge I \xrightarrow{*} i \rhd I' \xrightarrow{*} \mathtt{nil}}{\{\mathtt{e}\} \vdash C \xrightarrow{*} m \;\mathtt{when}\; i \rhd C' \xrightarrow{*} \mathtt{nil}}$$

$$\text{(vi.4)}\quad \frac{\{\mathtt{e}\} \vdash R \xrightarrow{*} m \rhd R' \xrightarrow{*} m' \wedge I \xrightarrow{*} i \rhd C'' \xrightarrow{*} p'}{\{\mathtt{e}\} \vdash C \xrightarrow{*} m \;\mathtt{when}\; i \rhd C'' \xrightarrow{*} p'}$$

Règle vii : $R ::= G$

$$\text{(vii.1)}\quad \frac{\{\mathtt{e}\} \vdash G \xrightarrow{*} p \rhd G' \xrightarrow{*} p'}{\{\mathtt{e}\} \vdash R \xrightarrow{*} p \rhd R' \xrightarrow{*} p'}$$

$$\text{(vii.2)}\quad \frac{\{\mathtt{e}\} \vdash G \xrightarrow{*} p \rhd G' \xrightarrow{*} \mathtt{nil}}{\{\mathtt{e}\} \vdash R \xrightarrow{*} p \rhd R' \xrightarrow{*} \mathtt{nil}}$$

Règle viii : $R ::= \mathtt{loop}\; G_1\,(G_2)$

Lors de la compilation une structure répétitive de la forme `loop` G est récrite sous la forme `loop` $G_1\,(G_2)$ où G_1 est la récriture de G obtenue lors de cette étape de compilation, et G_2 est égal à G. G_1 représente donc l'exécution courante de G et G_2, ses exécutions futures. Bien entendu, G_2 n'a pas la possibilité d'évoluer lors de ces récritures.

$$\text{(viii.1)}\quad \frac{\{\mathtt{e}\} \vdash G_1 \xrightarrow{*} p_1 \rhd G_1' \xrightarrow{*} p_1' \wedge G_2 \xrightarrow{*} p_2 \rhd G_2' \xrightarrow{*} p_2}{\{\mathtt{e}\} \vdash R \xrightarrow{*} \mathtt{loop}\; p_1\,(p_2) \rhd R' \xrightarrow{*} \mathtt{loop}\; p_1'\,(p_2)}$$

$$\text{(viii.2)}\quad \frac{\{\mathtt{e}\} \vdash G_1 \xrightarrow{*} p_1 \rhd G_1' \xrightarrow{*} \mathtt{nil} \wedge G_2 \xrightarrow{*} p_2 \rhd G_2' \xrightarrow{*} p_2}{\{\mathtt{e}\} \vdash R \xrightarrow{*} \mathtt{loop}\; p_1\,(p_2) \rhd R' \xrightarrow{*} \mathtt{loop}\; p_2\,(p_2)}$$

Règle ix : $G ::= M$

$$\text{(ix.1)}\quad \frac{\{\mathtt{e}\} \vdash M \xrightarrow{*} m \rhd M' \xrightarrow{*} m'}{\{\mathtt{e}\} \vdash G \xrightarrow{*} m \rhd G' \xrightarrow{*} m'}$$

$$\text{(ix.2)}\quad \frac{\{\mathtt{e}\} \vdash M \xrightarrow{*} m \rhd M' \xrightarrow{*} \mathtt{nil}}{\{\mathtt{e}\} \vdash G \xrightarrow{*} m \rhd G' \xrightarrow{*} \mathtt{nil}}$$

Règle x : $G ::= [\,P\,]$

$$\text{(x.1)}\quad \frac{\{\mathtt{e}\} \vdash P \xrightarrow{*} p \rhd P' \xrightarrow{*} p'}{\{\mathtt{e}\} \vdash G \xrightarrow{*} [\,p\,] \rhd G' \xrightarrow{*} [\,p'\,]}$$

$$\text{(x.2)}\quad \frac{\{\mathtt{e}\} \vdash P \xrightarrow{*} p \rhd P' \xrightarrow{*} \mathtt{nil}}{\{\mathtt{e}\} \vdash G \xrightarrow{*} [\,p\,] \rhd G' \xrightarrow{*} \mathtt{nil}}$$

Règle xi : I : :=K

(xi.1) $$\frac{\{\mathtt{e}\} \vdash K \overset{*}{\longrightarrow} e \rhd K' \overset{*}{\longrightarrow} e'}{\{\mathtt{e}\} \vdash I \overset{*}{\longrightarrow} e \rhd I' \overset{*}{\longrightarrow} e'}$$

(xi.2) $$\frac{\{\mathtt{e}\} \vdash K \overset{*}{\longrightarrow} e \rhd K' \overset{*}{\longrightarrow} \mathtt{nil}}{\{\mathtt{e}\} \vdash I \overset{*}{\longrightarrow} e \rhd I' \overset{*}{\longrightarrow} \mathtt{nil}}$$

Règle xii : I : :=$K : C$

De même que pour les règles (iv1) à (iv3) concernant l'opérateur séquentiel ";", toute transformation du non terminal C ne peut se produire que lorsque le non terminal K a été réduit par la dérivation $K \overset{*}{\longrightarrow} \mathtt{nil}$.

(xii.1) $$\frac{\{\mathtt{e}\} \vdash K \overset{*}{\longrightarrow} e \rhd K' \overset{*}{\longrightarrow} e' \wedge C \overset{*}{\longrightarrow} c \rhd C' \overset{*}{\longrightarrow} c'}{\{\mathtt{e}\} \vdash I \overset{*}{\longrightarrow} e : c \rhd I' \overset{*}{\longrightarrow} e' : c'}$$

(xii.2) $$\frac{\{\mathtt{e}\} \vdash K \overset{*}{\longrightarrow} e \rhd K' \overset{*}{\longrightarrow} e' \wedge C \overset{*}{\longrightarrow} c \rhd C' \overset{*}{\longrightarrow} \mathtt{nil}}{\{\mathtt{e}\} \vdash I \overset{*}{\longrightarrow} e : c \rhd I' \overset{*}{\longrightarrow} e' : c'}$$

(xii.3) $$\frac{\{\mathtt{e}\} \vdash K \overset{*}{\longrightarrow} e \rhd K' \overset{*}{\longrightarrow} \mathtt{nil} \wedge C \overset{*}{\longrightarrow} c \rhd C' \overset{*}{\longrightarrow} c'}{\{\mathtt{e}\} \vdash I \overset{*}{\longrightarrow} e : c \rhd C'' \overset{*}{\longrightarrow} c'}$$

Règle xiii : K : :=E

(xiii.1) $$\frac{\{\mathtt{e}\} \vdash E \overset{*}{\longrightarrow} i \rhd E' \overset{*}{\longrightarrow} i'}{\{\mathtt{e}\} \vdash K \overset{*}{\longrightarrow} i \rhd K' \overset{*}{\longrightarrow} i'}$$

(xiii.2) $$\frac{\{\mathtt{e}\} \vdash E \overset{*}{\longrightarrow} i \rhd E' \overset{*}{\longrightarrow} \mathtt{nil}}{\{\mathtt{e}\} \vdash K \overset{*}{\longrightarrow} i \rhd K' \overset{*}{\longrightarrow} \mathtt{nil}}$$

Règle xiv : K : :=$\{ J \}$

(xiv.1) $$\frac{\{\mathtt{e}\} \vdash J \overset{*}{\longrightarrow} i \rhd J' \overset{*}{\longrightarrow} i'}{\{\mathtt{e}\} \vdash K \overset{*}{\longrightarrow} \{i\} \rhd K' \overset{*}{\longrightarrow} \{i'\}}$$

(xiv.2) $$\frac{\{\mathtt{e}\} \vdash J \overset{*}{\longrightarrow} i \rhd J' \overset{*}{\longrightarrow} \mathtt{nil}}{\{\mathtt{e}\} \vdash K \overset{*}{\longrightarrow} \{i\} \rhd K' \overset{*}{\longrightarrow} \mathtt{nil}}$$

Règle xv : J : :=I

(xv.1) $$\frac{\{\mathtt{e}\} \vdash I \overset{*}{\longrightarrow} i \rhd I' \overset{*}{\longrightarrow} i'}{\{\mathtt{e}\} \vdash J \overset{*}{\longrightarrow} i \rhd J' \overset{*}{\longrightarrow} i'}$$

(xv.2) $$\frac{\{\mathtt{e}\} \vdash I \overset{*}{\longrightarrow} i \rhd I' \overset{*}{\longrightarrow} \mathtt{nil}}{\{\mathtt{e}\} \vdash J \overset{*}{\longrightarrow} i \rhd J' \overset{*}{\longrightarrow} \mathtt{nil}}$$

(xv.3) $$\frac{\{\mathtt{e}\} \vdash I \overset{*}{\longrightarrow} i \rhd C \overset{*}{\longrightarrow} c}{\{\mathtt{e}\} \vdash J \overset{*}{\longrightarrow} i \rhd C' \overset{*}{\longrightarrow} c}$$

La récriture de $J \overset{*}{\longrightarrow} i$ en $C' \overset{*}{\longrightarrow} c$ se produit lorsque I se tranforme en une structure de module sous l'effet de l'opérateur ":" conformément à la régle (xii3).

Règle xvi : $J_1 ::= I \mid J_2$

$$(\text{xvi.1})\quad \frac{\{\mathtt{e}\} \vdash I \xrightarrow{*} i \rhd I' \xrightarrow{*} i' \wedge J_2 \xrightarrow{*} j \rhd J_2' \xrightarrow{*} j'}{\{\mathtt{e}\} \vdash J_1 \xrightarrow{*} i \mid j \rhd J_1' \xrightarrow{*} i' \mid j'}$$

$$(\text{xvi.2})\quad \frac{\{\mathtt{e}\} \vdash I \xrightarrow{*} i \rhd C \xrightarrow{*} c \wedge J_2 \xrightarrow{*} j \rhd J_2' \xrightarrow{*} j'}{\{\mathtt{e}\} \vdash J_1 \xrightarrow{*} i \mid j \rhd C' \xrightarrow{*} c}$$

$$(\text{xvi.3})\quad \frac{\{\mathtt{e}\} \vdash I \xrightarrow{*} i \rhd I' \xrightarrow{*} \mathtt{nil} \wedge J_2 \xrightarrow{*} j \rhd J_2' \xrightarrow{*} j'}{\{\mathtt{e}\} \vdash J_1 \xrightarrow{*} i \mid j \rhd J_1' \xrightarrow{*} \mathtt{nil}}$$

$$(\text{xvi.4})\quad \frac{\{\mathtt{e}\} \vdash I \xrightarrow{*} i \rhd I' \xrightarrow{*} i' \wedge J_2 \xrightarrow{*} j \rhd C \xrightarrow{*} c}{\{\mathtt{e}\} \vdash J_1 \xrightarrow{*} i \mid j \rhd C' \xrightarrow{*} c}$$

$$(\text{xvi.5})\quad \frac{\{\mathtt{e}\} \vdash I \xrightarrow{*} i \rhd I' \xrightarrow{*} i' \wedge J_2 \xrightarrow{*} j \rhd J_2' \xrightarrow{*} \mathtt{nil}}{\{\mathtt{e}\} \vdash J_1 \xrightarrow{*} i \mid j \rhd J_1' \xrightarrow{*} \mathtt{nil}}$$

La comparaison de ces cinq règles avec les dix suivantes (correspondant à la composition parallèle forte pour les cinq premières et à la composition parallèle faible pour les suivantes) met en valeur la différence entre les compositions exclusive (ci-dessus) et parallèles (ci-dessous).

Règle xvii : $J_1 ::= I \mid\mid J_2$

Malgré la surcharge de l'opérateur "$||$", il est aisé de distinguer les cas où il correspond à une composition d'événements de ceux où il sert à la composition de structures de programmes comme dans les règles (xvii2) et (xvii4).

$$(\text{xvii.1})\quad \frac{\{\mathtt{e}\} \vdash I \xrightarrow{*} i \rhd I' \xrightarrow{*} i' \wedge J_2 \xrightarrow{*} j \rhd J_2' \xrightarrow{*} j'}{\{\mathtt{e}\} \vdash J_1 \xrightarrow{*} i \mid\mid j \rhd J_1' \xrightarrow{*} i' \mid\mid j'}$$

$$(\text{xvii.2})\quad \frac{\{\mathtt{e}\} \vdash I \xrightarrow{*} i \rhd C \xrightarrow{*} c \wedge J_2 \xrightarrow{*} j \rhd J_2' \xrightarrow{*} j'}{\{\mathtt{e}\} \vdash J_1 \xrightarrow{*} i \mid\mid j \rhd C' \xrightarrow{*} c \mid\mid \mathtt{await}\ j'}$$

$$(\text{xvii.3})\quad \frac{\{\mathtt{e}\} \vdash I \xrightarrow{*} i \rhd I' \xrightarrow{*} \mathtt{nil} \wedge J_2 \xrightarrow{*} j \rhd J_2' \xrightarrow{*} j'}{\{\mathtt{e}\} \vdash J_1 \xrightarrow{*} i \mid\mid j \rhd J_1' \xrightarrow{*} j'}$$

$$(\text{xvii.4})\quad \frac{\{\mathtt{e}\} \vdash I \xrightarrow{*} i \rhd I' \xrightarrow{*} i' \wedge J_2 \xrightarrow{*} j \rhd C \xrightarrow{*} c}{\{\mathtt{e}\} \vdash J_1 \xrightarrow{*} i \mid\mid j \rhd C' \xrightarrow{*} \mathtt{await}\ i' \mid\mid c}$$

$$(\text{xvii.5})\quad \frac{\{\mathtt{e}\} \vdash I \xrightarrow{*} i \rhd I' \xrightarrow{*} i' \wedge J_2 \xrightarrow{*} j \rhd J_2' \xrightarrow{*} \mathtt{nil}}{\{\mathtt{e}\} \vdash J_1 \xrightarrow{*} i \mid\mid j \rhd J_1' \xrightarrow{*} i'}$$

Règle xviii : $J_1 ::= I \mid\mid\mid J_2$

Dans les règles (xviii3) et (xviii5), il suffit que l'un des non terminaux I ou J (respectivement) s'annihile pour que la structure compositionnelle s'annihile elle aussi, ce qui constitue la différence avec la composition forte, où l'annihilation du second non terminal est requise pour l'annihilation de toute la structure.

$$\text{(xviii.1)}\quad \frac{\{\mathtt{e}\} \vdash I \xrightarrow{*} i \rhd I' \xrightarrow{*} i' \wedge J_2 \xrightarrow{*} j \rhd J_2' \xrightarrow{*} j'}{\{\mathtt{e}\} \vdash J_1 \xrightarrow{*} i \mid\mid\mid j \rhd J_1' \xrightarrow{*} i' \mid\mid\mid j'}$$

$$\text{(xviii.2)}\quad \frac{\{\mathtt{e}\} \vdash I \xrightarrow{*} i \rhd C \xrightarrow{*} c \wedge J_2 \xrightarrow{*} j \rhd J_2' \xrightarrow{*} j'}{\{\mathtt{e}\} \vdash J_1 \xrightarrow{*} i \mid\mid\mid j \rhd C' \xrightarrow{*} c \mid\mid\mid \mathtt{when}\ j'}$$

$$\text{(xviii.3)}\quad \frac{\{\mathtt{e}\} \vdash I \xrightarrow{*} i \rhd I' \xrightarrow{*} \mathtt{nil} \wedge J_2 \xrightarrow{*} j \rhd J_2' \xrightarrow{*} j'}{\{\mathtt{e}\} \vdash J_1 \xrightarrow{*} i \mid\mid\mid j \rhd J_1' \xrightarrow{*} \mathtt{nil}}$$

$$\text{(xviii.4)}\quad \frac{\{\mathtt{e}\} \vdash I \xrightarrow{*} i \rhd I' \xrightarrow{*} i' \wedge J_2 \xrightarrow{*} j \rhd C \xrightarrow{*} c}{\{\mathtt{e}\} \vdash J_1 \xrightarrow{*} i \mid\mid\mid j \rhd C' \xrightarrow{*} \mathtt{when}\ i' \mid\mid\mid c}$$

$$\text{(xviii.5)}\quad \frac{\{\mathtt{e}\} \vdash I \xrightarrow{*} i \rhd I' \xrightarrow{*} i' \wedge J_2 \xrightarrow{*} j \rhd J_2' \xrightarrow{*} \mathtt{nil}}{\{\mathtt{e}\} \vdash J_1 \xrightarrow{*} i \mid\mid\mid j \rhd J_1' \xrightarrow{*} \mathtt{nil}}$$

Règle xix : $E ::= id$

(xix.1) $\{\mathtt{e}\} \vdash E \rightarrow \mathtt{e}' \rhd E' \rightarrow \mathtt{nil}$ si $\mathtt{e} = \mathtt{e}'$

(xix.2) $\{\mathtt{e}\} \vdash E \rightarrow \mathtt{e}' \rhd E' \rightarrow \mathtt{e}'$ si $\mathtt{e} \neq \mathtt{e}'$

Règle xx : $M ::= id$

(xx.1) $\{\mathtt{e}\} \vdash M \rightarrow \mathtt{A} \rhd M \rightarrow \mathtt{nil}$ si $\mathtt{e} = \mathtt{fin}_{\mathtt{A}}$

(xx.2) $\{\mathtt{e}\} \vdash M \rightarrow \mathtt{A} \rhd M \rightarrow \mathtt{A}$ si $\mathtt{e} \neq \mathtt{fin}_{\mathtt{A}}$

Les deux dernières règles déclenchent l'effet des occurrences d'événement sur les programmes Electre en annihilant les événements qui surviennent ou les modules dont l'exécution s'achève, ce qui se répend par la suite dans tout le programme par l'intermédiaire des autres règles.

Annexe B

Résultats pour les Lossy machines à compteurs

Nous apportons ici la preuve de l'indécidabilité des problèmes de model-checking de LTL et de la bornitude pour les Lossy machines à 3 compteurs. Notons que des résultats équivalents ont été établis dans [77] pour les machines qui possèdent au moins 4 compteurs. Les preuves présentées dans cette annexe sont fortement basées sur la notion de «capacité» due à [29].

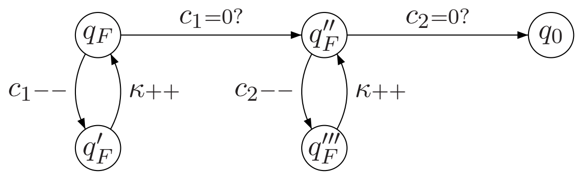
B.1 Indécidabilité du model-checking de LTL pour les Lossy MC(3)

Lemme B.1
Le problème du model-checking de LTL (MC-LTL) est indécidable pour les Lossy machines à 3 compteurs. ♦

Preuve (due à Grégoire Sutre). Considérons une machine à deux compteurs déterministes M. Si c_1 et c_2 sont les deux compteurs de M, et κ est le troisième compteur de $\overset{\rightsquigarrow}{M}$ (la "capacité" de $\overset{\rightsquigarrow}{M}$), la Lossy machine à compteurs obtenue depuis M par :

- À toute instruction $q \xrightarrow{c_k++} q'$ de M correspondent les instructions : $q \xrightarrow{\kappa=0?}$ *échec*, $q \xrightarrow{\kappa--} q_1$ et $q_1 \xrightarrow{c_k++} q'$ dans $\overset{\rightsquigarrow}{M}$, où q_1 est un nouvel état de $\overset{\rightsquigarrow}{M}$ et *échec* est le nouvel état final de $\overset{\rightsquigarrow}{M}$,
- Tout couple d'instructions $q \xrightarrow{c_k=0?} q'$ et $q \xrightarrow{c_k--} q''$ de M est transcris par les instructions : $q \xrightarrow{c_k=0?} q'$, $q \xrightarrow{c_k--} q_1$ et $q_1 \xrightarrow{\kappa++} q''$ dans $\overset{\rightsquigarrow}{M}$, où q_1 est un nouvel état de $\overset{\rightsquigarrow}{M}$.
- L'état initial de $\overset{\rightsquigarrow}{M}$ est q_0', avec les instructions : $q_0' \xrightarrow{\kappa++} q_0'$ et[1] $q_0' \xrightarrow{c_1=0?} q_0$,
- Enfin, nous ajoutons à $\overset{\rightsquigarrow}{M}$ la séquence de "réinitialisation" suivante :

[1] La dernière instruction $q_0' \xrightarrow{c_1=0?} q_0$ est toujours franchissable. Ainsi, lors de son exécution, $\overset{\rightsquigarrow}{M}$ peut atteindre l'état q_0 dans une configuration où $c_1 = c_2 = 0$, mais la valeur de κ est quelconque dans $\mathbb{N}$.



qui remet à zéro les compteurs c_1 et c_2 tout en ajoutant leurs valeurs en q_F au compteur κ, puis qui replace $\overset{\leadsto}{M}$ en q_0 pour poursuivre son exécution.

Ainsi, dans toute configuration accessible de $\overset{\leadsto}{M}$, $c_1 + c_2 + \kappa \leq n$, où n est la valeur de κ lors du passage de q_0' à q_0. De plus, la valeur de $c_1 + c_2 + \kappa$ ne croît jamais[2] lors de l'exécution de $\overset{\leadsto}{M}$. Nous montrons que M s'arrête ssi $\overset{\leadsto}{M}$ visite q_0 infiniment souvent.

Si M s'arrête, alors $\exists m_1, m_2$ tels que $\langle q_F, m_1, m_2 \rangle$ est accessible dans M. Donc pour n assez grand[3], il existe une exécution de $\overset{\leadsto}{M}$ qui visite q_0 infiniment souvent : $\overset{\leadsto}{M}$ exécute n fois l'instruction $q_0' \xrightarrow{\kappa++} q_0'$ puis l'instruction $q_0' \xrightarrow{c_1=0?} q_0$ atteignant ainsi la configuration $\langle q_0, 0, 0, n \rangle$. Ensuite, $\overset{\leadsto}{M}$ exécute la même séquence d'instructions (donc sans perte) que M, atteignant ainsi $\langle q_F, m_1, m_2, n' \rangle$ avec $n' + m_1 + m_2 = n$. Puis, l'exécution de la séquence de "réinitialisation" ramène $\overset{\leadsto}{M}$ en $\langle q_0, 0, 0, n \rangle$ où elle peut itérer infiniment la même séquence d'instructions, et donc visiter infiniment souvent q_0.

Si $\overset{\leadsto}{M}$ visite infiniment souvent q_0, alors M s'arrête. En effet, la perte ne peut se produire qu'un nombre fini de fois dans $\overset{\leadsto}{M}$ puisqu'elle fait décroître la valeur de $c_1 + c_2 + \kappa$ qui ne croît jamais lors de l'exécution de $\overset{\leadsto}{M}$. Ainsi, après avoir exécuté un nombre fini d'instructions, $\overset{\leadsto}{M}$ s'exécute sans aucune perte, et puisqu'elle visite infiniment souvent q_0 (et par la défінion du chemin de q_F à q_0), il existe un chemin sans perte de q_0 à q_F.

Finalement, $\overset{\leadsto}{M}$ visite infiniment souvent q_0 si et seulement si elle satisfait la formule LTL : $\mathsf{GF}\, q_0$. D'où l'indécidabilité du model-checking de LTL pour une Lossy machine à 3 compteurs.

♦

B.2 Indécidabilité du problème de la bornitude pour les Lossy MC(3)

Notons $|\langle q, m_1, \ldots, m_n \rangle| = m_1 + \cdots + m_n$ l'espace utilisé par la configuration $\langle q, m_1, \ldots, m_n \rangle$. Le problème de la bornitude (B) pour une Lossy machine à compteurs $\overset{\leadsto}{M}$ consiste à déterminer s'il existe $b \in \mathbb{N}$ tel que pour toute configuration accessible $\langle q, m_1, \ldots, m_n \rangle$ de $\overset{\leadsto}{M}$:

$$|\langle q, m_1, \ldots, m_n \rangle| \leq b$$

Lemme B.2

Le problème (B) est indécidable pour les Lossy machines à 3 compteurs déterministes. ♦

[2]La valeur de $c_1 + c_2 + \kappa$ peut être augmentée de 1 si elle vient juste d'être diminuée de 1, lors de l'exécution des instructions correspondant à une décrémentation ($--$) de M.

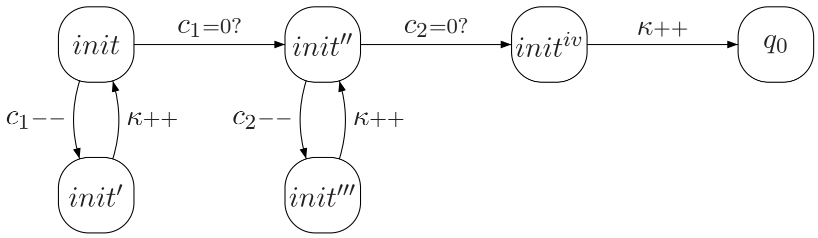
[3]Un tel n existe puisque M s'arrête : les compteurs c_1 et c_2 initialement nuls ne peuvent avoir de valeurs arbitrairement grandes.

Preuve. Cette preuve se base sur la construction de [29] de l'indécidabilité de (B) pour les réseaux de Petri avec reset, basée sur la notion de "capacité".

Nous considérons une machine à compteurs déterministe M, et nous construisons une Lossy machine à compteurs déterministe $\overset{\rightsquigarrow}{M}$ telle que l'une est bornée si et seulement si l'autre l'est. Supposons que c_1 et c_2 sont les deux compteurs de M, et soit κ le troisième compteur de $\overset{\rightsquigarrow}{M}$. Ce dernier compteur représente la "capacité de perte" de $\overset{\rightsquigarrow}{M}$.

– Toute instruction d'incrémentation de c_1 ou c_2 dans M est gardée par κ dans $\overset{\rightsquigarrow}{M}$. Ainsi, à chaque instruction $q \xrightarrow{c_k{+}{+}} q'$ de M correspondent les instructions $q \xrightarrow{\kappa=0?} init$, $q \xrightarrow{\kappa{-}{-}} q_1$ et $q_1 \xrightarrow{c_k{+}{+}} q'$ dans $\overset{\rightsquigarrow}{M}$,
– À chaque couple d'instructions $q \xrightarrow{c_k=0?} q'$ et $q \xrightarrow{c_k{-}{-}} q''$ de M nous associons les instructions $q \xrightarrow{c_k=0?} q'$, $q \xrightarrow{c_k{-}{-}} q_1$ et $q_1 \xrightarrow{\kappa{+}{+}} q''$,

où les états notés q_1 sont de nouveaux états de $\overset{\rightsquigarrow}{M}$ et $init$ est un nouvel état de $\overset{\rightsquigarrow}{M}$. Finalement, à chaque fois que l'état $init$ est atteint (la valeur de κ ne permet pas l'incrémentation d'un compteur), la séquence de "réinitialisation" suivante est exécutée :



Les configurations initiales de M et $\overset{\rightsquigarrow}{M}$ sont respectivement $\langle q_0, 0, 0\rangle$ et $\langle q_0, 0, 0, 0\rangle$. Dans $\overset{\rightsquigarrow}{M}$, la valeur de la somme $c_1+c_2+\kappa$ ne peut croître que si la séquence de "réinitialisation" est franchie.

Si M n'est pas bornée, alors la somme c_1+c_2 n'est pas bornée. Dans $\overset{\rightsquigarrow}{M}$, la valeur de la somme $c_1 + c_2 + \kappa$ ne peut pas excéder la valeur de κ suite à la dernière exécution de la séquence de "réinitialisation". Lorsque $\overset{\rightsquigarrow}{M}$ exécute (sans perte) la même séquence d'instructions que M, après l'exécution d'un nombre fini d'instructions, le franchissement d'une instruction d'incrémentation $c_1{+}{+}$ ou $c_2{+}{+}$ devient impossible car la valeur de κ est nulle. $\overset{\rightsquigarrow}{M}$ exécute donc la séquence de réinitialisation et atteint la configuration $\langle q_0, 0, 0, m'_\kappa\rangle$ avec[4] $m'_\kappa = m_\kappa + 1$, m_κ étant la valeur de κ après l'exécution précédente de la séquence de réinitialisation. $\overset{\rightsquigarrow}{M}$ peut donc à nouveau exécuter la séquence d'instructions exécutée par M depuis sa configuration initiale $\langle q_0, 0, 0\rangle$, un peu plus loin, jusqu'à ce qu'à nouveau la valeur de κ soit nulle, nécessitant à nouveau l'exécution de la séquence de "réinitialisation", etc. Nous en déduisons qu'il existe une exécution de $\overset{\rightsquigarrow}{M}$ qui admet des configurations telles que la somme $c_1 + c_2 + \kappa$ est arbitrairement grande. Par conséquent, $\overset{\rightsquigarrow}{M}$ n'est pas bornée.

Si $\overset{\rightsquigarrow}{M}$ n'est pas bornée, alors il existe une exécution sur laquelle l'instruction $init^{iv} \xrightarrow{\kappa{+}{+}} q_0$ est franchie infiniment souvent puisque c'est la seule façon de faire croître la somme $c_1 + c_2 + \kappa$. Soit :

$$\ldots \xrightarrow{\kappa{+}{+}} \langle q_0, 0, 0, m_\kappa\rangle \ldots \xrightarrow{\kappa{+}{+}} \langle q_0, 0, 0, m'_\kappa\rangle \ldots \xrightarrow{\kappa{+}{+}} \langle q_0, 0, 0, m''_\kappa\rangle \ldots$$

avec $m'_\kappa < m_\kappa < m''_\kappa$, une partie d'une telle exécution. Puisque $m'_\kappa < m_\kappa$, il est survenu (au moins) une transition de perte entre les configurations $\langle q_0, 0, 0, m_\kappa\rangle$ et $\langle q_0, 0, 0, m'_\kappa\rangle$. De plus,

[4]Rappelons que l'on considère une exécution sans perte de $\overset{\rightsquigarrow}{M}$.

le passage dans la séquence de "réinitialisation" augmente la valeur de κ d'une unité (au plus, lorsqu'aucune perte ne se produit). Par conséquent, $\langle q_0, 0, 0, m_k\rangle$ apparaît sur le chemin de $\langle q_0, 0, 0, m'_\kappa\rangle$ à $\langle q_0, 0, 0, m''_\kappa\rangle$:

$$\ldots \xrightarrow{\kappa{+}{+}} \langle q_0, 0, 0, m_\kappa\rangle \ldots \xrightarrow{\kappa{+}{+}} \langle q_0, 0, 0, m'_\kappa\rangle \ldots \xrightarrow{\kappa{+}{+}} \langle q_0, 0, 0, m_\kappa\rangle \ldots \xrightarrow{\kappa{+}{+}} \langle q_0, 0, 0, m''_\kappa\rangle \ldots$$

et il existe donc une exécution de $\overset{\leadsto}{M}$ qui ignore le chemin séparant les deux occurrences de $\langle q_0, 0, 0, m_\kappa\rangle$:

$$\ldots \xrightarrow{\kappa{+}{+}} \langle q_0, 0, 0, m_\kappa\rangle \ldots \xrightarrow{\kappa{+}{+}} \langle q_0, 0, 0, m''_\kappa\rangle \ldots$$

Puisque toute transition de perte (sur c_1, c_2 ou κ) diminue la valeur de la somme $c_1 + c_2 + \kappa$, nous en déduisons que $\overset{\leadsto}{M}$ admet une exécution sans perte telle que la somme $c_1 + c_2 + \kappa$ (donc la somme $c_1 + c_2$) n'est pas bornée. Cette exécution est de la forme :

$$\langle q_0, 0, 0, 0\rangle \xrightarrow{\sigma_0}{}^{*} \langle q_0, 0, 0, 1\rangle \xrightarrow{\sigma_1}{}^{*} \langle q_0, 0, 0, 2\rangle \ldots$$

et les séquences $\sigma_0, \sigma_1, \ldots$ sont les préfixes d'une exécution non bornée de M.

Puisque (B) n'est pas décidable pour les machines à 2 compteurs, nous en déduisons qu'il n'est pas non plus décidable pour les Lossy machine à 3 compteurs déterministes. ♦

Annexe C

Preuve de la simulation des Lossy MC par les Lossy SFR Embarqués

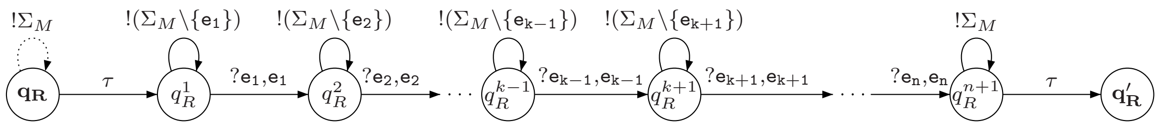
Nous prouvons le théorème 4.26 (page 105), c'est à dire, la simulation des Lossy machines à n compteurs déterministes par les Lossy SFR Embarqués avec n événements mémorisables et un environnement périodique. Pour cela, nous reprenons la démarche des sections 3.1.2 (page 72) et 3.1.2 (page 75) pour la preuve de la simulation des machines à 2 compteurs déterministes par les SFR Embarqués avec 2 événements mémorisables et un environnement périodique. Cependant, la simulation des machines à n compteurs par les machines à 2 compteurs [80] n'est plus valide dans le cadre d'une sémantique de perte. Nous adaptons donc notre construction en considérant :

- une Lossy machine à n compteurs déterministes au lieu d'une machine à 2 compteurs déterministes.
- un Lossy SFR Embarqué avec n événements mémorisables, à la place d'un SFR Embarqué avec 2 événements mémorisables.

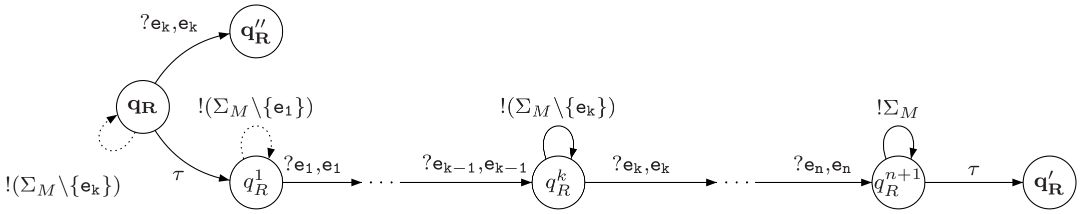
Nous nous intéressons donc à une Lossy machine à compteurs déterministe $\overset{\leadsto}{M}= (Q, C, I)$ ayant un nombre $n \geq 1$ quelconque de compteurs : $C = \{c_1, \ldots, c_n\}$. Les structures présentées en figure 3.1 (page 73) pour les machines à 2 compteurs, sont étendues à n compteurs, et donc n événements mémorisables $e_1, \ldots, e_n$. Elles sont représentées en figure C.1 (page 206).

Les transitions de mémorisation sont étiquetées par des alphabets : $!\Sigma$ représente alors l'ensemble des transitions $!\mathtt{e}$ de mémorisation de $\mathtt{e}$ pour l'ensemble des événements $\mathtt{e} \in \Sigma$. De la même façon, dans la structure de terminaison de la figure C.1(c), les transitions étiquetées $?\Sigma_M$ et Σ_M représentent les transitions $?\mathtt{e}$ et $\mathtt{e}$ pour tout événement $\mathtt{e} \in \Sigma_M$. Dans la structure de la figure C.1(a), nous notons particulièrement la mémorisation de $\mathtt{e_k}$ qui est forcée par l'abscence de transition de prise en compte de cet événement. Finalement, l'environnement correspondant est lui-aussi étendu à n événements mémorisables. Il est représenté en figure C.2 (page 206). Par analogie au modèle simulé, l'AFR obtenu en suivant cette construction est appelé *AFR à compteurs*.

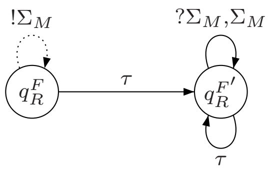
Définition C.1 (AFR à n compteurs)
Soit $\overset{\leadsto}{M}= (Q, C, I)$ une Lossy machine à n compteurs déterministe avec $C = \{c_1, \ldots, c_n\}$. L'*automate à file réactif à n compteurs (AFR à n compteurs)* $R = (Q_R, q_R^0, A_R, \rightarrow_R)$ associé à $\overset{\leadsto}{M}$ est



(a) Simulation de l'instruction : $q \xrightarrow{c_k++} q'$.



(b) Simulation des instructions : $q \xrightarrow{c_k=0?} q'$ et $q \xrightarrow{c_k--} q''$.



(c) Structure de terminaison.

FIG. C.1 – Structures d'AFR pour la simulation d'une Lossy machine à n compteurs déterministe.

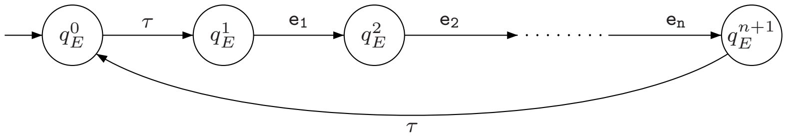


FIG. C.2 – Environnement périodique pour la simulation d'une Lossy machine à n compteurs déterministe.

défini par :

1. L'ensemble des états Q_R est calculé depuis les instructions de $\overset{\leadsto}{M}$ par :

$$
\begin{aligned}
Q_R = \bigcup_{q \in Q} \Bigg[& \left\{ q_R^1, \ldots, q_R^{k-1}, q_R^{k+1}, \ldots, q_R^{n+1} \mid \exists c_k \in C, q' \in Q \text{ t.q. } I_q = \{ q \xrightarrow{c_k{+}{+}} q' \} \right\} \\
& \cup \left\{ q_R^1, \ldots, q_R^{n+1} \mid \exists c_k \in C \text{ et } q', q'' \in Q \text{ t.q. } I_q = \{ q \xrightarrow{c_k=0?} q', q \xrightarrow{c_k{-}{-}} q'' \} \right\} \Bigg] \\
& \cup \{ q_R \mid q \in Q \} \cup \{ q_R^{F'} \}
\end{aligned}
$$

2. L'état initial de R, q_R^0, correspond à q_0, l'état initial de $\overset{\leadsto}{M}$,
3. L'ensemble des actions de R est défini conformément à la définition 2.5 (page 42) en considérant : $\Sigma_M = \{e_1, \ldots, e_n\}$ et $\Sigma_F = \{\tau\}$. Donc $A_R = \{\tau, e_1, \ldots, e_n\} \cup (\{!, ?\} \times \{e_1, \ldots, e_n\})$,
4. Finalement, la relation de transition $\rightarrow_R$ correspond à la description des structures en figure C.1 :

$$
\begin{aligned}
& \bigcup_{\left(q \xrightarrow{c_k{+}{+}} q'\right) \in I} \Bigg[\left\{ q_R \xrightarrow{\tau}_R q_R^1 \right\} \cup \bigcup_{\mathsf{e} \in \Sigma_M} \left\{ q_R \xrightarrow{!\mathsf{e}}_R q_R \right\} \cup \bigcup_{j \in (\{1,\ldots,n\} \setminus \{k\})} \left(\bigcup_{\mathsf{e} \in (\Sigma_M \setminus \{\mathsf{e_j}\})} \left\{ q_R^j \xrightarrow{!\mathsf{e}}_R q_R^j \right\} \right) \\
& \quad \cup \bigcup_{j \in \{1,\ldots,k-2,k+1,\ldots,n\}} \left\{ q_R^j \xrightarrow{?\mathsf{e_j},\, \mathsf{e_j}}_R q_R^{j+1} \right\} \cup \left\{ q_R^{k-1} \xrightarrow{?\mathsf{e_{k-1}},\, \mathsf{e_{k-1}}}_R q_R^{k+1} \right\} \\
& \quad \cup \left\{ q_R^{n+1} \xrightarrow{\tau}_R q_R' \right\} \cup \bigcup_{\mathsf{e} \in \Sigma_M} \left\{ q_R^{n+1} \xrightarrow{!\mathsf{e}}_R q_R^{n+1} \right\} \Bigg] \\
& \cup \bigcup_{\left\{ q \xrightarrow{c_k=0?} q',\ q \xrightarrow{c_k{-}{-}} q'' \right\} \subseteq I} \Bigg[\left\{ q_R \xrightarrow{\tau}_R q_R^1, q_R \xrightarrow{?\mathsf{e_k},\, \mathsf{e_k}}_R q_R'' \right\} \cup \bigcup_{\mathsf{e} \in (\Sigma_M \setminus \{\mathsf{e_k}\})} \left\{ q_R \xrightarrow{!\mathsf{e}}_R q_R \right\} \\
& \quad \cup \bigcup_{j \in \{1,\ldots,n\}} \left(\left\{ q_R^j \xrightarrow{?\mathsf{e_j},\, \mathsf{e_j}}_R q_R^{j+1} \right\} \cup \bigcup_{\mathsf{e} \in \left(\Sigma_M \setminus \{\mathsf{e_j}\}\right)} \left\{ q_R^j \xrightarrow{!\mathsf{e}}_R q_R^j \right\} \right) \\
& \quad \cup \left\{ q_R^{n+1} \xrightarrow{\tau}_R q_R' \right\} \cup \bigcup_{\mathsf{e} \in \Sigma_M} \left\{ q_R^{n+1} \xrightarrow{!\mathsf{e}}_R q_R^{n+1} \right\} \Bigg] \\
& \cup \Bigg[\left\{ q_R^F \xrightarrow{\tau}_R q_R^{F'} \right\} \cup \bigcup_{\mathsf{e} \in \Sigma_M} \left\{ q_R^F \xrightarrow{!e}_R q_R^F, q_R^{F'} \xrightarrow{?\mathsf{e},\, \mathsf{e}}_R q_R^{F'} \right\} \cup \left\{ q_R^{F'} \xrightarrow{\tau}_R q_R^{F'} \right\} \Bigg]
\end{aligned}
$$

♦

Dans la section C.1, nous étudions les propriétés structurelles de AFR à n compteurs. Puis, dans la section C.2 (page 212), nous utilisons ces propriétés pour démontrer la simulation des Lossy machines à n compteurs déterministes par les Lossy SFR Embarqués avec n événements mémorisables.

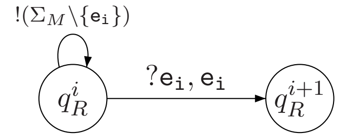
C.1 Analyse de la structure des AFR à n compteurs

Nous étudions, dans cette première section, le comportement des structures de simulation représentées en figure C.1 (page 206). Pour cela, nous nous intéressons successivement aux structures élémentaires qui les composent. Nous caractérisons l'ensemble des configurations accessibles par le franchissement de ces structures élémentaires : les $\mathtt{e_i}$-*motifs* (voir ci-dessous), les *séquences ordonnées de* $\mathtt{e_i}$-*motifs* (page 209) et finalement, les *séquences ordonnées et complètes de* $\mathtt{e_i}$-*motifs* (page 212).

C.1.1 Les $\mathtt{e_i}$-motifs

Définition C.2

Un $\mathtt{e_i}$-*motif*, avec $\mathtt{e_i} \in \Sigma_M = \{\mathtt{e_1}, \dots, \mathtt{e_n}\}$, est une structure d'AFR de la forme :



♦

Les structures de simulation de la figure C.1 (page 206) sont construites à partir de séquences de $\mathtt{e_i}$-motifs. Afin de prouver la simulation, nous étudions la transformation définie par un $\mathtt{e_i}$-motif dans le lemme C.3, puis, dans le lemme C.6, l'effet cumulé d'une séquence de $\mathtt{e_i}$-motifs.

Dans la suite, nous considérons l'environnement[1] E de la figure C.3. Un $\mathtt{e_i}$-motif plongé dans

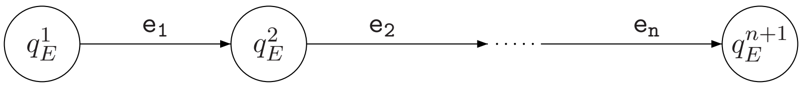


FIG. C.3 – Environnement E.

son environnement est appelé $\mathtt{e_i}$-*motif embarqué*.

Lemme C.3

Considérons un $\mathtt{e_i}$-motif embarqué. Depuis $\langle q_R^i, w, q_E^{i-r} \rangle$ avec $0 \leq r \leq i-1$, $S||E$ atteint la configuration :

1. $\langle q_R^{i+1}, \mathtt{e_i}^{\ominus 1} w, q_E^{i-r} \rangle$ si $(\Psi(w))_i > 0$,
2. $\langle q_R^{i+1}, w\mathtt{e_{i-r}} \dots \mathtt{e_{i-1}}, q_E^{i+1} \rangle$ sinon.

♦

Preuve. Supposons que $(\Psi(w))_i > 0$. Alors, la priorité etant donnée au traitement des occurrences mémorisées, la transition $q_R^i \xrightarrow{?\mathtt{e_i}}_R q_R^{i+1}$ est franchie, et par conséquent, $S||E$ atteint la configuration : $\langle q_R^{i+1}, \mathtt{e_i}^{\ominus 1} w, q_E^{i-r} \rangle$.

À l'inverse, lorsque $(\Psi(w))_i = 0$, la transition franchie est déterminée par l'événement fourni par l'environnement. Ainsi, les événements $\mathtt{e_{i-r}}, \dots, \mathtt{e_{i-1}}$ sont mémorisés lors de l'exécution de la séquence :

$$\langle q_R^i, w, q_E^{i-r} \rangle \xrightarrow{!\mathtt{e_{i-r}}}_{S||E} \langle q_R^i, w\mathtt{e_{i-r}}, q_E^{i-r+1} \rangle \dots \xrightarrow{!\mathtt{e_{i-1}}}_{S||E} \langle q_R^i, w\mathtt{e_{i-r}} \dots \mathtt{e_{i-1}}, q_E^i \rangle$$

[1]Nous l'obtenons depuis celui de la figure C.2 (page 206), en nous restreignant au chemin de q_E^1 à q_E^{n+1}

Puis, l'environnement propose l'événement $\mathtt{e_i}$, contraignant le système à franchir la transition $q_R^i \xrightarrow{\mathtt{e_i}}_R q_R^{i+1}$. Il atteint ainsi la configuration : $\langle q_R^{i+1}, w\mathtt{e_{i-r}} \dots \mathtt{e_{i-1}}, q_E^{i+1} \rangle$. ♦

Remarque C.4

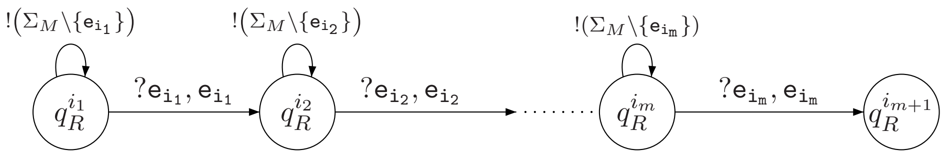
Dans le lemme C.3 nous considérons un décalage entre E et S exprimé par la relation $0 \leq r \leq i-1$. Il peut être vu comme un "retard" (d'où l'indice r) de E sur S provoqué par l'immobilisme de l'environnement lorsque le système franchit une transition de prise en compte différée, comme par exemple dans le premier cas du lemme C.3. De la même façon, dans le lemme C.6, la condition $1 \leq r \leq i_m$ exprime l'éventuel "retard" de l'environnement.

Il est bien sûr impossible que E soit en "avance" sur S puisqu'à chaque fois que E franchit une transition, S fait de même. ♦

C.1.2 Les séquences ordonnées de $\mathtt{e_i}$-motifs

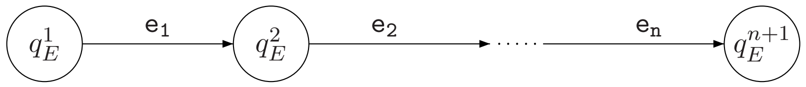
Définition C.5

Une *séquence ordonnée de* $\mathtt{e_{i_1}}, \dots, \mathtt{e_{i_m}}$*-motifs* est la structure d'AFR de la forme :



avec $m > 0$ et $1 < i_1 < i_2 < \dots < i_m < n$. ♦

Nous nous interessons maintenant à une séquence ordonnée de $\mathtt{e_{i_1}}, \dots, \mathtt{e_{i_m}}$-motifs plongée dans son environnement :



qui est donc appelée *séquence ordonnée et embarquée de* $\mathtt{e_{i_1}}, \dots, \mathtt{e_{i_m}}$*-motifs*.

Lemme C.6

Depuis la configuration $\langle q_R^{i_1}, w, q_E^1 \rangle$, l'exécution d'une séquence ordonnée et embarquée de $\mathtt{e_{i_1}}, \dots, \mathtt{e_{i_m}}$-motifs conduit à la configuration $\langle q_R^{i_m+1}, w', q_E^{i_m+1-r} \rangle$ avec $0 \leq r \leq i_m$ vérifiant :

1. $\forall k \in \{i_1, \dots, i_j\},\ (\Psi(w'))_k = (\Psi(w))_k$,
2. $\forall k \in \{i_{j+1}, \dots, i_m\},\ (\Psi(w'))_k = (\Psi(w))_k - 1$,
3. $\forall k \in (\{1, \dots, i_m - r\} \setminus \{i_1, \dots, i_m\}),\ (\Psi(w'))_k = (\Psi(w))_k + 1$,
4. $\forall k \in (\{i_m - r + 1, \dots, n\} \setminus \{i_1, \dots, i_m\}),\ (\Psi(w'))_k = (\Psi(w))_k$.

où $j = \max\{l \in \{1, \dots, m\} \mid i_l \leq i_m - r\}$ si $i_1 \leq i_m - r$ et $j = 0$ sinon. ♦

Dans le lemme C.6, l'entier j identifie le dernier événement de la séquence dont une occurrence a été émise par E et dont une occurrence a été traitée par le système. Les propriétés (1) à (4) expriment que :

1. pour tous les événements e qui ont été pris en compte par S et émis par E jusqu'à $\langle q_R^{i_{m+1}}, w', q_E^{i_m+1-r}\rangle$, le nombre d'occurrences mémorisées de e est le même dans w' et w. Nous montrons que soit l'occurrence émise par E a été prise en compte immédiatement, soit le système a tout d'abord consommé une occurrence de e puis il a mémorisé celle qui lui est proposée par E.
2. tous les événements qui ont été consommés par le système et qui n'ont pas été proposés depuis par E comptent une occurrence mémorisée de moins en w' par rapport à w.
3. un événement qui a été émis par l'environnement et qui n'a pas encore été pris en compte par le système est mémorisé et ne sera jamais pris en compte.
4. pour tout événement qui n'a pas encore été émis et qui n'a pas été consommé, le nombre d'occurrences mémorisées dans w' est le même que dans w.

Remarque C.7
Notons que les propriétés (1) à (4) qualifient l'évolution des occurrences mémorisées de l'ensemble des événements le long de la séquence puisque $\{i_1, \ldots, i_j\} \cup \{i_{j+1}, \ldots, \imath_m\} \cup \big(\{1, \ldots, i_m - r\} \setminus \{i_1, \ldots, i_m\}\big) \cup \big(\{i_m - r + 1, \ldots, n\} \setminus \{i_1, \ldots, i_m\}\big) = \{1, \ldots, n\}$. ♦

Remarque C.8
L'événement de rang $i_m - r$ est nécessairement l'un de ceux de rangs $i_1, \ldots, i_m$ (ou alors $r = i_m$ auquel cas aucune occurrence d'événement n'a été prise en compte depuis l'environnement). En effet, si ce n'est pas le cas, cela signifie que, lors de l'exécution de la séquence, cet événement a été mémorisé dans une configuration instable puisque plus aucune occurrence d'événement n'a été fournie par l'environnement ensuite. ♦

Preuve (Lemme C.6). Nous allons prouver ce lemme par induction sur la longueur de la séquence considérée, c'est à dire le nombre de $\mathtt{e}_{\mathtt{i_j}}$-motifs qu'elle contient. Soit $p \geq 1$, nous prouvons maintenant que depuis la configuration $\langle q_R^{i_1}, w, q_E^1\rangle$ après le franchissement de p motifs, le système atteint la configuration $\langle q_R^{i_{1+p}}, w', q_E^{i_p+1-r}\rangle$ avec $0 \leq r \leq i_p$ vérifiant :

1. $\forall k \in \{i_1, \ldots, i_j\},\ (\Psi(w'))_k = (\Psi(w))_k$,
2. $\forall k \in \{i_{j+1}, \ldots, i_p\},\ (\Psi(w'))_k = (\Psi(w))_k - 1$,
3. $\forall k \in (\{1, \ldots, i_p - r\} \setminus \{i_1, \ldots, i_p\}),\ (\Psi(w'))_k = (\Psi(w))_k + 1$,
4. $\forall k \in (\{i_p - r + 1, \ldots, n\} \setminus \{i_1, \ldots, i_p\}),\ (\Psi(w'))_k = (\Psi(w))_k$.

où $j = \max\{l \in \{1, \ldots, p\} \,|\, i_l \leq i_p - r\}$ si $i_1 \leq i_p - r$ et $j = 0$ sinon.

- **Si $\mathbf{p = 1}$ (un motif franchi).** D'après le lemme C.3, suivant la valeur de $(\Psi(w))_{i_1}$ il y a deux possibilités :
  - Soit $(\Psi(w))_{i_1} > 0$, alors le franchissement de l'$\mathtt{e}_{\mathtt{i_1}}$-motif conduit à la configuration $\langle q_R^{i_2}, \mathtt{e}_{\mathtt{i_1}}{}^{\ominus 1}w, q_E^1\rangle$. Posons $w' = \mathtt{e}_{\mathtt{i_1}}{}^{\ominus 1}w$, nous avons $j = 0$, $r = i_1$ et par définition de ${}^{\ominus 1}$:
    - $(\Psi(w'))_{i_1} = (\Psi(w))_{i_1} - 1$,
    - $\forall k \in (\{1, \ldots, n\} \setminus \{i_1\}),\ (\Psi(w'))_k = (\Psi(w))_k$.

Ces propriétés correspondent respectivement aux points (2) et (4), alors que les points (1) et (3) sont eux aussi vérifés puisqu'ils correspondent à des ensembles vides de valeurs pour k,

– Soit $(\Psi(w))_{i_1} = 0$, conduisant à la configuration $\langle q_R^{i_2}, w\mathtt{e_1}\ldots\mathtt{e_{(i_1-1)}}, q_E^{i_1+1}\rangle$. Posons $w' = w\mathtt{e_1}\ldots\mathtt{e_{(i_1-1)}}$, nous avons : $j = 1$, $r = 0$ et par définition :
  – $(\Psi(w'))_{i_1} = (\Psi(w))_{i_1}$ (transition de prise en compte immédiate de $\mathtt{e_{i_1}}$),
  – $\forall k \in \{1, \ldots, i_1 - 1\},\ (\Psi(w'))_k = (\Psi(w))_k + 1$,
  – $\forall k \in \{i_1 + 1, \ldots, n\},\ (\Psi(w'))_k = (\Psi(w))_k$.

  Nous avons donc dans l'ordre les points (1), (3) et (4), quant au point (2), l'absence de telles valeurs de k suffit à le prouver.

– **Étape d'induction.** Supposons donc que pour un $p > 1$ donné, l'exécution des p premiers motifs de la séquence amène dans une configuration $\langle q_R^{i_{(1+p)}}, w', q_E^{i_p+1-r}\rangle$ vérifiant les conditions (1) à (4). Conformément au lemme C.3, le franchissement du motif suivant conduit dans l'une des deux configurations suivantes, où $\overline{p} = p+1$, $\overline{j}$ et $\overline{r}$ représentent les nouvelles valeurs de p, j et r :
  – Soit $(\Psi(w))_{i_{1+p}} > 0$ et par le lemme C.3, l'exécution aboutit dans la configuration $\langle q_R^{i_{1+(p+1)}}, \mathtt{e_{i_{1+p}}}^{\ominus 1}w', q_E^{i_p+1-r}\rangle$. Posons $w'' = \mathtt{e_{i_{1+p}}}^{\ominus 1}w'$, nous avons donc $\overline{j} = j$, $i_{\overline{p}} - \overline{r} = i_p - r$ et par définition :
    a. $(\Psi(w''))_{i_{1+p}} = (\Psi(w'))_{i_{1+p}} - 1$,
    b. $\forall k \in (\{1, \ldots, n\} \setminus \{i_{1+p}\}),\ (\Psi(w''))_k = (\Psi(w'))_k$.

    Donc :
    1. $\forall k \in \{i_1, \ldots, i_{\overline{j}}\},\ (\Psi(w''))_k = (\Psi(w))_k$ par hypothèse d'induction sur (1) et par (b),
    2. en utilisant l'hypothèse d'induction sur (2) pour $k \in \{i_{\overline{j}}, \ldots, i_p\}$ et sur (4) pour $i_{\overline{p}}$ ainsi que (a) : $\forall k \in \{i_{\overline{j}+1}, \ldots, i_{\overline{p}}\}, (\Psi(w''))_k = (\Psi(w))_k - 1$,
    3. notons que $i_{\overline{p}} = i_{p+1} \notin \{1, \ldots, i_p - r\}$, donc par (b) et par induction sur (3) : $\forall k \in (\{1, \ldots, i_{\overline{p}} - \overline{r}\} \setminus \{i_1, \ldots, i_{\overline{p}}\}),\ (\Psi(w''))_k = (\Psi(w))_k + 1$,
    4. $\forall k \in (\{i_{\overline{p}} - \overline{r} + 1, \ldots, n\} \setminus \{i_1, \ldots, i_{\overline{p}}\}),\ (\Psi(w''))_k = (\Psi(w))_k$ par hypothèse d'induction sur le point (4) et par (b).

  – Soit $(\Psi(w))_{i_{1+p}} = 0$, alors d'après le lemme C.3 après le franchissement du motif, la configuration du système devient : $\langle q_R^{i_{1+(p+1)}}, w'\mathtt{e_{i_p+1-r}}\ldots\mathtt{e_{i_{(1+p)}-1}}, q_E^{i_{(1+p)}+1}\rangle$. Posons $w'' = w'\mathtt{e_{i_{(1+p)}-r}}\ldots\mathtt{e_{i_{(1+p)}-1}}$, nous avons $\overline{j} = 1 + p = \overline{p}$, $\overline{r} = 0$ et :
    a. $\forall k \in \{i_p + 1 - r, \ldots, i_{(1+p)} - 1\},\ (\Psi(w''))_k = (\Psi(w'))_k + 1$,
    b. $\forall k \in (\{1, \ldots, n\} \setminus \{i_p + 1 - r, \ldots, i_{(1+p)} - 1\}),\ (\Psi(w''))_k = (\Psi(w'))_k$.

    Par conséquent :
    1. en utilisant (a) et par hypothèse d'induction sur (1) pour $k \in \{i_1, \ldots, i_j\}$, sur (2) pour $k \in \{i_{j+1}, \ldots, i_p\}$, sur (4) pour $i_{\overline{p}}$ et avec (b) : $\forall k \in \{i_1, \ldots, i_{\overline{p}}\},\ (\Psi(w''))_k = (\Psi(w))_k$,
    3. $\forall k \in (\{1, \ldots, i_{\overline{p}} - \overline{r}\} \setminus \{i_1, \ldots, i_{\overline{p}}\}),\ (\Psi(w''))_k = (\Psi(w))_k + 1$ par hypothèse d'induction sur (3) et en utilisant (a) et $\overline{r} = 0$,

4. $\forall k \in (\{i_{\overline{p}} - \overline{r} + 1, \ldots, n\} \setminus \{i_1, \ldots, i_{\overline{p}}\})$, $(\Psi(w''))_k = (\Psi(w))_k$ par hypothèse d'induction sur le point (4) et car $\overline{r} = 0$.

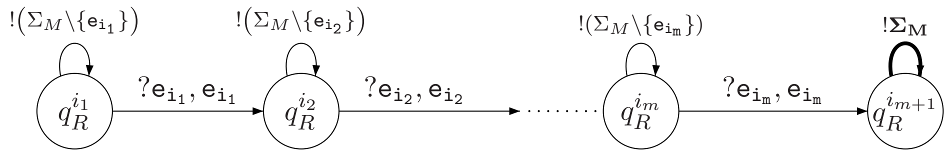
Dans ce dernier cas, le point (2) est trivialement vérifié car l'ensemble des valeurs de k qu'il décrit est vide.

♦

C.1.3 Les séquences ordonnées et complètes de $\mathtt{e_i}$-motifs

Définition C.9

Une séquence ordonnée de $\mathtt{e_{i_1}}, \ldots, \mathtt{e_{i_m}}$-motifs est *complète* si quel que soit $\mathtt{e} \in \Sigma_M$, $q_R^{i_m+1} \xrightarrow{!\mathtt{e}}_R q_R^{i_m+1}$:



♦

Une séquence ordonnée et complète de $\mathtt{e_i}$-motifs est *embarquée* lorsqu'elle est est plongée dans l'environnement E de la figure C.3 (page 208).

Corollaire C.10

Pour une séquence ordonnée complète et embarquée de $\mathtt{e_{i_1}}, \ldots, \mathtt{e_{i_m}}$-motifs, depuis $\langle q_R^{i_1}, w, q_E^1 \rangle$, $S||E$ atteint finalement la configuration $\langle q_R^{i_m+1}, w', q_E^{n+1} \rangle$ et :

$$\forall k \in \{1, \ldots, n\}, \ \big(\Psi(w')\big)_k = \begin{cases} (\Psi(w))_k & \text{si } \exists l \in \{1, \ldots, m\} \text{ t.q. } k = i_l \\ (\Psi(w))_k + 1 & \text{sinon} \end{cases}$$

♦

Preuve. Depuis $\langle q_R^{i_1}, w, q_E^1 \rangle$, $S||E$ atteint la configuration $\langle q_R^{i_m+1}, w', q_E^{i_m+1-r} \rangle$ avec $0 \leq r \leq i_m$ par le lemme C.6. Puisque la séquence est complète, il mémorise alors les occurrences d'événements que l'environnement lui propose, menant ainsi à la configuration $\langle q_R^{i_m+1}, w'', q_E^{n+1} \rangle$ avec $w'' = w' \mathtt{e_{i_m+1-r}} \ldots \mathtt{e_n}$. Alors, puisque :

- $\forall k \in \{i_m + 1 - r, \ldots, n\}$, $(\Psi(w''))_k = (\Psi(w'))_k + 1$,
- $\forall k \in (\{1, \ldots, n\} \setminus \{i_m + 1 - r, \ldots, n\})$, $(\Psi(w''))_k = (\Psi(w'))_k$,

en appliquant le lemme C.6, nous obtenons le résultat. ♦

C.2 Simulation des Lossy MC par les Lossy SFR Embarqués

Nous avons prouvé l'équivalence entre les machines à compteurs et les systèmes à file réactifs embarqués dans le théorème 3.6, page 76. Cependant, l'ajout de la sémantique de perte à ces deux modèles ne préserve à priori pas l'équivalence. En effet, alors que les opérations (++, −− et =0?) d'une machine à compteurs sont atomiques vis à vis des actions de perte, les structures de simulation (voir la figure C.1, page 206 et la définition C.1, page 205), du système à file réactif embarqué équivalent, ne le sont pas. Il faut donc considérer l'effet des actions de perte qui surviennent au beau milieu de l'exécution de ces structures.

Définition C.11

Dans le cadre d'une sémantique de perte, une structure d'AFR est *atomique (vis-à-vis des actions de perte)* si lors de son exécution il n'est pas possible (permis) de franchîr une transition de perte. ♦

Nous montrons maintenant, en nous appuyant sur les résultats de la section C.1, que nous pouvons considérer que les structures de simulation de la figure C.1 (page 206) sont atomiques vis à vis des actions de perte.

C.2.1 Atomicité des $\mathtt{e_i}$-motifs

Considérons à nouveau les $\mathtt{e_i}$-motifs embarqués (définition C.2 page 208), en leur appliquant, cette fois-ci la sémantique de perte de la définition 4.3 (page 90).

Un $\mathtt{e_i}$-motif *atomique* ne permet donc aucune transition de perte lors du passage de q_R^i à q_R^{i+1} : en suivant le lemme C.3 (page 208) depuis une configuration $\langle q_R^i, w, q_E^{i-r}\rangle$ (avec $0 \leq r \leq i-1$), le système atteint la configuration :

- $\langle q_R^{i+1}, \mathtt{e_i}^{\ominus 1}w, q_E^{i-r}\rangle$ si $(\Psi(w))_i > 0$,
- $\langle q_R^{i+1}, w\mathtt{e_{i-r}}\dots\mathtt{e_{i-1}}, q_E^{i+1}\rangle$ sinon.

sans qu'aucune transition de perte ne se produise (ne soit franchissable). L'exécution de ce motif commence donc par une transition de perte, puis le franchissement des transitions du motif, sans aucune perte, puis, lorsque toutes les transitions franchissables ont été exécutées, à nouveau une transition de perte :

$$\langle q_R^i, w, q_E^{i-r}\rangle \xrightarrow{\Upsilon} \langle q_R^i, w', q_E^{i-r}\rangle \to^* \langle q_R^{i+1}, w'', q_E^{k}\rangle \xrightarrow{\Upsilon} \langle q_R^{i+1}, w''', q_E^{k}\rangle$$

où : $k = i - r$ et $w'' = \mathtt{e_i}^{\ominus 1}w'$ si $(\Psi(w'))_i > 0$ et $k = i+1$ et $w'' = w'\mathtt{e_{i-r}}\dots\mathtt{e_{i-1}}$ sinon. Une structure atomique est alors vue comme un ensemble indivisible et suivant la section 4.2, page 90, l'exécution du système est donc composée d'une transition de perte, puis l'exécution de la structure atomique sans perte, puis à nouveau une transition de perte.

Lemme C.12

Considérons un Lossy $\mathtt{e_i}$-motif embarqué M et un Lossy $\mathtt{e_i}$-motif atomique embarqué M_a. Depuis une configuration $\langle q_R^i, w, q_E^{i-r}\rangle$ avec $0 \leq r \leq i-1$, les ensembles de configurations atteintes par M et M_a sont égaux. ♦

Preuve. L'exécution de M et de M_a commence donc par une transition de perte :

$$\langle q_R^i, w, q_E^{i-r}\rangle \xrightarrow{\Upsilon} \langle q_R^i, w', q_E^{i-r}\rangle$$

avec $w' \preccurlyeq w$. Deux cas se présentent alors :

– **Si $(\mathbf{\Psi(w')})_{\mathtt{i}} > 0$.** Conformément au lemme C.3, l'exécution de M_a se poursuit par la consommation de la plus ancienne occurrence mémorisée de $\mathtt{e_i}$, puis, par définition de l'atomicité, par une transition de perte :

$$\langle q_R^i, w', q_E^{i-r} \rangle \xrightarrow{?\mathtt{e_i}} \langle q_R^{i+1}, \mathtt{e_i}^{\ominus 1} w', q_E^{i-r} \rangle \xrightarrow{\curlyvee} \langle q_R^{i+1}, w'', q_E^{i-r} \rangle$$

avec $w'' \preccurlyeq \mathtt{e_i}^{\ominus 1} w'$.
De la même façon, puisque la priorité est donnée au traitement des occurrences mémorisées, M consomme $\mathtt{e_i}$, puis exécute une transition de perte :

$$\langle q_R^i, w', q_E^{i-r} \rangle \xrightarrow{?\mathtt{e_i}} \langle q_R^{i+1}, \mathtt{e_i}^{\ominus 1} w', q_E^{i-r} \rangle \xrightarrow{\curlyvee} \langle q_R^{i+1}, w'', q_E^{i-r} \rangle$$

avec $w'' \preccurlyeq \mathtt{e_i}^{\ominus 1} w'$.
Les deux motifs M et M_a ont bien le même ensemble de configurations accessibles[2] : $\{\langle q_R^{i+1}, w'', q_E^{i-r} \rangle \mid w'' \preccurlyeq \mathtt{e_i}^{\ominus 1} w\}$.

– **Si $(\mathbf{\Psi(w')})_{\mathtt{i}} = 0$.** Soit w ne contient aucune occurrence de $\mathtt{e_i}$, soit elles ont tout été perdues. D'après le lemme C.3, M_a atteint la configuration $\langle q_R^{i+1}, w'\mathtt{e_{i-r}} \dots \mathtt{e_{i-1}}, q_E^{i+1} \rangle$ où la perte d'occurrences mémorisées survient à nouveau, conduisant ainsi aux configurations $\langle q_R^{i+1}, w'', q_E^{i+1} \rangle$ avec $w'' \preccurlyeq w'\mathtt{e_{i-r}} \dots \mathtt{e_{i-1}}$.
Contrainte par l'environnement, l'exécution de M se poursuit par la mémorisation des événements $\mathtt{e_{i-r}}, \dots, \mathtt{e_{i-1}}$ le long de la séquence avec perte :

$$\begin{aligned}\langle q_R^i, w'_{i-r}, q_E^{i-r} \rangle \xrightarrow{!\mathtt{e_{i-r}}} \langle q_R^i, w'_{i-r}\mathtt{e_{i-r}}, q_E^{i-r+1} \rangle \xrightarrow{\curlyvee} \langle q_R^i, w'_{i-r+1}, q_E^{i-r+1} \rangle \dots \\ \langle q_R^i, w'_{i-1}, q_E^{i-1} \rangle \xrightarrow{!\mathtt{e_{i-1}}} \langle q_R^i, w'_{i-1}\mathtt{e_{i-1}}, q_E^{i} \rangle \xrightarrow{\curlyvee} \langle q_R^i, w'_{i}, q_E^{i} \rangle\end{aligned}$$

avec : $w'_{i-r} = w'$ et $\forall k \in \{i-r, \dots, i-1\}$, $w'_{k+1} \preccurlyeq w'_k\mathtt{e_k}$. Par conséquent, $w'_i \preccurlyeq w'\mathtt{e_{i-r}} \dots \mathtt{e_{i-1}}$. Pour terminer, M prend en compte l'occurrence de $\mathtt{e_i}$, puis franchit à nouveau une transition de perte :

$$\langle q_R^i, w'_i, q_E^i \rangle \xrightarrow{\mathtt{e_i}} \langle q_R^{i+1}, w'_i, q_E^{i+1} \rangle \xrightarrow{\curlyvee} \langle q_R^{i+1}, w'', q_E^{i+1} \rangle$$

avec $w'' \preccurlyeq w'_i \preccurlyeq w'\mathtt{e_{i-r}} \dots \mathtt{e_{i-1}}$.
Dans ces deux cas, en utilisant le lemme 1.7 (page 27) nous observons que l'ensemble des configurations accessibles est défini par : $\{\langle q_R^{i+1}, w'', q_E^{i+1} \rangle \mid w'' \preccurlyeq w\mathtt{e_{i-r}} \dots \mathtt{e_{i-1}}\}$.

♦

C.2.2 Atomicité des séquences ordonnées de $\mathtt{e_i}$-motifs

Une séquence ordonnée et embarquée de $\mathtt{e_i}$-motifs (définition C.5, page 209) est *atomique* si aucune perte d'occurrence mémorisée n'est permise lors du franchissement de ses transitions (motifs). Ainsi, son exécution se résume à une transition de perte, puis l'exécution des transitions de la séquence, sans aucune perte, et pour terminer, le franchissement d'une transition

[2]Rappelons que par le lemme 1.7 (page 27), $w' \preccurlyeq w \Rightarrow \mathtt{e_i}^{\ominus 1} w' \preccurlyeq \mathtt{e_i}^{\ominus 1} w$.

de perte. Comme toujours, nous considérons des séquences de $\mathtt{e}_{\mathtt{i}_1}, \ldots, \mathtt{e}_{\mathtt{i}_\mathtt{m}}$-motifs plongées en environnement :

$$q_E^1 \xrightarrow{\mathtt{e}_1} q_E^2 \xrightarrow{\mathtt{e}_2} \cdots\cdots \xrightarrow{\mathtt{e}_\mathtt{n}} q_E^{n+1}$$

donnant naissance aux *séquences atomiques ordonnées et embarquées de* $\mathtt{e}_{\mathtt{i}_1}, \ldots, \mathtt{e}_{\mathtt{i}_\mathtt{m}}$*-motifs*.

Lemme C.13

Soient S_a et S deux Lossy séquences ordonnées et embarquées de $\mathtt{e}_{\mathtt{i}_1}, \ldots, \mathtt{e}_{\mathtt{i}_\mathtt{m}}$-motifs respectivement atomique et non-atomique. Depuis une configuration $\langle q_R^{i_1}, w, q_E^1 \rangle$, les ensembles de configurations atteintes par S et S_a sont égaux. ♦

Preuve. Notons tout d'abord que par le lemme C.12, nous considérons que les motifs de S sont atomiques. Par conséquent, lors de l'exécution de S, les transitions de perte sont possibles uniquement entre les exécutions de chacuns des motifs, c'est à dire lors du passage de l'exécution des transitions de $q_R^{i_j}$ à l'exécution des transitions de $q_R^{i_{j+1}}$.

– $\boxed{S_a}$: depuis la configuration $\langle q_R^{i_1}, w, q_E^1 \rangle$, l'exécution débute par une transition de perte, puis le franchissement de S_a (sans perte) et pour finir, à nouveau une transition de perte :

$$\langle q_R^{i_1}, w, q_E^1 \rangle \xrightarrow{\vec{\tilde{l}}}_{S \widetilde{\Vert} E} \langle q_R^{i_1}, w', q_E^1 \rangle \xrightarrow{S_a}{}^{*}_{S \widetilde{\Vert} E} \langle q_R^{i_{m+1}}, w'', q_E^{i_m+1-r} \rangle \xrightarrow{\vec{\tilde{l}}}_{S \widetilde{\Vert} E} \langle q_R^{i_{m+1}}, w''', q_E^{i_m+1-r} \rangle$$

où la transition d'étiquette S_a désigne symboliquement le franchissement de la séquence, et $o \leq r \leq i_m$. Par définition de $\vec{\tilde{l}}$, nous avons :
– $\forall k \in \{1, \ldots, n\},\ (\Psi(w'))_k \leq (\Psi(w))_k$,
– $\forall k \in \{1, \ldots, n\},\ (\Psi(w'''))_k \leq (\Psi(w''))_k$.
De plus, par le lemme C.6 :

1. $\forall k \in \{i_1, \ldots, i_j\},\ (\Psi(w''))_k = (\Psi(w'))_k$,
2. $\forall k \in \{i_{j+1}, \ldots, i_m\},\ (\Psi(w''))_k = (\Psi(w'))_k - 1$,
3. $\forall k \in (\{1, \ldots, i_m - r\} \setminus \{i_1, \ldots, i_m\}),\ (\Psi(w''))_k = (\Psi(w'))_k + 1$,
4. $\forall k \in (\{i_m - r + 1, \ldots, n\} \setminus \{i_1, \ldots, i_m\}),\ (\Psi(w''))_k = (\Psi(w'))_k$.

où $j = \max\{l \in \{1, \ldots, m\} \mid i_l \leq i_m - r\}$ si $i_1 \leq i_m - r$ et $j = 0$ sinon.
Nous en déduisons donc que depuis la configuration $\langle q_R^{i_1}, w, q_E^1 \rangle$, le franchissement de S_a conduit à l'ensemble des configurations $\langle q_R^{i_{m+1}}, w''', q_E^{i_m+1-r} \rangle$ vérifiant :

1. $\forall k \in \{i_1, \ldots, i_j\},\ (\Psi(w'''))_k \leq (\Psi(w))_k$,
2. $\forall k \in \{i_{j+1}, \ldots, i_m\},\ (\Psi(w'''))_k \leq (\Psi(w))_k - 1$,
3. $\forall k \in (\{1, \ldots, i_m - r\} \setminus \{i_1, \ldots, i_m\}),\ (\Psi(w'''))_k \leq (\Psi(w))_k + 1$,
4. $\forall k \in (\{i_m - r + 1, \ldots, n\} \setminus \{i_1, \ldots, i_m\}),\ (\Psi(w'''))_k \leq (\Psi(w))_k$.

– $\boxed{S}$: il reste à prouver que le franchissement de S conduit aux mêmes configurations depuis $\langle q_R^{i_1}, w, q_E^1 \rangle$. Pour cela, nous prouvons par induction sur le nombre $p \geq 1$ de motifs franchis que l'ensemble des configurations atteintes $\langle q_R^{i_{p+1}}, w''', q_E^{i_p+1-r} \rangle$ vérifient :

1. $\forall k \in \{i_1, \ldots, i_j\},\ (\Psi(w'''))_k \leq (\Psi(w))_k,$
2. $\forall k \in \{i_{j+1}, \ldots, i_p\},\ (\Psi(w'''))_k \leq (\Psi(w))_k - 1,$
3. $\forall k \in (\{1, \ldots, i_p - r\} \setminus \{i_1, \ldots, i_p\}),\ (\Psi(w'''))_k \leq (\Psi(w))_k + 1,$
4. $\forall k \in (\{i_p - r + 1, \ldots, n\} \setminus \{i_1, \ldots, i_p\}),\ (\Psi(w'''))_k \leq (\Psi(w))_k.$

où $j = \max\{l \in \{1, \ldots, p\} \mid i_l \leq i_p - r\}$ si $i_1 \leq i_p - r$ et $j = 0$ sinon.
Nous considérons que l'exécution d'un $\mathtt{e}_{\mathtt{i_j}}$-motif est précédée d'une transition de perte et suivie d'une transition de perte. Il est possible d'avoir ces deux transitions de pertes consécutives et de se ramener finalement à une seule transition de perte comme nous l'avons expliqué précédemment.

– **p = 1 (un motif franchi).**

$$\langle q_R^{i_1}, w, q_E^1\rangle \xrightarrow{\tilde{\Upsilon}}_{S \overset{\rightsquigarrow}{\shortparallel} E} \langle q_R^{i_1}, w', q_E^1\rangle \xrightarrow{M_1}{}^{*}_{S \overset{\rightsquigarrow}{\shortparallel} E} \langle q_R^{i_2}, w'', q_E^{i_1+1-r}\rangle \xrightarrow{\tilde{\Upsilon}}_{S \overset{\rightsquigarrow}{\shortparallel} E} \langle q_R^{i_2}, w''', q_E^{i_1+1-r}\rangle$$

La transition étiquetée M_1 représente de façon symbolique le franchissement du premier motif.
Par définition de $\tilde{\Upsilon}$, $\Psi(w') \leq \Psi(w)$ et $\Psi(w''') \leq \Psi(w'')$. D'après le lemme C.3, depuis $\langle q_R^{i_1}, w', q_E^1\rangle$, l'une des deux configurations suivantes est atteinte :
– Si $(\Psi(w'))_{i_1} > 0$, alors après la consommation de $\mathtt{e}_{\mathtt{i_1}}$ le système atteint la configuration $\langle q_R^{i_2}, \mathtt{e}_{\mathtt{i_1}}{}^{\ominus 1}w', q_E^1\rangle$ (lemme C.3). En posant $w'' = \mathtt{e}_{\mathtt{i_1}}{}^{\ominus 1}w'$ nous avons $j = 0$, $r = i_1$ et :
  – $(\Psi(w''))_{i_1} = (\Psi(w'))_{i_1} - 1,$
  – $\forall k \in (\{1, \ldots, n\} \setminus \{i_1\}),\ (\Psi(w''))_k = (\Psi(w'))_k.$

  Par conséquent :
  – $(\Psi(w'''))_{i_1} \leq (\Psi(w))_{i_1} - 1,$
  – $\forall k \in (\{1, \ldots, n\} \setminus \{i_1\}),\ (\Psi(w'''))_k \leq (\Psi(w))_k.$

  ce qui prouve respectivement les points (2) et (4). Les propriétés (1) et (3) sont triviallement vérifiées puisque les ensembles de valeurs de k qu'elles décrivent sont vides.
– Si $(\Psi(w'))_{i_1} = 0$, par le lemme C.3, la configuration $\langle q_R^{i_2}, w'\mathtt{e}_1 \ldots \mathtt{e}_{\mathtt{i_1-1}}, q_E^{i_1+1}\rangle$ est atteinte. Posons $w'' = w'\mathtt{e}_1 \ldots \mathtt{e}_{\mathtt{i_1-1}}$, nous avons $j = 1$, $r = 0$ et :
  – $\forall k \in \{1, \ldots, i_1 - 1\},\ (\Psi(w''))_k = (\Psi(w'))_k + 1,$
  – $\forall k \in (\{1, \ldots, n\} \setminus \{1, \ldots, i_1 - 1\}),\ (\Psi(w''))_k = (\Psi(w'))_k.$

  Donc :

  a. $\forall k \in \{1, \ldots, i_1 - 1\},\ (\Psi(w'''))_k \leq (\Psi(w))_k + 1,$

  b. $\forall k \in (\{1, \ldots, n\} \setminus \{1, \ldots, i_1 - 1\}),\ (\Psi(w'''))_k \leq (\Psi(w))_k.$

  nous obtenons que (a) prouve (3) et que (b) prouve (1) pour $k = i_1$ et (4) pour les autres valeurs de k. Cette fois, l'ensemble des valeurs de k pointées par (2) est vide, donc (2) est vérifié.

– **Étape d'induction.** Supposons que les propriétés (1) à (4) sont vérifiées jusqu'à une configuration $\langle q_R^{i_{1+p}}, w, q_E^{i_p+1-r}\rangle$ et examinons le franchissement du $(p+1)^{ième}$ motif

M_{1+p} :

$$\langle q_R^{i_{1+p}}, w, q_E^{i_p+1-r} \rangle \xrightarrow{\Upsilon}_{S \Upsilon E} \langle q_R^{i_{1+p}}, w', q_E^{i_p+1-r} \rangle \xrightarrow{M_{1+p}}{}^{*}_{S \Upsilon E} \langle q_R^{i_{1+\overline{p}}}, w'', q_E^{i_{\overline{p}}+1-\overline{r}} \rangle$$
$$\xrightarrow{\Upsilon}_{S \Upsilon E} \langle q_R^{i_{1+\overline{p}}}, w''', q_E^{i_{\overline{p}}+1-\overline{r}} \rangle$$

où $\overline{j}$, $\overline{p}$ et $\overline{r}$ reprśentent les nouvelles valeurs de j, p et r, respectivement, après le franchissement du motif, reprśenté par la transition d'étiquette M_{1+p}.

Par définition de Υ, $\Psi(w') \leq \Psi(w)$ et $\Psi(w''') \leq \Psi(w'')$. De plus, d'après le lemme C.3, depuis $\langle q_R^{i_{1+p}}, w', q_E^{i_p+1-r} \rangle$, la configuration atteinte est l'une des suivantes, selon la valeur de $(\Psi(w'))_{i_{1+p}}$:

– Soit $(\Psi(w'))_{i_{1+p}} > 0$ auquel cas, le système atteint la configuration $\langle q_R^{i_{1+\overline{p}}}, \mathtt{i_{1+p}}^{\ominus 1} w',$ $q_E^{i_p+1-r} \rangle$, donc $\overline{j} = j$, $i_{\overline{p}} - \overline{r} = i_p - r$ et, en posant $w'' = \mathtt{i_{1+p}}^{\ominus 1} w'$:

  a. $(\Psi(w''))_{i_{1+p}} = (\Psi(w'))_{i_{1+p}} - 1$,

  b. $\forall k \in (\{1, \ldots, n\} \setminus \{i_{1+p}\}),\ (\Psi(w''))_k = (\Psi(w'))_k$.

  Alors, il vient que :

  1. par hypothèse d'induction sur (1) : $\forall k \in \{i_1, \ldots, i_{\overline{j}}\},\ (\Psi(w'''))_k \leq (\Psi(w))_k$,
  2. $\forall k \in \{i_{\overline{j}+1}, \ldots, i_{\overline{p}}\},\ (\Psi(w'''))_k \leq (\Psi(w))_k - 1$ par induction sur (2) et en utilisant (a) et l'hypothèse d'induction sur (4) pour $i_{\overline{p}}$,
  3. puisque $i_{\overline{p}} \notin \{1, \ldots, i_p - r\}$, en utilisant (b) et l'induction sur (3) : $\forall k \in (\{1, \ldots, i_{\overline{p}} - \overline{r}\} \setminus \{i_1, \ldots, i_{\overline{p}}\}),\ (\Psi(w'''))_k \leq (\Psi(w))_k + 1$,
  4. enfin, par (b) et l'hypothèse d'induction sur (4), nous obtenons : $\forall k \in (\{i_{\overline{p}} + 1 - \overline{r}, \ldots, n\} \setminus \{i_1, \ldots, i_{\overline{p}}\}),\ (\Psi(w'''))_k \leq (\Psi(w))_k$.

– Soit $(\Psi(w'))_{i_{1+p}} = 0$, alors la configuration $\langle q_R^{i_{1+(1+p)}}, w' \mathtt{e}_{\mathtt{i_p+1-r}} \ldots \mathtt{e}_{\mathtt{i_{(1+p)}-1}}, q_E^{i_{(1+p)}+1} \rangle$ est atteinte. Posons $w'' = w' \mathtt{e}_{\mathtt{i_p+1-r}} \ldots \mathtt{e}_{\mathtt{i_{(1+p)}-1}}$, nous avons $\overline{j} = 1 + p = \overline{p}$, $\overline{r} = 0$ et :

  a. $\forall k \in \{i_p + 1 - r, \ldots, i_{1+p} - 1\},\ (\Psi(w''))_k = (\Psi(w'))_k + 1$,

  b. $\forall k \in (\{1, \ldots, n\} \setminus \{i_p + 1 - r, \ldots, i_{1+p} - 1\}),\ (\Psi(w''))_k = (\Psi(w'))_k$.

  Alors :

  1. Par hypothèse d'induction sur (1) pour $k \in \{i_1, \ldots, i_j\}$, en utilisant (a) et l'induction sur (2) pour $k \in \{i_{j+1}, \ldots, i_p\}$ et (b) et l'hypothèse d'induction sur (4) pour i_{1+p}, nous avons : $\forall k \in \{i_1, \ldots, i_{\overline{p}}\},\ (\Psi(w'''))_k \leq (\Psi(w))_k$,
  3. avec (b), l'hypothèse d'induction sur (3) et sachant que $\overline{r} = 0$: $\forall k \in (\{1, \ldots, i_{\overline{p}} - \overline{r}\} \setminus \{i_1, \ldots, i_{\overline{p}}\}),\ (\Psi(w'''))_k \leq (\Psi(w))_k + 1$,
  4. par hypothèse d'induction sur (4) et puisque $\overline{r} = 0$ nous obtenons : $\forall k \in (\{i_{\overline{p}} + 1 - \overline{r}, \ldots, n\} \setminus \{i_1, \ldots, i_{\overline{p}}\}),\ (\Psi(w'''))_k \leq (\Psi(w))_k$.

  La preuve de (2) est triviale du fait que l'ensemble des valeurs de k référencées est vide.

♦

C.2.3 Atomicité des séquences ordonnées et complètes de $\mathtt{e_i}$-motifs.

Une séquence ordonnée, complète et embarquée de $\mathtt{e_i}$-motifs (définition C.9, page 212) est *atomique* s'il n'est pas permis de perdre des événements pendant l'exécution des transitions de la séquence. Ainsi, une exécution d'une telle séquence débute par une transition de perte, puis le franchissement des transitions de la séquence, sans perte, et finalement, une transition de perte lorsque toutes les transitions franchissables en $q_R^{i_{m+1}}$ ont été exécutées.

Lemme C.14
Soient S_{c_a} et S_c deux Lossy séquences ordonnées complètes et embarquées de $\mathtt{e_{i_1}}, \ldots, \mathtt{e_{i_m}}$-motifs, respectivement atomique et non-atomique. Depuis une configuration $\langle q_R^{i_1}, w, q_E^1 \rangle$, les ensembles d'états atteints par S_{c_a} et S_c sont égaux. ♦

Preuve. Par le lemme C.13, les deux séquences S_{c_a} et S_c atteignent l'état $q_R^{i_{m+1}}$ avec les mêmes ensembles de configurations. Notons que la perte d'une occurrence mémorisée ne modifie par l'exécutabilité d'une transition de mémorisation : toutes les configurations atteintes par la séquence :

$$\langle q_R^{i_{m+1}}, w, q_E^{i_m+1-r} \rangle \xrightarrow{\tau}_{S \overset{\leadsto}{\shortparallel} E} \langle q_R^{i_{m+1}}, w', q_E^{i_m+1-r} \rangle \xrightarrow{!\mathtt{e_{i_m+1-r}}}_{S \overset{\leadsto}{\shortparallel} E} \langle q_R^{i_{m+1}}, w'\mathtt{e_{i_m+1-r}}, q_E^{i_m+2-r} \rangle$$

sont aussi atteintes par la séquence :

$$\langle q_R^{i_{m+1}}, w, q_E^{i_m+1-r} \rangle \xrightarrow{!\mathtt{e_{i_m+1-r}}}_{S \overset{\leadsto}{\shortparallel} E} \langle q_R^{i_{m+1}}, w\mathtt{e_{i_m+1-r}}, q_E^{i_m+2-r} \rangle \xrightarrow{\tau}_{S \overset{\leadsto}{\shortparallel} E} \langle q_R^{i_{m+1}}, w', q_E^{i_m+2-r} \rangle$$

et vice-versa, ce sont les configurations : $\langle q_R^{i_{m+1}}, w', q_E^{i_m+2-r} \rangle$ avec $w' \preccurlyeq w\mathtt{e_{i_m+1-r}}$. Donc, les exécutions de S qui consistent à mémoriser les événements $\mathtt{e_{i_m+1-r}}$ à $\mathtt{e_n}$, puis à franchir une transition de perte permettent d'atteindre toutes les configurations que l'ensemble des exécutions de S atteignent. Par conséquent, les ensembles de configurations atteints par S_{c_a} et S_c sont égaux. ♦

C.2.4 Preuve de la simulation des Lossy MC par les Lossy SFR Embarqués

À partir de l'AFR à compteurs R (de sémantique S) de la définition C.1 (page 205) et de l'environnement E de la figure C.2 (page 206), nous obtenons le *Lossy système à file réactif embarqué à compteurs (Lossy SFR Embarqué à compteurs)* $S \overset{\leadsto}{\shortparallel} E$ en appliquant les définitions 4.3 (page 90) et 2.21 (page 51) qui simule une Lossy machine à n compteurs déterministe $\overset{\leadsto}{M}$ donnée.

Soient $\langle q, m_1, \ldots, m_n \rangle$ et $\langle q_R, w, q_E^0 \rangle$ deux configurations de $\overset{\leadsto}{M}$ et $S \overset{\leadsto}{\shortparallel} E$ respectivement. La relation d'équivalence[3] $\sim_\Psi$ entre les configurations de $\overset{\leadsto}{M}$ et $S \overset{\leadsto}{\shortparallel} E$ est définie par :

$$\langle q, m_1, \ldots, m_n \rangle \sim_\Psi \langle q_R, w, q_E \rangle \Leftrightarrow q = q_R \quad \text{et} \quad \Psi(w) = \begin{pmatrix} m_1 & m_2 & \ldots & m_n \end{pmatrix} \quad \text{et} \quad q_E = q_E^0$$

Nous prouvons que $S \overset{\leadsto}{\shortparallel} E$ simule $\overset{\leadsto}{M}$ en terme d'accessibilité vis-à-vis de Q :

[3] Cette relation est l'extension à n compteurs de la relation $\sim_\Psi$ définie page 75.

– une configuration est accessible dans $\overset{\leadsto}{M}$ si et seulement si une configuration $\sim_\Psi$-équivalente est accessible dans $S \overset{\rightrightarrows}{\shortparallel} E$.
– soient $\langle q, m_1, \ldots, m_n\rangle$ et $\langle q_R, w, q_E\rangle$ deux configurations accessibles et équivalentes de $\overset{\leadsto}{M}$ et $S \overset{\rightrightarrows}{\shortparallel} E$ respectivement. Alors, $\langle q, m_1, \ldots, m_n\rangle \rightarrow \langle q', m'_1, \ldots, m'_n\rangle$ si et seulement si $\langle q_R, w, q_E\rangle \rightarrow^*_{S \overset{\rightrightarrows}{\shortparallel} E} \langle q'_R, w', q'_E\rangle$, et $\langle q', m'_1, \ldots, m'_n\rangle \sim_\Psi \langle q'_R, w', q'_E\rangle$.

Preuve (théorème 4.26, page 105). Considérons les structures de simulation des instructions de $\overset{\leadsto}{M}$ représentées en figure C.1 (page 206). Nous prouvons que dans le cadre d'une sémantique de perte, considérer que ces structures sont atomiques préserve l'ensemble des états accessibles.

Pour la structure de simulation de l'incrémentation représentée en figure C.1(a), nous pouvons considérer que la séquence ordonnée complète de q_R^1 à q_R^{n+1} est atomique par le lemme C.14. Par conséquent, lors de l'exécution de cette séquence, les transitions de perte peuvent survenir en q_R, en q'_R (ce qui correspond à l'effet souhaité, c'est à dire l'atomicité de la séquence), en q_R^1 (avant exécution des transitions issues de cet état) et en q_R^{n+1} (après exécution des transitions entrant dans cet état). Par définition des transitions de traitement immédiat (point (4b), de la défintion 2.11, page 44), la file reste invariante par les transitions $q_R \xrightarrow{\tau}_R q_R^1$ et $q_R^{n+1} \xrightarrow{\tau}_R q'_R$. De plus, la perte d'occurrences mémorisée ne modifie pas l'exécutabilité de ce type de transition puisque la stabilité des configurations est invariante par la perte. Par conséquent, toutes les pertes d'occurrences mémorisées qui se produisent en q_R^1 peuvent se produire en q_R (il y a les mêmes occurrences mémorisées), et avec le même effet (l'ensemble des configurations atteintes est le même) :

$$\langle q_R, w, q_E^0\rangle \xrightarrow{\tau}_{S \overset{\rightrightarrows}{\shortparallel} E} \langle q_R^1, w, q_E^1\rangle \xrightarrow{\tau}_{S \overset{\rightrightarrows}{\shortparallel} E} \langle q_R^1, w', q_E^1\rangle, \qquad w' \preccurlyeq w$$

est équivalente à :

$$\langle q_R, w, q_E^0\rangle \xrightarrow{\tau}_{S \overset{\rightrightarrows}{\shortparallel} E} \langle q_R, w', q_E^0\rangle \xrightarrow{\tau}_{S \overset{\rightrightarrows}{\shortparallel} E} \langle q_R^1, w', q_E^1\rangle, \qquad w' \preccurlyeq w$$

en terme d'accessibilité. Par conséquent, la structure de simulation de l'incrémentation peut être considérée atomique vis à vis des transitions de perte.

Pour la structure de simulation de l'instruction de test à zéro/décrémentation décrite en figure C.1(b), par le lemme C.14, les pertes d'occurrences mémorisées peuvent survenir en q_R, en q'_R, en q''_R, en q_R^1 et en q_R^{n+1}. De même que précédemment, toute perte se produisant en q_R^1 (resp. q_R^{n+1}) peut se produire, avec le même effet (en terme d'accessibilité) en q_R (resp. q'_R). Donc cette structure est elle-aussi atomique vis à vis des pertes.

Alors, chaque instruction (atomique par définition) de la Lossy machine à compteur $\overset{\leadsto}{M}$ est simulée par une structure atomique du Lossy SFR Embarqué $S \overset{\rightrightarrows}{\shortparallel} E$, et chaque transition de perte de $\overset{\leadsto}{M}$ est simulée par une transition de perte dans $S \overset{\rightrightarrows}{\shortparallel} E$. Nous en déduisons que le théorème 3.6 (page 76) s'étend aux Lossy machines à n compteurs déterministes et aux Lossy SFR Embarqué avec n événements mémorisables.

Notons que nous n'avons pas traité le cas de la structure de terminaison reprsésentée en figure C.1(c). Puisqu'elle n'intervient pas dans la preuve du théorème 3.6, nous pouvons l'ignorer. Remarquons de plus que toute transition de perte qui survient lors de son exécution fait diminuer le nombre d'occurrences mémorisées. Ceci garantit que $\overset{\rightsquigarrow}{M}$ est bornée si et seulement si $S \overset{\rightsquigarrow}{\shortparallel} E$ est borné. ♦

Annexe D

Preuves des résultats de réduction par méthode d'ordre partiel

D.1 Preuve du théorème d'indépendance pour le langage Electre

Nous rappelons ici l'énoncé du théorème 7.9, page 165 :

> Soient P un programme Electre et $\sigma \in \mathcal{E}^*$ une séquence d'événements. Quel que soit $\sigma' \in [\sigma]$, si $\{\sigma\} \vdash P \xrightarrow{*} p \rhd P' \xrightarrow{*} p'$ et $\{\sigma'\} \vdash P \xrightarrow{*} p \rhd P'' \xrightarrow{*} p''$ alors $p' = p''$.

Nous ne donnons pas de preuve complète de ce théorème car elle est purement mécanique et fastidieuse, tout en n'étant pas d'un grand interêt car il est aisé de se convaincre de la justesse du théorème 7.9. Afin de le prouver, nous devrions montrer ce résultat par induction sur les règles de reécriture du langage Electre présentées en annexe A, section A.2 page 194. Nous nous contentons d'indiquer la voie à suivre sur l'exemple de la régle $J_1 ::= I \mid\mid J_2$ de la grammaire d'Electre (présentée en annexe, section A.1, page 193).

Nous devons donc considérer deux événements, `e` et `e`$'$, le premier figurant dans I, et le second dans J_2. Ensuite, il faut considérer que sur occurrence de `e`, la dérivation issue de I, $I \xrightarrow{*} i$ peut se récrire de 3 façons différentes soit $I' \xrightarrow{*} i'$, soit $C \xrightarrow{*} c$, soit $I' \xrightarrow{*}$ `nil`, correspondant respectivement aux 3 règles (xvii1), (xvii2) et (xvii3). Puis il faut considérer les réécritures des dérivations ainsi obtenues sur occurrence de `e`$'$ mettant en oeuvre à nouveau les règles (xvii1), (xvii4) et (xvii5) dans les premier et troisième cas, et les règles (ii1) à (ii4) dans le deuxième. Ensuite, il faut appliquer le même principe en considérant tout d'abord les réécritures sur occurrence de `e`$'$ puis sur occurrence de `e`, et finalement, il faut vérifier la convergence de réécriture avec la première séquence `ee`$'$. Cette démarche doit être appliquée pour tous les opérateurs du langage, à l'exception de "`|`", "`await`" et "`when`".

Notons cependant que l'indépendance entre les événements induite par les opérateurs "`:`" et "`;`" provient du fait que ces opérateurs introduisent la séquentialité dans les programmes Electre, et par conséquent, tout événement "à droite" de ces opérateurs ne pourra être traité que lorsque l'exécution de la "partie gauche" de l'opérateur sera terminée. À l'inverse, l'indépendance introduite par les autres opérateurs provient de la possibilité de traiter des occurrences d'événements en parallèle sans que l'ordre de traitement n'intervienne.

D.2 Preuve de la complexité du modèle de la file dans le cas au pire

Nous prouvons par induction le lemme 7.17, page 174 qui affirme qu'au rang n, le nombre d'états du modèle de la file est :

$$rang(n) = (\eta_*)^n + n \times \sum_{i=0}^{n-2} \left[\left(\prod_{j=0}^{i} (\eta_1 - j) \right) \times (\eta_*)^{(n-(i+1))} \right] + \prod_{i=0}^{n-1} (\eta_1 - i)$$

Les états de rang $n+1$ sont obtenus depuis ceux de rang n en ajoutant (mémorisant) un événement qui peut être soit à mémorisation unique, soit à mémorisation multiple. S'il est toujours possible d'ajouter un événement à mémorisation multiple, cela n'est pas toujours possible pour les événements à mémorisation unique : lorsque $n+1 > \eta_1$. Nous pouvons cependant ignorer ce problème car, s'il se présente, les produits de la forme $\prod(\eta_1 - j)$ sont nuls : il existe un $j = \eta_1$. Ainsi, nous obtenons bien un nombre d'états nul au final. Nous avons donc :

$$\begin{aligned}
rang(n+1) = {} & (\eta_*)^n \times (\eta_1 + \eta_*) + n \times \sum_{i=0}^{n-2} \left[\left(\prod_{j=0}^{i} (\eta_1 - j) \right) \times (\eta_*)^{(n-(i+1))} \times ((\eta_1 - (i+1)) + \eta_*) \right] \\
& + \left[\left(\prod_{i=0}^{n-1} (\eta_1 - i) \right) \times ((\eta_1 - n) + \eta_*) \right] \\
\\
= {} & (\eta_*)^{(n+1)} \\
& + n \times \left\{ \sum_{i=0}^{n-2} \left[\left(\prod_{j=0}^{i} (\eta_1 - j) \right) \times (\eta_*)^{(n+1-(i+1))} \right] + \sum_{i=0}^{n-2} \left[\left(\prod_{j=0}^{i+1} (\eta_1 - j) \right) \times (\eta_*)^{(n-(i+1))} \right] \right\} \\
& + (\eta_*)^n \times \eta_1 + \left(\prod_{i=0}^{n-1} (\eta_1 - i) \times \eta_* \right) + \prod_{i=0}^{n} (\eta_1 - i) \\
\\
= {} & (\eta_*)^{(n+1)} + n \times (\eta_1 \times (\eta_*)^n) + n \times \left[\left(\prod_{j=0}^{n-1} (\eta_1 - j) \right) \times \eta_* \right] \\
& + n \times \left\{ \sum_{i=1}^{n-2} \left[\left(\prod_{j=0}^{i} (\eta_1 - j) \right) \times (\eta_*)^{(n+1-(i+1))} \right] + \sum_{i=0}^{n-3} \left[\left(\prod_{j=0}^{i+1} (\eta_1 - j) \right) \times (\eta_*)^{(n-(i+1))} \right] \right\} \\
& + (\eta_*)^n \times \eta_1 + \left(\prod_{i=0}^{n-1} (\eta_1 - i) \times \eta_* \right) + \prod_{i=0}^{n} (\eta_1 - i) \\
\\
= {} & (\eta_*)^{(n+1)} + (n+1) \times \sum_{i=0}^{n-1} \left[\left(\prod_{j=0}^{i} (\eta_1 - j) \right) \times (\eta_*)^{(n+1-(i+1))} \right] + \prod_{i=0}^{n} (\eta_1 - i)
\end{aligned}$$

D.3 Preuve de la complexité du modèle de la file dans le meilleur des cas

Nous prouvons par induction sur n que dans le cas au mieux, le nombre d'états du modèle réduit pour la file au rang n est donné par le lemme 7.18, page 175 :

$$rang(n) = \sum_{i=0}^{n} \left[\left(\prod_{j=0}^{i-1} (\eta_1 - j) \right) \times \left(\sum_{p_1=0}^{(n-i)} \sum_{p_2=0}^{(n-i-p_1)} \cdots \sum_{p_{(\eta_*-1)}=0}^{\left(n-i-\sum_{k=1}^{(\eta_*-2)} p_k\right)} 1 \right) \right]$$

Les états de rang $n+1$ sont obtenus depuis ceux de rang n en leur ajoutant soit un événement à mémorisation unique, soit un événement à mémorisations multiples. De même que pour le cas précédent, il n'est pas nécessaire de se préoccuper du cas ou $n+1 > \eta_1$. Nous avons :

$$\begin{aligned} rang(n+1) = & \sum_{i=0}^{n} \left[\left(\prod_{j=0}^{i-1} (\eta_1 - j) \right) \times \left(\sum_{p_1=0}^{(n+1-i)} \sum_{p_2=0}^{(n+1-i-p_1)} \cdots \sum_{p_{(\eta_*-1)}=0}^{\left(n+1-i-\sum_{k=1}^{(\eta_*-2)} p_k\right)} 1 \right) \right] \\ & + \sum_{i=0}^{n} \left[\left(\prod_{j=0}^{i} (\eta_1 - j) \right) \times \left(\sum_{p_1=0}^{(n+1-i)} \sum_{p_2=0}^{(n+1-i-p_1)} \cdots \sum_{p_{(\eta_*-1)}=0}^{\left(n+1-i-\sum_{k=1}^{(\eta_*-2)} p_k\right)} 1 \right) \right] \\ = & \sum_{i=0}^{n+1} \left[\left(\prod_{j=0}^{i-1} (\eta_1 - j) \right) \times \left(\sum_{p_1=0}^{(n+1-i)} \sum_{p_2=0}^{(n+1-i-p_1)} \cdots \sum_{p_{(\eta_*-1)}=0}^{\left(n+1-i-\sum_{k=1}^{(\eta_*-2)} p_k\right)} 1 \right) \right] \end{aligned}$$

Thèse de doctorat en Automatique et Informatique Appliquée,
réalisée de septembre 1998 à décembre 2001,
à l'IRCCyN, Nantes.

Doctorat obtenu le vendredi 14 décembre 2001, avec la mention très
honorable et les félicitations du jury,
à l'IRCCyN, École Centrale de Nantes,
Nantes.

Dernières corrections apportées le 4 février 2003.