Vers des outils efficaces pour la vérification de systèmes concurrents

Thomas Geffroy

université de BORDEAUX

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

par **Thomas Geffroy**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Vers des outils efficaces pour la vérification de systèmes concurrents

---

Date de soutenance : 12 Décembre 2017

Devant la commission d'examen composée de :

Nathalie Bertrand	Chargée de recherche, Inria Bretagne-Atlantique	Examinatrice
Serge Haddad	Professeur, ENS Paris-Saclay	Rapporteur
Colette Johnen	Professeur, Université de Bordeaux	Présidente du jury
Jérôme Leroux	Directeur de recherche, CNRS	Directeur
Mihaela Sighireanu	Maître de conférences, Université Paris Diderot .	Rapporteur
Grégoire Sutre	Chargé de recherche, CNRS	Co-Directeur

– 2017 –

Vers des outils efficaces pour la vérification de systèmes concurrents

Cette thèse cherche à résoudre *en pratique* le problème de couverture dans les réseaux de Petri et les systèmes de canaux à pertes (LCS). Ces systèmes sont intéressants à étudier car ils permettent de modéliser facilement les systèmes concurrents et les systèmes distribués. Le problème de couverture dans un système de transitions consiste à savoir si on peut, à partir d'un état initial arriver à un état plus grand qu'un état cible. La résolution de ce problème dans les systèmes de transitions bien structurés (WSTS) sera le sujet d'études de la première partie. Les réseaux de Petri et les LCS sont des WSTS. On donnera dans la première partie une méthode générale pour le résoudre rapidement en pratique. Cette méthode utilise des invariants de couverture, qui sont des sur-approximations de l'ensemble des états couvrables. La seconde partie sera consacrée aux réseaux de Petri. Elle présentera diverses comparaisons théoriques et pratiques de différents invariants de couverture. Nous nous intéresserons notamment à la combinaison de l'invariant classique de l'inéquation d'état avec une analyse de signe simple. Les LCS seront le sujet d'études de la troisième partie. On présentera une variante de l'inéquation d'état adaptée aux LCS ainsi que deux invariants qui retiennent des propriétés sur l'ordre dans lequel les messages sont envoyés. La thèse a mené à la création de deux outils, `ICover` et `BML`, pour résoudre le problème de couverture respectivement dans les réseaux de Petri et dans les LCS.

Towards efficient tools for the verification of concurrent systems

The goal of this thesis is to solve *in practice* the coverability problem in Petri nets and lossy channel systems (LCS). These systems are interesting to study because they can be used to model concurrent and distributed systems. The coverability problem in a transition system is to decide whether it is possible, from an initial state, to reach a greater state than a target state. In the first part, we discuss how to solve this problem for well-structured transition systems (WSTS). Petri nets and LCS are WSTS. In the first part, we present a general method to solve this problem quickly in practice. This method uses coverability invariants, which are over-approximations of the set of coverable states. The second part studies Petri nets. We present comparisons of coverability invariants, both in theory and in practice. A particular attention will be paid on the combination of the classical state inequation and a simple sign analysis. LCS are the focus of the third part. We present a variant of the state inequation for LCS and two invariants that compute properties for the order in which messages are sent. Two tools, `ICover` and `BML`, were developed to solve the coverability problem in Petri nets and LCS respectively.

Mots-clés Méthodes Formelles, Réseau de Petri, Vérification, Algorithmes, Système de canaux à pertes

Keywords Formal methods, Petri net,Verification, Algorithms, Lossy Channel System

Laboratoire d'accueil LaBRI Domaine universitaire, 351, cours de la Libération, 33405 Talence

Table des matières

Introduction

Les systèmes informatiques qui nous entourent sont de plus en plus présents et de plus en plus complexes. Les *smartphones* sont utilisés constamment, de plus en plus d'objets connectés communiquent avec ces téléphones ou entre eux. Des voitures autonomes nous sont promises dans les prochaines années. Notre dépendance aux machines est actée.

Le mauvais fonctionnement d'un système informatique peut être catastrophique. On trouve l'exemple mortel de la machine de radiothérapie Therac-25 dont le mauvais fonctionnement entre 1985 et 1987 a entraîné la mort d'au moins cinq personnes. Le programme informatique contrôlant les doses de radiations que la machine envoyait aux patients était défaillant. Un des bugs pouvait survenir lorsque l'opérateur de la machine changeait rapidement les paramètres. Les premiers testeurs n'avaient pas pu découvrir ce bug car ils étaient moins à l'aise avec l'interface qu'un opérateur médical aguerri. Dans le domaine spatial, le dysfonctionnement de l'ordinateur de bord du lanceur européen Ariane V a causé la perte de quatre satellites de la mission Cluster de l'Agence spatiale européenne lors du vol 501 en 1996. Cet échec aura coûté autour d'un demi milliard d'euros. Un autre exemple est celui du bug dans l'unité de calcul en virgule flottante de certains modèles de processeur Pentium d'Intel au début des années 90. Par exemple, pour l'opération $4195835,0$ divisé par $3145727,0$ le processeur pouvait retourner la valeur $1,333739068902037589$ et non $1,333820449136241002$. Ce problème était dans le fonctionnement même de l'architecture du processeur, ce n'était pas un bug qui pouvait être résolu avec une mise à jour. Intel a donc dû changer l'architecture de ses processeurs, et remplacer les processeurs fautifs. Les bugs peuvent être évités par différents moyens : une plus grande attention à la qualité du code, plus de tests, une meilleure communication entre les différentes équipes impliquées dans l'élaboration d'un système, etc. On va s'intéresser aux moyens formels de vérifier le bon fonctionnement d'un système.

Plusieurs outils sont utilisés par l'industrie. Il y a par exemple l'outil `Astrée` Cousot *et al.* [2001], notamment utilisé par Airbus Souyris et Delmas [2007], qui permet d'analyser du code écrit en langage `C` pour trouver des bugs potentiels. Cet outil utilise l'interprétation abstraite Cousot et Cousot [1977]. L'outil `SLAM` slam est un outil d'analyse statique proposé par Micro-

soft Ball *et al.* [2004] pour les développeurs de pilotes de périphérique pour les systèmes d'exploitation Windows. Cet outil utilise la méthode Cegar (*counter example-guided abstraction refinement*) Clarke [2003]. L'architecture de processeur `PowerScale` a été vérifiée Chehaibar *et al.* [1996] en utilisant le langage de spécification LOTOS Bolognesi et Brinksma [1987]. On voit donc qu'il existe déjà des moyens de vérifier formellement le comportement de certains systèmes.

Les systèmes présents aujourd'hui ont évolué. Ils sont de plus en plus complexes et nombreux. Les processeurs des premiers *smartphones* n'avaient qu'un unique coeur, ils peuvent en avoir maintenant quatre ou six. L'internet des objets promet de multiplier le nombre de systèmes et donc le nombre d'interactions entre les différents systèmes.

Les outils de vérification utilisés dans l'industrie sont en général faits pour des systèmes séquentiels. On peut toutefois citer l'outil `SCADE` SCADE qui utilise le langage `Lustre` Bergerand [1986]. Les systèmes concurrents posent des problèmes supplémentaires. On en distingue en général deux types, les systèmes *multi-threads* et les systèmes distribués. Les systèmes avec plusieurs *threads* doivent par exemple se partager des données. Les systèmes distribués sont des systèmes autonomes qui communiquent entre eux via des canaux de communication de différents types. Du point de vue d'un programmeur, ces systèmes sont plus durs à concevoir qu'un programme séquentiel. Il faut réussir à comprendre toutes les interactions que chaque système ou processus peut avoir avec tous les autres. On a donc des raisons supplémentaires de vouloir vérifier formellement que ces systèmes répondent à des spécifications précises.

1. Vérifier des systèmes.

Il y a différentes façon de vérifier le bon fonctionnement d'un système. On peut partir d'un programme déjà écrit. Ensuite on abstrait ce programme P en un modèle M qui est plus facile à analyser que P. On veut vérifier le bon fonctionnement de ce programme, par exemple on veut vérifier que les instructions `assert` sont toujours vraies. Ces assertions sont traduites dans le modèle M en une propriété ϕ. On vérifie ensuite que le modèle M satisfait ϕ. Si ce n'est pas le cas et qu'on trouve un contre-exemple sur le modèle M, on cherche à reproduire ce comportement sur le programme P. Il y a deux cas : soit il y a effectivement une erreur dans le programme P, et on doit donc corriger ce bug, soit le contre-exemple n'est pas reproductible. Dans ce dernier cas, le modèle M était une abstraction trop imprécise et on va donc choisir un modèle plus précis M' tel que le contre-exemple n'est plus possible. Toutefois M' sera potentiellement plus difficile à analyser. On itère ces opérations jusqu'à trouver un contre-exemple dans le programme ou un modèle qui satisfait la propriété ϕ. Dans ce cas, nous obtenons une preuve que les instructions `assert` sont toujours vraies dans le programme P.

Une autre approche consiste à partir de spécifications qui sont décrites dans

un langage naturel, en français, en anglais, etc. Le fonctionnement du système n'est pas encore décrit de manière formelle. Il faut donc que ces spécifications soient transformées en un modèle M, qui lui sera sans ambiguïté. La même problématique s'applique pour les exigences qu'on a sur ce système. Il faut formaliser ces exigences en une propriété ϕ. On veut alors vérifier que ϕ est satisfaite sur le modèle M. Si le modèle M satisfait ϕ alors, sous réserve que les spécifications et les exigences ont été bien traduites, on a montré que les spécifications respectent les exigences. Si par contre la vérification montre que le modèle ne satisfait pas ϕ, et trouve un contre-exemple, on peut modifier les spécifications pour éviter un bug. Et recommencer le même processus. D'autres approches de vérification existent.

Pour vérifier que la propriété vérifie ou non le modèle il y a plusieurs approches, par exemple la preuve assistée, les tests et la vérification de modèle. Dans cette thèse, on s'intéressera à cette dernière approche.

On peut chercher à vérifier différents types de propriétés. Dans cette thèse on s'intéressera uniquement aux propriétés de sûreté. Une propriété de sûreté est une propriété qui, lorsqu'elle n'est pas satisfaite, admet des contre-exemples sous la forme d'exécutions finies. Cela peut être la propriété qu'il n'existe pas d'exécution d'un programme qui arrive dans une certaine ligne de code. Dans les autres propriétés il y a notamment les propriétés d'équité, pour par exemple vérifier que dans un système concurrent avec plusieurs processus, dans chaque exécution infinie, tous les programmes peuvent avancer.

Il y a deux aspects à considérer pour choisir un modèle. D'abord il faut que l'abstraction du programme ou la traduction des spécifications puisse être assez précise. Mais il faut aussi que le vérificateur de modèle puisse conclure. Si le modèle choisi est trop expressif, par exemple une machine de Turing, les problèmes seront indécidables. Si le modèle est trop peu expressif, par exemple si on utilise un modèle qui a un nombre fini d'états possibles, on ne pourra pas vérifier toutes les propriétés que l'on voudrait. Il faut trouver un juste milieu entre expressivité et décidabilité. Les réseaux de Petri et les systèmes de canaux à pertes (LCS) sont tous les deux de bons compromis. Ils ne sont pas limités à un nombre fini d'états, ce qui est utile, par exemple quand le nombre de processus d'un système *multi-threads* n'est pas borné ou est trop grand pour être précisément modélisé, ou quand la taille des canaux de communications d'un système distribué n'est pas borné non plus ou est trop grand pour être précisément modélisé. Les systèmes *multi-threads* peuvent être modélisés par des réseaux de Petri, et les LCS sont utiles pour vérifier les systèmes distribués.

2. Réseaux de Petri

Nous présentons la classe de modèle des réseaux de Petri à l'aide d'une variante du problème des lecteurs-écrivains. Dans ce problème, différents processus doivent se partager une ressource. On ne veut pas qu'un processus accède en écriture à cette ressource pendant que d'autres processus y accèdent,

que ce soit en écriture ou en lecture. De plus on veut qu'à tout moment, aux plus trois processus accèdent en lecture à la ressource. On appelle lecteur et écrivain respectivement étiquetées un processeur qui accède en lecture à la ressource et un processeur qui accède en écriture à la ressource. On a donc comme spécification les deux règles suivantes :

— Un processus peut écrire seulement si aucun processus n'est en train de lire ou en train d'écrire.
— Un processus peut lire seulement si aucun processus n'est en train d'écrire et si aux plus deux processus sont en train de lire.

Les exigences sont : il y a au plus un écrivain, il y a au plus trois lecteurs et il n'y a pas de lecteur si il y a un écrivain.

On a modélisé ce système avec le réseau de Petri $\mathcal{N}$ de la Figure 1. Un réseau de Petri est composé de *places*, qui peuvent contenir des *jetons*, et de *transitions* qui en fonction du nombre de jetons à certaines places peuvent ajouter ou enlever des jetons. Les places sont représentées par des ronds, les jetons par des points et les transitions par des carrés. On explique la sémantique d'une transition avec l'exemple de la transition t_3. Sur le schéma on voit qu'il y a deux flèches qui vont vers t_3 depuis $q_{capacité}$ et q_{oisif}, respectivement avec le nombre 3 et sans nombre. Il y a une unique flèche sortant vers $q_{écrivain}$. Cela veut dire que quand on exécute t_3 on prend trois jetons dans la place $q_{capacité}$, un jeton dans la place q_{oisif} et qu'on met un jeton dans $q_{écrivain}$. Un réseau de Petri a aussi un ensemble de marquages initiaux, qui pour la Figure 1 est $Init$ l'ensemble des marquages qui ont trois jetons dans $q_{capacité}$, aucun jeton dans $q_{écrivain}$, $q_{lecteur}$ et q_{oisif}.

L'intuition de la modélisation est la suivante. Le nombre de jetons dans les places q_{oisif}, $q_{lecteur}$ et $q_{écrivain}$ représente le nombre de processus ne faisant rien, en train de lire et en train d'écrire respectivement. Le nombre de jetons dans la place $q_{capacité}$ correspond au nombre maximum de processus qui peuvent se mettre à lire. La transition t_c ne prend aucun jeton mais émet un jeton dans q_{oisif}. C'est la transition qui modélise la création d'un processus, qui sera initialement oisif. Un état du système qui décrit combien il y a de jetons dans chacune des places est appelé un marquage.

Pour décrire de manière informelle un marquage on notera par exemple $m = 3 \cdot q_{oisif} + q_{lecteur} + 2 \cdot q_{capacité}$ pour dire qu'il y a trois jetons dans q_{oisif}, un jeton dans $q_{lecteur}$ et deux dans $q_{capacité}$.

Une propriété de sûreté qu'on veut vérifier est d'avoir un unique processus en écriture. Cela veut dire qu'il ne faut pas aller dans un marquage m qui contient au moins deux jetons dans la place $q_{écrivain}$. Formellement on veut qu'aucun marquage accessible m soit plus grand que $m_{final} = 2 \cdot q_{écrivain}$. C'est ce qu'on appelle le problème de couverture.

Cet exemple montre comment utiliser le problème de couverture dans les réseaux de Petri pour vérifier des systèmes *multi-threads*. La méthode générale est de représenter les états des processus par des places et chaque jeton dans

une place indique qu'il y a un processus dans l'état correspondant à la place.

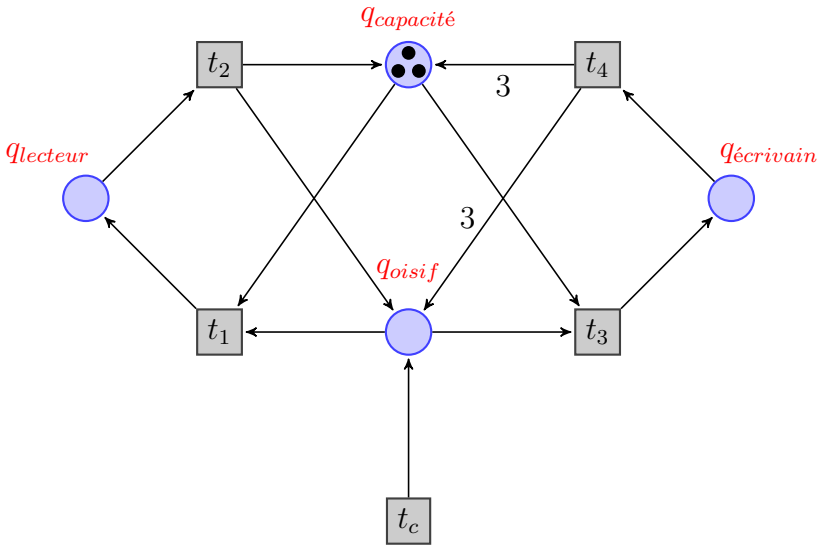

FIGURE 1 – Réseau de Petri modélisant le problème lecteurs-écrivains

Il existe des outils pour obtenir des réseaux de Petri à partir de programmes *multi-threads*. L'outil `Soter` D'Osualdo *et al.* [2013] permet de transformer du code `Erlang` et une propriété, par exemple est ce qu'une ligne de programme est accessible, en un réseau de Petri $\mathcal{N}$ et un marquage cible m_{final}. L'outil `SATABS` Donaldson *et al.* [2011] permet de faire la même chose avec des programmes *multi-threads* écrits en `C`. On résout ensuite le problème de couverture, qui nous permet de savoir si le programme respecte la propriété.

Le problème de couverture dans les réseaux de Petri est exponentiel en espace Lipton [1976]; Rackoff [1978]. Toutefois c'est une complexité dans le pire des cas. Ce qui nous intéresse c'est de résoudre le problème de couverture dans des cas réels, par exemple avec les réseaux de Petri générés par les outils `Soter` D'Osualdo *et al.* [2013] et `SATABS` Donaldson *et al.* [2011]. Certaines méthodes, par exemple *Expand, Enlarge and Check* (EEC) Geeraerts *et al.* [2006], et *Incremental, Inductive Coverability* (IC3) Bradley [2011], permettent de vérifier le problème de couverture dans les réseaux de Petri. Il existe des outils spécialement conçus pour résoudre le problème de couverture dans les réseaux de Petri, par exemple `Petrinizer` Esparza *et al.* [2014], qui utilise les trappes, qui sont des propriétés structurelles sur les réseaux de Petri Desel et Esparza [2005], `QCover` qui utilise la sémantique continue des réseaux de Petri Blondin *et al.* [2016] et `BFC` qui fait de l'élargissement en arrière Kaiser *et al.* [2014]. Ces outils ne sont pas encore suffisants pour résoudre les études de cas réalisées avec les outils `SATABS` et `Soter`.

3. Systèmes de canaux à pertes (LCS)

On peut utiliser les systèmes de canaux à pertes (LCS) pour modéliser des systèmes distribués. Un LCS est un ensemble fini d'automates qui s'échangent des messages via des canaux de type file FIFO à pertes. La Figure 2 représente un exemple de protocole de connexion-déconnexion. C'est un exemple très simple qui ne correspond pas à un vrai protocole. On verra au Chapitre 1 l'exemple du protocole du bit alterné (ABP) qui est plus intéressant à analyser. Un client, à gauche sur la figure, peut demander une connexion au serveur puis fermer cette connexion. Le serveur, à droite sur la figure, peut décider de déconnecter le client. Les opérations de communication sont les étiquettes des transitions, par exemple $1!Ouvrir$ pour l'envoie du message $Ouvrir$ dans le canal 1 et $2?Déconnecter$ pour la réception du message $Déconnecter$ dans le canal 2. L'alphabet de message est $M = \{Ouvrir, Fermer, Déconnecter\}$. Un état du système, qui est appelé une configuration, est de la forme $(q \times q', w_1, w_2)$ avec q un état du premier automate, q' un état du second automate, et w_1 et w_2 deux mots de M qui représentent le contenu des canaux 1 et 2. La sémantique d'un LCS est la suivante. Le système commence avec les deux automates dans leurs états initiaux et les deux canaux vides. Le premier automate peut émettre un message en allant de q_0 à q_1 avec l'opération $1!Ouvrir$. Le canal 1 contiendra le message $Ouvrir$. Le second automate peut recevoir le message $Ouvrir$ en se déplaçant de r_0 à r_1. À tout moment un message peut être perdu dans un des canaux sans qu'aucun automate ne change d'état.

La propriété qu'on peut vouloir vérifier est que le serveur ne peut pas être dans l'état r_0 s'il y a un message $Fermer$ dans le canal 1. Cela revient à dire qu'on ne veut pas aller dans une configuration plus grande que $c_{final} = (q_0 \times r_0, Fermer, \varepsilon)$ ou $c'_{final} = (q_1 \times r_0, Fermer, \varepsilon)$. Pour deux configurations c et c', on dit que c' est plus grande que c, noté $c' \geq c$, si c est obtenue à partir de c' en supprimant des messages dans les canaux. La propriété n'est pas vérifiée par le LCS. Un contre-exemple est donné dans la Figure 3.

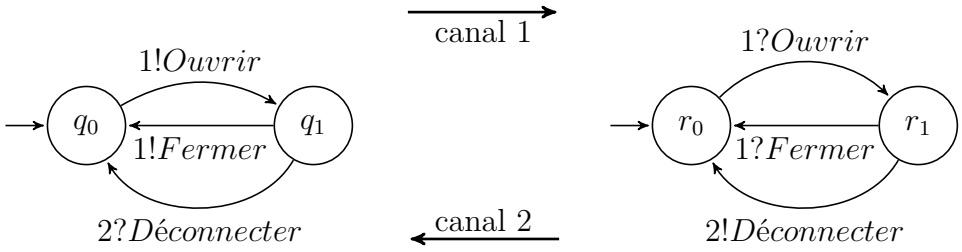

FIGURE 2 – Système de canaux à pertes (LCS) pour le protocole connexion-déconnexion

Les LCS ne sont pas toujours le modèle le plus adapté pour représenter certains systèmes distribués. Ils sont surtout utiles pour représenter des systèmes qui utilisent des communications où l'ordre de réception des messages est le même que celui des émissions. Une contrainte des LCS est l'apparition

$$
\begin{gathered}
(q_0 \times r_0, \varepsilon, \varepsilon) \\
\downarrow 1!Ouvrir \\
(q_1 \times r_0, Ouvrir, \varepsilon) \\
\downarrow 1?Ouvrir \\
(q_1 \times r_1, \varepsilon, \varepsilon) \\
\downarrow 1!Fermer \\
(q_0 \times r_1, Fermer, \varepsilon) \\
\downarrow 2!Déconnecter \\
(q_0 \times r_0, Fermer, Déconnecter) \geq c_{final}
\end{gathered}
$$

FIGURE 3 – La configuration $c_{final} = (q_0 \times r_0, Fermer, \varepsilon)$ est couvrable dans le LCS de la Figure 2.

de pertes dans les canaux, mais utiliser un LCS pour modéliser un système qui n'a pas de perte peut être un avantage. On peut en effet voir un système avec pertes comme une sur-approximation d'un même système sans perte. Si un marquage cible n'est pas couvrable avec les pertes, il ne sera pas non plus couvrable sans perte. Or, le problème de couverture est décidable dans les systèmes de canaux à pertes Abdulla et Jonsson [1996], mais indécidable dans les systèmes à canaux sans perte Brand et Zafiropulo [1983].

Le problème de couverture dans les LCS est hyper-Ackermanien Chambart et Schnoebelen [2008] mais, comme pour les réseaux de Petri, c'est une complexité dans le pire des cas et cela ne nous empêche pas de chercher à le résoudre sur des instances non triviales. La méthode *Expand, Enlarge and Check* (EEC) Geeraerts *et al.* [2006] permet de résoudre le problème de couverture entre autres dans les LCS. L'outil `TReX` Annichini *et al.* [2001] permet d'analyser directement les LCS. Les outils `LASH` Boigelot et Godefroid [1999] et `McScM` Heußner *et al.* [2012] ont été créés pour les systèmes sans perte mais on peut les utiliser pour analyser des LCS en ajoutant explicitement des transitions pour simuler les pertes.

4. Contributions de la thèse

Notre objectif est de résoudre en pratique le problème de couverture dans les réseaux de Petri et les LCS sur des exemples réels. Il existe déjà des outils qui permettent de le faire, mais pas sur toutes les instances générées par les outils comme `SATABS` et `Soter`. Nous présentons notre méthode générale dans un cadre qui englobe les réseaux de Petri et les LCS. Notre méthode générale fonctionne, sous certaines conditions, pour tous les systèmes de transitions bien structurés (WSTS) Finkel [1987]; Abdulla *et al.* [2000]; Finkel et Schnoebelen [2001]. Les réseaux de Petri et les LCS sont tous les deux des WSTS qui remplissent ces conditions d'effectivité.

De nombreuses techniques ont été développés pour résoudre le problème de couverture dans les WSTS en général ou dans les réseaux de Petri ou les

LCS en particulier comme nous l'avons déjà vu. On verra en détail dans les différentes parties un état de l'art de ces techniques.

Notre méthode générale est basée sur l'analyse en arrière Finkel [1987]; Abdulla *et al.* [2000]; Finkel et Schnoebelen [2001]. Cette analyse calcule itérativement des ensembles $A_0, A_1, \ldots$ représentants les états qui peuvent couvrir l'état cible s_{final} en au plus k étapes. Cela permet de décider du problème de couverture mais la taille des A_i peut être très grande et le calcul peut ne pas terminer dans un temps raisonnable. La modification que nous proposons consiste à utiliser un invariant de couverture I, c'est-à-dire un ensemble qui contient tous les états couvrables, pour réduire la taille des ensembles $A_0, A_1, \ldots$ en des ensembles $B_0, B_1, \ldots$ qui peuvent aussi décider du problème de couverture. On appelle cette méthode l'analyse en arrière avec élagage.

Cette méthode est une généralisation de la méthode utilisée par Blondin *et al.* [2016]. Nous l'avons d'abord étudiée pour les réseaux de Petri avec des invariants clos par le bas dans Geffroy *et al.* [2016a]. Puis, pour résoudre le problème de couverture dans les LCS, nous l'avons généralisée aux WSTS dans Geffroy *et al.* [2017b]. Ensuite, dans la version journal de Geffroy *et al.* [2016a], nous avons fait, pour les réseaux de Petri, la généralisation à tous les invariants dits *de couverture*, c'est-à-dire qui contiennent l'ensemble des états couvrables. Cette version journal est en cours de publication. Nous présentons dans cette thèse la double généralisation, c'est-à-dire d'une part pour tous les WSTS et d'autre part pour tous les invariants de couverture, et pas seulement pour les invariants clos par le bas.

Les invariants de couverture sont à adapter en fonction du système de transitions étudié, les réseaux de Petri ou les LCS. Pour les réseaux de Petri on en considère quatre : l'inéquation d'états, les trappes combinées à l'inéquation d'état, la sémantique continue et l'analyse des signes. L'inéquation d'état est une technique classique qui, en quelque sorte, relaxe la condition d'atteindre un marquage via des marquages positifs. L'analyse des signes consiste à trouver des places qui restent constamment vides depuis les marquages initiaux. En pratique, on ne l'utilise pas comme un invariant mais on l'utilise comme un précalcul en éliminant ces places qui ne peuvent pas contenir des jetons. Pour les LCS, on a introduit trois nouveaux invariants : une adaptation pour les LCS de l'inéquation d'état et deux invariants qui capturent l'ordre dans lequel les messages peuvent être émis, les expressions régulières simples et compactes, ou en anglais CSRE pour *compact simple regular expressions*, et les flots ordonnés de messages, ou en anglais MOF pour *message ordering flows*. Les *CSRE* sont plus précis que les *MOF* mais en pratique le calcul des *CSRE* peut être trop long et les *MOF* élaguent presque autant d'états que les *CSRE*.

L'analyse en arrière avec élagage accélère le calcul en réduisant les ensembles $A_0, A_1, \ldots$ utilisés par l'analyse classique. Nous introduisons dans cette thèse deux heuristiques, qui peuvent accélérer l'analyse en arrière avec élagage. Ce sont des résultats nouveaux. On a d'abord l'heuristique d'exploration qui

change l'ordre dans lequel les états sont explorés en arrière. Elle est basée sur une mesure de priorité entre les états. La seconde est appelée heuristique parallèle. L'idée vient du problème que rencontre l'invariant *CSRE* pour les LCS. En pratique, cet invariant a besoin de beaucoup trop de temps pour être calculé. La modification qu'effectue l'heuristique parallèle est de commencer en parallèle l'analyse en arrière classique et le précalcul d'un invariant. Une fois que le précalcul a terminé, on utilise l'invariant dans l'analyse en arrière. On montre que sous certaines conditions, utiliser un invariant après plusieurs itérations en arrière est aussi précis qu'utiliser le même invariant dès le début.

On introduit un précalcul supplémentaire pour les réseaux de Petri, avec les places à oméga. Une *place à oméga* est une place p telle que si un marquage m_{final} est couvrable alors tout marquage m qui a le même nombre de jetons que m_{final} dans les places $p' \neq p$ est aussi couvrable. Calculer l'ensemble des places à oméga d'un réseau est au moins aussi complexe que le problème de couverture. En pratique un calcul similaire à l'analyse des signes permet de trouver certaines places à oméga. Même si ces places sont peu nombreuses, les expérimentations montrent que les retirer du réseau permet d'accélérer le calcul.

Le but de cette thèse est de résoudre efficacement le problème de couverture sur des exemples réels. Nous avons donc logiquement cherché à implémenter l'analyse en arrière avec élagage pour les LCS et les réseaux de Petri ainsi que le calcul des invariants dont nous avons parlé. Cela a donné naissance à deux outils, `ICover` pour les réseaux de Petri et `BML` pour les LCS. Les deux sont disponibles en téléchargement Geffroy *et al.* [2016b, 2017a]. Le premier est en fait un *fork* de `QCover` Blondin *et al.* [2016]. Il est codé en `Python`. Le second est un outil entièrement nouveau codé en `OCaml`. Même si `BML` a été créé dans le but de tester nos trois invariants pour les LCS, son architecture est générale et peut être étendue pour résoudre le problème de couverture dans d'autres WSTS et avec d'autres invariants. Ces deux outils sont parmi les plus efficaces pour décider du problème de couverture.

5. Plan de la thèse

La thèse est composée de trois parties.

La première partie présente les systèmes de transitions bien structurés (WSTS). Le Chapitre 1 les définit et introduit un certain nombre de définitions et notations utiles pour toute la thèse. Un état de l'art du problème de couverture pour les WSTS est fourni. L'analyse en arrière classique est présentée en détail. Le Chapitre 2 présente l'analyse en arrière avec élagage paramétrée avec des invariants de couverture. On présente aussi l'heuristique d'exploration et l'heuristique parallèle.

La seconde partie est consacrée aux réseaux de Petri. On les introduit dans le Chapitre 3 et on présente un état de l'art du problème de couverture pour les

réseaux de Petri. On y présente notamment les trois invariants suivants : l'inéquation d'états, les trappes combinées à l'inéquation d'état et la sémantique continue. Le Chapitre 4 présente un nouvel invariant, l'analyse des signes. On y fait ensuite une comparaison des différents invariants. On montre que l'analyse des signes combinée à l'inéquation d'état est proche de la sémantique continue. Ce chapitre présente aussi une nouvelle technique, les places à oméga. Le Chapitre 5 présente notre outil `ICover` et nos expérimentations.

La troisième partie étudie les systèmes de canaux à pertes (LCS). Ils sont introduits dans le Chapitre 6 et on présente un état de l'art du problème de couverture pour les LCS. Le Chapitre 7 présente nos trois invariants, l'adaptation de l'inéquation d'état, les *CSRE* et les *MOF*. Le Chapitre 8 présente notre outil `BML` et nos résultats expérimentaux.

Le manuscrit terminera par une conclusion qui fera le bilan des résultats et présentera des perspectives pour aller plus loin dans l'efficacité des outils de vérification du problème de couverture.

Première partie

Systèmes de transitions bien structurés

Chapitre 1

Problème de couverture dans les systèmes de transitions bien structurés

Dans ce chapitre nous introduisons les systèmes de transitions bien structurés Finkel [1987]; Abdulla *et al.* [2000]; Finkel et Schnoebelen [2001], ou en anglais WSTS pour *Well-Structured Transition Systems*. Ils forment un cadre général pour l'étude de nombreux systèmes, dont les réseaux de Petri et les systèmes de canaux à pertes qui seront les sujets d'étude des parties deux et trois. Le problème qui nous intéresse est le problème de couverture. De nombreuses propriétés de sûreté peuvent être résolues avec ce problème. Comme nous l'avons vu dans l'introduction, l'idée est de modéliser des spécifications d'un système réel dans un système de transitions abstrait tel que les réseaux de Petri ou les systèmes de canaux à pertes. La propriété de sûreté devient une question de couverture. Ensuite il faut résoudre ce problème pour vérifier la sûreté du système réel. Ce problème est décidable avec des conditions d'effectivité naturelles sur le système de transitions Abdulla *et al.* [2000]; Finkel et Schnoebelen [2001] mais la complexité théorique est en général très élevée. Les systèmes tirés du monde réel ne faisant pas partie des pires cas dont découle cette complexité théorique, cela ne doit pas nous décourager d'essayer de résoudre rapidement *en pratique* ce problème. C'est ce que nous nous efforçons de faire dans cette thèse.

On commencera par donner la définition des beaux préordres, ou en anglais wqo pour *well quasi-orders*, dont ces systèmes sont munis. Nous présenterons ensuite de manière informelle les réseaux de Petri et les systèmes de canaux à pertes. Nous présenterons ensuite le problème de couverture dans les systèmes de transitions bien structurés. Pour finir nous donnerons des méthodes pour résoudre le problème de couverture dans les WSTS. D'abord en présentant en détail une méthode classique, l'algorithme de couverture en arrière, qui sera améliorée dans le prochain chapitre et ensuite en présentant un état de l'art

d'autres méthodes utilisées pour les WSTS.

1.1 Préliminaires

Dans cette section nous allons définir les notions de base sur les beaux préordres et les systèmes de transitions bien structurés (WSTS). Nous donnerons aussi quelques propriétés qui nous serviront tout au long de cette thèse.

1.1.1 Préordres (wqo)

Un *préordre* sur un ensemble S est une relation binaire $\leq$ sur S qui est transitive et réflexive. Pour un sous-ensemble $X \subseteq S$, on note $\uparrow X$ et $\downarrow X$ respectivement sa *clôture par le haut* et sa *clôture par le bas*. Ces ensembles sont définis comme suit :

$$\begin{aligned} \uparrow X &= \{s \in S \mid \exists x \in X : x \leq s\} \\ \downarrow X &= \{s \in S \mid \exists x \in X : s \leq x\} \end{aligned}$$

Un sous-ensemble $X \subseteq S$ est dit *clos par le haut* quand $X = \uparrow X$ et *clos par le bas* quand $X = \downarrow X$. Une *base* d'un clos par le haut U est un ensemble *fini* $B \subseteq U$ tel que $\uparrow B = U$. Une base *minimale* d'un clos par le haut U est une base B de U telle qu'il n'existe pas d'ensemble $B' \subsetneq B$ tel que $\uparrow B' = \uparrow B$. C'est-à-dire que si on retire un élément s à B, le nouvelle ensemble $B \backslash \{s\}$, ne représente plus le clos par le haut U car $\uparrow(B \backslash \{s\}) \neq U$. Tous les éléments de cette base sont incomparables deux à deux.

Pour simplifier les notations, nous noterons $\uparrow s$ à la place de $\uparrow\{s\}$.

Étant donné un préordre $\leq$ sur un ensemble S, pour un sous-ensemble $X \subseteq S$, on appelle *ensemble des éléments minimaux* de X l'ensemble qui contient tous les éléments $x \in X$ tel que pour tout élément $y \in S$ si $y \leq x$ alors $x \leq y$. On note cet ensemble $\mathrm{Min}(X)$. Un préordre $\leq$ sur S induit une relation d'équivalence $\approx$ sur S tel que pour deux états $x, y \in S$, $x \approx y$ si $x \leq y$ et $y \leq x$. On appelle *classe d'équivalence* un ensemble $X \subseteq S$ tels que $x \approx y$ pour tout $x, y \in X$ et qui pour tous $x \in X$ et $y \in S$, si $x \approx y$ alors $y \in X$.

Pour trouver une base minimale B à un ensemble U, on peut s'intéresser aux éléments minimaux de U, $\mathrm{Min}(U)$. Toutefois il y a trois problèmes potentiels avec $\mathrm{Min}(U)$ qui font que ce n'est pas une base minimale de U : on peut avoir que $\uparrow\mathrm{Min}(U) \neq U$ (par exemple avec les entiers $\mathbb{Z}$, $\mathrm{Min}(X) = \emptyset$ pour tout ensemble $X \subseteq Z$), l'ensemble $\mathrm{Min}(U)$ peut ne pas être une base minimale et il peut ne pas être fini. Les deux exemples illustrent respectivement les deux derniers cas.

Exemple 1.1.1. *Nous donnons un exemple d'un préordre $\leq$ sur un ensemble S tel que l'ensemble des minimaux n'est pas une base minimale. On fixe un alphabet $\Sigma = \{a, b\}$ et un ensemble d'états $S = \Sigma^*$. Pour deux mots $m_1, m_2 \in S$,*

$m_1 \leq m_2$ si la taille de m_1 est plus petite que la taille de m_2. La relation $\leq$ est un préordre : elle est réflexive et transitive. Pour $X \subseteq S$ l'ensemble des mots de longueurs au moins 2, l'ensemble des éléments minimaux est $M = \mathrm{Min}(X) = \{ab, aa, bb, ba\}$. On remarque que $aa \leq ab \leq bb \leq ba \leq aa$. L'ensemble M est une base de X mais elle n'est pas minimale. Le singleton $\{aa\}$ ainsi que tous les singletons contenant un mot de taille 2, sont des bases minimales.

Exemple 1.1.2. *Pour $S = \mathbb{N}$ et la relation R sur S tel que pour tout $x, y \in S$, xRy soit vraie. On remarque qu'il n'existe que deux clos par le haut dans S : l'ensemble $\mathbb{N}$ en entier et l'ensemble vide. Tout sous-ensemble fini non vide X de S est une base de S. L'ensemble vide est une base de lui même. Il existe donc une infinité de bases non comparables par l'inclusion. Toutefois il existe bien une base pour tout clos par le haut.*

Un *ordre partiel* $\leq$ sur un ensemble S est un préordre qui est aussi antisymétrique ($\forall x, y \in S, x \leq y \wedge y \leq x \Rightarrow x = y$). Lorsque $\leq$ est un ordre partiel sur un ensemble S, l'ensemble des éléments minimaux sont incomparables.

1.1.2 Beaux préordres

Un *beau préordre*, ou en anglais wqo pour *well quasi-order*, sur un ensemble S est un préordre $\leq$ sur S tel que, pour toute suite infinie $x_0, x_1, \ldots$ d'éléments de S, il existe deux indices i et j tels que $i < j$ et $x_i \leq x_j$. On a donc que tout wqo est bien fondé : il n'admet pas de suite infinie strictement décroissante.

Proposition 1.1.3 (Higman [1952]; Finkel et Schnoebelen [2001])**.** *Pour un préordre $\leq$ sur un ensemble S, les propositions suivantes sont équivalentes*

1. *$\leq$ est un beau préordre sur S,*
2. *tout sous-ensemble de S clos par le haut admet une base, et*
3. *toute suite infinie croissante $U_0 \subseteq U_1 \subseteq \cdots$ de sous-ensembles clos par le haut de S est stationnaire (il existe $k \in \mathbb{N}$ tel que $U_k = U_{k+1} = U_{k+2} = \cdots$)*

Démonstration. $(1 \to 2)$ Supposons qu'il existe un ensemble clos par le haut $U \subseteq S$ qui n'admet pas de base. On construit une suite d'éléments $x_0, x_1, \ldots$ de U tel que $x_0 \in U$, pour tout $i \geq 1$, $x_i \in U \backslash \uparrow\{x_0, \ldots, x_{i-1}\}$ avec x_i et x_j incomparables pour tout $j < i$. Si cette suite ne pouvait pas être infinie, alors l'ensemble $\{x_0, \ldots, x_n\}$, avec x_n le dernier élément de la suite, serait une base de U. Donc cette suite est infinie et donc $\leq$ n'est pas un wqo, ce qui est une contradiction.

$(2 \to 3)$ Supposons que tout sous-ensemble de S clos par le haut admet une base. Soit une suite infinie croissante $U_0 \subseteq U_1 \subseteq \cdots$ de sous-ensembles clos par le haut de S. L'ensemble $\bigcup_k U_k$ est un ensemble clos par le haut. Il admet

donc une base B. Soit $b \in B$. Il appartient à un ensemble U_k pour $k \geq 0$. Comme la suite est croissante, il appartient à tout $U_{k'}$ avec $k' \geq k$. Comme B est fini, il existe $h \in \mathbb{N}$ tel que $U_h = U_{h+1}$. La suite est stationnaire.

$(3 \to 1)$ Supposons que toute suite infinie croissante $U_0 \subseteq U_1 \subseteq \cdots$ de sous-ensembles clos par le haut de S soit stationnaire. Soit une suite d'éléments $x_0, x_1, \ldots$ dans S. La suite $(\bigcup_{i \leq k} \uparrow x_i)$ est une suite croissante infinie d'ensemble clos par le haut de S, elle est donc stationnaire. Il existe $k \geq 0$ tel que $\bigcup_{i \leq k} \uparrow x_i = \bigcup_{i \leq k+1} \uparrow x_i$. On a donc $\uparrow x_{k+1} \subseteq \bigcup_{i \leq k} \uparrow x_i$ et en particulier $x_{k+1} \in \bigcup_{i \leq k} \uparrow x_i$ ce qui implique qu'il existe donc par hypothèse $k' < k+1$ tel que $x_{k'} \leq x_{k+1}$. Ce qui prouve que $\leq$ est un wqo. □

1.1.3 Systèmes de transitions bien structurés (WSTS)

Nous pouvons maintenant donner la définition des systèmes de transitions bien structurés.

Définition 1.1.4 (Système de transitions)**.** *Un* système de transitions *est un tuple* $\mathcal{S} = (S, Init, \to)$ *avec* S *un ensemble d'*états*,* $Init \subseteq S$ *un ensemble d'*états initiaux *et* $\to \subseteq S \times S$ *une* relation de transition*.*

On note $\xrightarrow{*}$ la clôture réflexive et transitive de $\to$. Deux états s et s' sont tels que $s \xrightarrow{*} s'$ quand il existe des états $s_0, s_1, \ldots s_n$ tels que :

$$s = s_0 \to s_1 \to \ldots \to s_n = s'$$

Dans ce cas, la suite $s_0, s_1, \ldots s_n$ est appelée une *exécution*.

Nous allons donner maintenant plusieurs définitions classiques d'ensembles définis pour un système de transition $\mathcal{S} = (S, Init, \to)$. Pour un état $s \in S$ on note $pre_{\mathcal{S}}(s)$ (resp. $post_{\mathcal{S}}(s)$) l'ensemble des états qui vont dans (resp. sont accessibles depuis) l'état s en une étape :

$$pre_{\mathcal{S}}(s) = \{s' \in S \mid s' \to s\}$$
$$post_{\mathcal{S}}(s) = \{s' \in S \mid s \to s'\}$$

On étend la définition pour un ensemble d'états $X \subseteq S$, $pre_{\mathcal{S}}(X) = \bigcup_{x \in X} pre_{\mathcal{S}}(x)$ et $post_{\mathcal{S}}(X) = \bigcup_{x \in X} post_{\mathcal{S}}(x)$.

Pour un état s et un entier $k \geq 0$, on notera $pre_{\mathcal{S}}^{=k}(s)$ l'ensemble défini tel que $pre_{\mathcal{S}}^{=0}(s) = \{s\}$ et $pre_{\mathcal{S}}^{=k+1}(s) = pre_{\mathcal{S}}(pre_{\mathcal{S}}^{=k}(s))$. De même, pour un état s et un entier $k \geq 0$, on notera $pre_{\mathcal{S}}^{\leq k}(s)$ l'ensemble des états tel que :

$$pre_{\mathcal{S}}^{\leq k}(s) = \bigcup_{k' \leq k} pre_{\mathcal{S}}^{=k'}(s)$$

Définition 1.1.5 (Système de transitions bien structuré (WSTS) Abdulla *et al.* [2000]; Finkel et Schnoebelen [2001])**.** *Un* système de transitions bien structuré *(WSTS), est un système de transitions* $\mathcal{S} = (S, Init, \to)$ *muni d'un wqo* $\leq$ *sur l'ensemble* S *qui est* compatible *avec* $\to$ *: pour tous les états* s_1, s_2, t_1 *tels que* $s_1 \to s_2$ *et* $s_1 \leq t_1$ *il existe un état* $t_2 \in S$ *tel que* $t_1 \xrightarrow{*} t_2$ *et* $s_2 \leq t_2$*.*

1.2 Exemples

Maintenant que nous avons défini la classe des WSTS, nous proposons de présenter de manière informelle deux sous-classes, les réseaux de Petri et les systèmes de canaux à pertes (LCS), qui seront étudiées dans les parties deux et trois.

1.2.1 Réseaux de Petri

Un réseau de Petri est composé de *places*, qui peuvent contenir des *jetons*, et de *transitions* qui en fonction du nombre de jetons à certaines places peuvent ajouter ou enlever des jetons. Un état du système, qui décrit combien il y a de jetons dans chaque place, est appelé un *marquage*. La sémantique d'une transition consiste à prendre des jetons dans des places et à mettre des jetons dans d'autres places. Une transition peut prendre un jeton dans une place et le remettre immédiatement dans la même place. Une transition peut mettre plus ou moins de jetons qu'elle n'en prend.

Pour décrire de manière informelle un marquage on notera par exemple $p + 2.q$ pour dire qu'il y a un jeton dans la place p, deux jetons dans la place q et aucun autre jeton dans les autres places.

On dit qu'on *exécute* une transition t lorsqu'on applique cette transition sur un marquage m pour obtenir un nouveau marquage m'. On le note $m \xrightarrow{t} m'$.

```
    Processus 1 :
    begin
        bit_1 := false
        while true do
p_1         bit_1 := true
p_2         while bit_2 do
             | skip
            end
p_3         (* section critique *)
            bit_1 := false
        end
    end
```

```
    Processus 2 :
    begin
        bit_2 := false
        while true do
q_1         bit_2 := true
q_2         if bit_1 then
q_3             bit_2 := false
q_4             while bit_1 do
                 | skip
                end
                goto q_1
            end
q_5         (* section critique *)
            bit_2 := false
        end
    end
```

FIGURE 1.1 – Protocole d'exclusion mutuelle à 1-bit Lamport [1986]; Esparza *et al.* [2014] avec deux processus

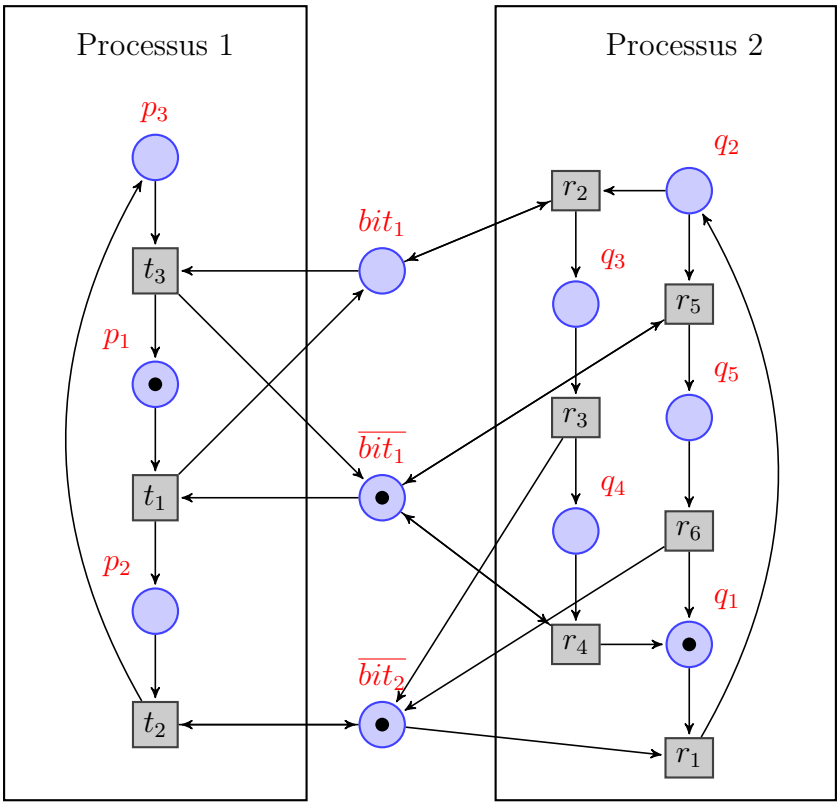

FIGURE 1.2 – Réseau de Petri modélisant le fonctionnement de l'algorithme 1 bit de Lamport

Nous allons maintenant modéliser le fonctionnement d'un système concurrent à l'aide d'un réseau de Petri. Nous reprenons l'exemple du protocole d'exclusion mutuelle à *1 bit* Lamport [1986] utilisé par Esparza *et al.* [2014] avec deux processus. Les algorithmes sont présentés Figure 1.1. Deux processus doivent partager des données : ils veulent les lire et les modifier mais ne doivent pas le faire en même temps. Ils réservent donc chacun une section pour effectuer ses opérations, qu'on appelle *section critique*. La Figure 1.2 est la modélisation du fonctionnement de ces deux processus. La propriété qu'on veut vérifier est celle de l'exclusion mutuelle : les deux processus ne doivent pas être en même temps dans la section critique. Nous verrons que cette propriété peut être modélisée dans le réseau de Petri avec le problème de couverture que nous verrons dans la prochaine section.

Les places sont représentées par des ronds et les transitions par des carrés. Un rond noir dans une place est un jeton. Le marquage $p_1 + q_1 + \overline{bit_1} + \overline{bit_2}$ est représenté sur la Figure 1.2. Une flèche d'une place vers une transition signifie que la transition prend un jeton dans cette place lorsqu'elle est exécutée. Inversement lorsqu'une flèche va d'une transition vers une place, cela signifie que la transition met un jeton dans cette place lorsqu'elle est exécu-

tée. Il est par exemple possible d'exécuter la transition t_1 depuis le marquage $p_1 + q_1 + \overline{bit_1} + \overline{bit_2}$. La transition va prendre un jeton dans p_1 et un dans $\overline{bit_1}$ et mettre un jeton dans p_2 et un dans bit_1. Les autres jetons ne bougent pas. On a donc : $p_1 + q_1 + \overline{bit_1} + \overline{bit_2} \xrightarrow{t_1} p_2 + q_1 + bit_1 + \overline{bit_2}$.

Les places p_1, p_2 et p_3 représentent respectivement les lignes du même nom du premier processus et les places q_1, q_2, q_3, q_4 et q_5 les lignes du second processus. Lorsqu'on a un jeton dans p_1 cela indique que le processus 1 est à la ligne correspondante. L'ensemble des états initiaux ne contient que l'état $s_{init} = p_1 + q_1 + \overline{bit_1} + \overline{bit_2}$. Lorsqu'il y a un jeton dans la place $\overline{bit_1}$ (resp. $\overline{bit_2}$) cela indique que la variable bit_1 (resp. bit_2) est à `false`. Inversement, lorsqu'il y a un jeton dans bit_1 cela indique que la variable bit_1 est à `true`. On remarque qu'il y aura exactement un jeton entre les places bit_1 et $\overline{bit_1}$.

$$
\begin{array}{c}
(p_1 + q_1 + \overline{bit_1} + \overline{bit_2}) \\
\downarrow t_1 \\
(p_2 + q_1 + bit_1 + \overline{bit_2}) \\
\downarrow t_2 \\
(p_3 + q_1 + bit_1 + \overline{bit_2}) \\
\downarrow r_1 \\
(p_3 + q_2 + bit_1) \\
\downarrow t_3 \\
(p_1 + q_2 + \overline{bit_1}) \\
\downarrow r_5 \\
(p_1 + q_5 + \overline{bit_1})
\end{array}
$$

FIGURE 1.3 – Exemple d'exécution dans le réseau de Petri de la Figure 1.2

La Figure 1.3 montre une exécution qui part de l'unique état initial $s_{init} = (p_1 + q_1 + \overline{bit_1} + \overline{bit_2})$ et emprunte cinq transitions.

Nous voulons maintenant convaincre le lecteur que les réseaux de Petri sont des WSTS. Cela sera démontré précisément dans la partie 2 consacrée au problème de couverture dans les réseaux de Petri. Les états d'un réseau de Petri sont les marquages. On définit l'ordre $\leq$ sur S tel que pour deux marquages $m_1, m_2 \in S$, l'inégalité $m_1 \leq m_2$ soit vraie lorsqu'on obtient m_1 en enlevant des jetons à m_2. D'après le lemme de Dickson sur des mots, la relation $\leq$ est un wqo. On montre maintenant pourquoi la relation $\leq$ est compatible avec la relation de transitions $\rightarrow$. Soit trois marquages $m_1, m_2, m_1' \in S$ et une transition t tels que $m_1 \xrightarrow{t} m_2$ et $m_1 \leq m_1'$. Par définition si $m_1 \leq m_1'$ alors on peut construire un marquage $m_\delta \in S$, tel que $m_1' = m_1 + m_\delta$. Il est possible d'exécuter la transition t depuis m_1'. On a donc $m_1' \xrightarrow{t} m_2 + m_\delta \geq m_2$. La relation $\leq$ est donc bien compatible avec $\rightarrow$. Sur l'exemple de la Figure 1.2 on peut par exemple exécuter la transition t_3 depuis le marquage $p_3 + bit_1$ comme suit $p_3 + bit_1 \xrightarrow{t_3} p_1 + \overline{bit_1}$. Cette même transition peut être exécutée depuis un

marquage m plus grand que $p_3 + bit_1$. Ce dernier marquage est forcément de la forme $m = p_3 + bit_1 + m_\delta$. Et il est possible de faire $m = p_3 + bit_1 + m_\delta \xrightarrow{t_3} p_1 + \overline{bit_1} + m_\delta \geq p_1 + \overline{bit_1}$. Les réseaux de Petri sont donc bien des WSTS.

1.2.2 Systèmes de Canaux à Pertes

Un *système de canaux à pertes*, ou en anglais LCS pour *Lossy Channel System*, est composé de *localités*, de *canaux* de type file à pertes, d'*opérations* et d'un alphabet fini de *messages*. Un LCS se veut la représentation d'un groupe de systèmes qui s'échangent des messages via des canaux de communication. La particularité est que ces canaux ne sont pas fiables : des messages peuvent être perdus. Il n'y a pas de contraintes ou de propriétés sur ces pertes. Tous les messages peuvent être perdus ou aucun, ils peuvent être perdus au début, à la fin ou au milieu d'un canal, etc. Ces systèmes doivent donc s'affranchir de ces pertes pour pouvoir effectuer leur calcul en toute sécurité.

Un état $s \in S$ d'un LCS est un tuple $s = (q, w_1, w_2, \ldots, w_d)$ où q est une localité et $w_1, w_2, \ldots, w_d$ sont des mots sur alphabet fini de messages M qui décrivent le contenu des canaux. Alors que notre modèle est composé d'un unique automate, en pratique les systèmes sont composés de multiples automates. Il est aisé de faire le produit cartésien de plusieurs automates pour en obtenir un unique. Pour deux états q et r de deux automates, on notera $q \times r$ la localité correspondant dans l'automate du produit cartésien. Il y a au plus un automate qui change d'état à chaque opération.

Nous présentons une modélisation d'un petit protocole de gestion d'erreur, le protocole du bit alterné Bartlett *et al.* [1969], noté ABP pour *alternating bit protocol* en anglais. Dans ce protocole, deux systèmes vont s'échanger des messages, mais la communication n'est pas parfaite. Les messages peuvent se perdre ou être corrompus ce qui est abstrait par une perte. Il faut donc les réenvoyer. On veut s'assurer que les messages soient tous lus et le soient dans l'ordre. Le protocole suppose que toutes les erreurs sont détectées. Le protocole ajoute à chaque message un bit pour former un nouveau message. On modélisera les messages envoyés par le premier (resp. second) automate par A_0 et A_1 (resp. B_0 et B_1).

Nous ne nous intéressons pas au contenu *utile* des messages. Quand le premier automate envoie A_0, A_0, A_1, A_1, A_0, il faut comprendre que les deux premiers contenus utiles A sont identiques auxquels le protocole a accolé un bit 0 pour former les deux messages A_0. Il y a eu un problème, il a donc fallu réémettre une seconde fois le contenu A toujours suivit du bit 0. Puis les deux suivants sont un nouveau contenu $A' \neq A$, cette fois suivi d'un 1, ce message A_1 a lui aussi eu besoin d'être renvoyé. Le dernier message est différent des deux premiers, mais il est aussi envoyé avec un bit 0 pour le démarqué du précédent. Cette fois le message n'a pas eu besoin d'être renvoyé. Même principe pour le second automate et ses messages B_0 et B_1.

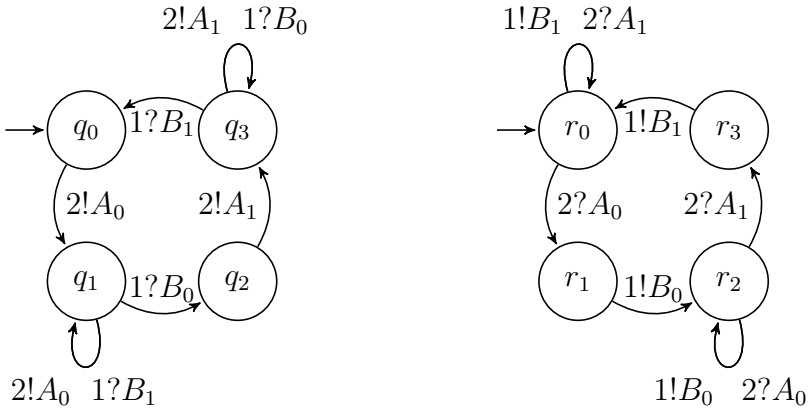

FIGURE 1.4 – Exemple de système de canaux à pertes modélisant le protocole du bit alterné Bartlett *et al.* [1969]

La Figure 1.4 représente une modélisation du protocole ABP. On laisse la représentation avec deux automates sans faire de produit cartésien. Les transitions correspondent toutefois à celle d'un LCS. Les localités, de type $q \times r$, ne sont donc pas représentées mais les états q et r des deux automates sont bien représentés comme d'habitude par des ronds. Les flèches qui représentent les transitions ont comme étiquettes des opérations. Informellement, sur l'exemple, une transition entre q_0 et q_1 dans le premier automate avec comme étiquette $2!A_0$ signifie qu'on change la localité de $q_1 \times r$ à $q_2 \times r$, avec r un état quelconque du second automate, et qu'on *envoie* le message A_0 à la fin du canal 2. Par exemple :

$$(q_0 \times r_2, B_1, A_1 \cdot A_1) \xrightarrow{2!A_0} (q_1 \times r_2, B_1, A_1 \cdot A_1 \cdot A_0)$$

Une transition entre r_0 et r_1 dans le second automate avec comme étiquette $2?A_0$ signifie qu'on change la localité de $p \times r_1$ à $p \times r_2$ avec p un état quelconque du premier automate, et qu'on *reçoit* le message A_0 qui était au début du canal 2. Par exemple :

$$(q_1 \times r_0, B_0, A_0 \cdot A_1) \xrightarrow{2?A_0} (q_2 \times r_1, B_0, A_1)$$

De plus, à tout moment un canal peut perdre un message (noté $\rightarrow_\lambda$), par exemple :

$$(q_3 \times r_0, B_0 \cdot B_1 \cdot B_1, A_1) \rightarrow_\lambda (q_3 \times r_0, B_0 \cdot B_1, A_1)$$

L'état initial est $s_{init} = (q_0 \times r_0, \varepsilon, \varepsilon)$.

La Figure 1.5 montre une exécution qui part de l'unique état initial $s_{init} = (q_0 \times r_0, \varepsilon, \varepsilon)$ et emprunte six transitions. L'unique perte survient après l'envoi du second message. On remarque que le système se trouve alors dans l'état

$$
\begin{array}{c}
(q_0 \times r_0, \varepsilon, \varepsilon) \\
\downarrow 2!A_0 \\
(q_1 \times r_0, \varepsilon, A_0) \\
\downarrow 2?A_0 \\
(q_1 \times r_1, \varepsilon, \varepsilon) \\
\downarrow 1!B_0 \\
(q_1 \times r_2, B_0, \varepsilon) \\
\downarrow_\lambda \\
(q_1 \times r_2, \varepsilon, \varepsilon) \\
\downarrow 1!B_0 \\
(q_1 \times r_2, B_0, \varepsilon) \\
\downarrow 1?B_0 \\
(q_2 \times r_2, \varepsilon, \varepsilon)
\end{array}
$$

FIGURE 1.5 – Exemple d'exécution dans le système de canaux à pertes la Figure 1.4

$(q_1 \times r_2, \varepsilon, \varepsilon)$ où les deux seules transitions qui peuvent être empruntées sont les boucles $r_2 \xrightarrow{1!B_0} r_2$ et $q_1 \xrightarrow{2!A_0} q_1$. Quand le second automate utilise $r_2 \xrightarrow{1!B_0} r_2$, il réenvoie le message B_0, composé du même contenu B, qui avait été perdu, et d'un bit 0. Si ce message n'est pas perdu, le premier automate peut donc le lire et le calcul peut continuer.

Comme les réseaux de Petri, les LCS sont des WSTS. Nous le prouverons en détail dans la partie 3 consacrée aux LCS. On définit l'ordre $\leq$ sur S tel que pour deux états $s = (q, w_1, w_2, \ldots, w_d)$ et $s' = (q', w'_1, w'_2, \ldots, w'_d)$ on a $s \leq s'$ lorsque s est obtenu à partir de s' en supprimant des messages dans les canaux.

D'après le lemme de Higman [1952], l'ordre $\leq$ est un wqo. Soit trois états $s_1 = (q_1, w_1, w_2, \ldots, w_d)$, $s_2 = (q_2, w'_1, w'_2, \ldots, w'_d)$ et $r_1 = (q'_1, w''_1, w''_2, \ldots, w''_d)$ tels que $s_1 \to s_2$ et $s_1 \leq r_1$. Comme $s_1 \leq r_1$, on a $q'_1 = q_1$. À partir de r_1 on peut donc perdre successivement des messages dans les différents canaux pour aller jusqu'à s_1 avec $\to_\lambda$. On a donc $r_1 \xrightarrow{*} s_1 \to r_2 \geq r_2$. Par exemple dans le LCS de la Figure 1.4, on peut par exemple exécuter la transition $q_0 \xrightarrow{2?A_0} q_1$ à partir de $s_1 = (q_0 \times r_0,\ B_1 \cdot B_1,\ A_0 \cdot A_0)$ pour obtenir $r_1 = (q_1 \times r_0,\ B_1 \cdot B_1,\ A_0)$. On a $s_1 \to r_1$. À partir de $s_2 = (q_0 \times r_0,\ B_1 \cdot B_1 \cdot B_0,\ A_1 \cdot A_0 \cdot A_1 \cdot A_0) \geq s_1$ on peut perdre successivement les trois messages (B_0,A_1 et encore A_1) en trop de s_2 par rapport à s_1 grâce à la transition $\to_\lambda$. On a donc $s_2 \xrightarrow{*} s_1 \to t_1 \geq t_1$. Le wqo $\leq$ est donc compatible avec $\to$. Les LCS sont des WSTS.

1.3 Problème de couverture

Le problème de couverture est un problème classique dans la vérification des propriétés de sûretés. On dit d'un état s_{final} d'un WSTS $\mathcal{S} = (S, Init, \rightarrow, \leq)$ qu'il est *couvrable* lorsqu'il existe un état s et un état initial $s_{init} \in Init$, tels que $s_{init} \xrightarrow{*} s$ et $s \geq s_{final}$. On dira que s_{final} est *accessible* lorsqu'il existe un état $s_{init} \in Init$, tel que $s_{init} \xrightarrow{*} s_{final}$.

Le problème de couverture est de décider si un état est couvrable ou non :

Problème : Couverture
Input : Un WSTS $\mathcal{S} = (S, Init, \rightarrow, \leq)$ et état cible s_{final}
Output : s'il existe $s_{init} \in Init$ et $s \in S$ tels que $s_{init} \xrightarrow{*} s \geq s_{final}$

On dira parfois que s_{final} est *couvrable depuis* s lorsque $s \xrightarrow{*} s' \geq s_{final}$ avec $s, s' \in S$.

L'ensemble des états couvrables est appelé *ensemble de couverture* et on le note $Cov_{\mathcal{S}} = {\downarrow}\{s \in S \mid \exists s_{init} \in Init, s_{init} \xrightarrow{*} s\}$.

Le problème de couverture dans les WSTS est décidable avec certaines conditions naturelles d'effectivité Abdulla *et al.* [2000]; Finkel et Schnoebelen [2001]. L'ensemble de couverture $Cov_{\mathcal{S}}$ d'un WSTS $\mathcal{S} = (S, Init, \rightarrow, \leq)$ est un clos par le bas et il peut être représenté finiment : par exemple son complémentaire est un clos par le haut qui lui peut être représenté finiment d'après la Proposition 1.1.3. Il est intéressant de noter que si cet ensemble peut être représenté et qu'en plus le problème de couverture est décidable, calculer l'ensemble de couverture n'est pas toujours possible. Par exemple, ce n'est pas possible pour les réseaux de Petri avec arcs de remise à zéro car cela permettrait de décider du problème du caractère borné (boundedness) qui est indécidable Dufourd *et al.* [1998].

En pratique, l'ensemble des états initiaux n'est pas un ensemble quelconque. Il est souvent donné sous la forme d'un clos par le bas. Dans un même WSTS, le problème de couverture avec comme ensemble d'états initiaux $Init$ ou ${\downarrow}Init$ sont équivalents. On note $Cov_{\mathcal{S}}(X)$ l'ensemble des états couvrables depuis les états de X.

Lemme 1.3.1. *Pour un WSTS $\mathcal{S} = (S, Init, \rightarrow, \leq)$, et un ensemble $X \subseteq S$, on a $Cov_{\mathcal{S}}(X) = Cov_{\mathcal{S}}({\downarrow}X)$.*

Démonstration. Parce que $X \subseteq {\downarrow}X$ et que $Cov_{\mathcal{S}}$ est une fonction croissante on a directement $Cov_{\mathcal{S}}(X) \subseteq Cov_{\mathcal{S}}({\downarrow}X)$.

Soit $x \in Cov_{\mathcal{S}}({\downarrow}X)$. Il existe $s_{init} \leq s'_{init} \in X$ et $s \in S$ tels que $s_{init} \xrightarrow{*} s \geq x$. Parce que $\leq$ est compatible avec $\rightarrow$, il existe $s' \in S$ tel que $s' \geq s$ (et donc $s' \geq x$) et $s'_{init} \xrightarrow{*} s'$. On a donc $x \in Cov_{\mathcal{S}}(X)$. Ce qui conclut $Cov_{\mathcal{S}}(X) = Cov_{\mathcal{S}}({\downarrow}X)$. □

1.3.1 Exemples d'instances du problème de couverture

On a modélisé le réseau de Petri de la Figure 1.2 pour vérifier la propriété d'exclusion mutuelle du protocole à 1-bit. On veut donc vérifier que le marquage $m_{final} = p_3 + q_5$ n'est pas couvrable car il représente le fait que les deux processus sont en même temps dans la section critique. Par exemple, s'il est possible d'exécuter plusieurs transitions depuis le marquage initial et d'atteindre $p_3 + q_5 + bit_1 \geq m_{final}$ la réponse au problème de couverture sera `vrai`. Pour information, dans cet exemple, $m_{final} = p_3 + q_5$ n'est pas couvrable.

Cela montre qu'on peut représenter le fonctionnement d'un protocole et vérifier sa propriété de sûreté avec un réseau de Petri.

Dans le LCS de la Figure 1.4 qui représente l'ABP, il y a plusieurs propriétés qu'on peut vérifier. Par exemple on peut vérifier qu'il n'est pas possible que le premier automate soit dans l'état q_1 et le second dans l'état r_3 en même temps. Cela se traduit par le problème de couvrir l'état $s_{final} = (q_1 \times r_3, \varepsilon, \varepsilon)$. On peut aussi vouloir vérifier que ce n'est pas possible d'avoir des messages A_0 puis des messages A_1 et encore des messages A_0 dans le canal 2. Cela se traduit par vérifier que les états $(q \times r, \varepsilon, A_0 \cdot A_1 \cdot A_0)$ avec $q \times r$ quelconque ne sont pas couvrables. Ces états sont en effet non-couvrables dans cet exemple.

1.4 Algorithme de couverture en arrière

Dans cette section nous rappelons un algorithme proposé Abdulla *et al.* [2000]; Finkel et Schnoebelen [2001] qui permet de décider du problème de couverture pour les WSTS qui ont certaines propriétés. Cet algorithme construit une suite $U_0 \subseteq U_1 \subseteq \cdots$ de sous ensembles de S clos par le haut qui converge vers $\uparrow pre_S^*(\uparrow s_{final})$. C'est cette suite qui sera modifiée dans le chapitre suivant pour accélérer la décision du problème de couverture.

1.4.1 Suite de clos par le haut

Nous introduisons la suite classique $U_0, U_1, \ldots$ de sous-ensemble de S définie par :

$$\begin{aligned} U_0 &= \uparrow s_{final} \\ U_{k+1} &= \uparrow pre_S(U_k) \cup U_k \end{aligned}$$

Cette suite permet de calculer $pre_S^*(\uparrow s_{final})$ qui est égale à $\uparrow pre_S^*(\uparrow s_{final})$ Finkel et Schnoebelen [2001].

Lemme 1.4.1. *Pour tout $k \geq 0$, $\uparrow pre_S^{\leq k}(\uparrow s_{final}) \subseteq U_k$.*

Démonstration. On le prouve par récurrence sur $k \geq 0$. Pour le cas de base on a $\uparrow pre_S^{\leq 0}(\uparrow s_{final}) = \uparrow\uparrow s_{final} = \uparrow s_{final} = U_0$. Supposons que pour un $k \geq 0$,

$\uparrow pre_{\mathcal{S}}^{\leq k}(\uparrow s_{final}) \subseteq U_k$. Soit $x \geq y \in \uparrow pre_{\mathcal{S}}^{\leq k+1}(\uparrow s_{final})$. Si $y \in \uparrow pre_{\mathcal{S}}^{\leq k}(\uparrow s_{final})$ on a directement $x \in U_k \subseteq U_{k+1}$ par hypothèse de récurrence. Sinon on a un état $z \in pre_{\mathcal{S}}^{\leq k}(\uparrow s_{final})$ tel que $y \to z$. Et donc $y \in pre_{\mathcal{S}}(U_k)$. Ce qui prouve que $x \in \uparrow pre_{\mathcal{S}}(U_k) \subseteq U_{k+1}$. □

Lemme 1.4.2. *La suite (U_k) est ultimement stationnaire et de plus $\bigcup_i U_i = pre_{\mathcal{S}}^*(\uparrow s_{final})$.*

Démonstration. Tout élément U_k de cette suite est un clos par le haut de S et cette suite est croissante. D'après Proposition 1.1.3 la suite est donc ultimement stationnaire. Pour la suite (U_k) il existe donc toujours un $k \geq 0$ tel que $U_k = \bigcup U_i$. D'après le Lemme 1.4.1, on a $\uparrow pre_{\mathcal{S}}^{\leq i}(\uparrow s_{final}) \subseteq U_i$ pour tout $i \geq 0$, donc $\uparrow pre_{\mathcal{S}}^*(\uparrow s_{final}) \subseteq \bigcup U_i$.

On prouve par récurrence que $U_i \subseteq pre_{\mathcal{S}}^*(\uparrow s_{final})$ pour tout $i \geq 0$. Pour le cas de base on a $U_0 = \uparrow s_{final} \subseteq pre_{\mathcal{S}}^*(\uparrow s_{final})$. Supposons que $U_i \subseteq pre_{\mathcal{S}}^*(\uparrow s_{final})$ pour un $i \geq 0$. Soit $x \in U_{i+1} = \uparrow pre_{\mathcal{S}}(U_i) \cup U_i$. Si $x \in U_i$ on a par hypothèse de récurrence $x \in pre_{\mathcal{S}}^*(\uparrow s_{final})$. Sinon on a $x \geq y \to z \in U_i$ avec y et z deux états. Par hypothèse de récurrence on a $x \in \uparrow pre_{\mathcal{S}}^*(\uparrow s_{final}) = pre_{\mathcal{S}}^*(\uparrow s_{final})$ d'après Finkel [1987]. □

Il n'y a plus qu'à tester s'il existe un état initial $s_{init} \in Init$ est dans U_k, ce qui décide du problème de couverture.

Lemme 1.4.3. *$s_{final} \in Cov_{\mathcal{S}}$ si, et seulement si, $Init \cap \bigcup_k U_k \neq \emptyset$.*

Démonstration. Si $s_{final} \in Cov_{\mathcal{S}}$ alors il existe $s_{init} \in Init$ tel que $s_{init} \xrightarrow{*} s \geq s_{final}$ pour un état s dans S. Parce que $s_{init} \xrightarrow{*} s$, il existe des états $s_0, \ldots, s_n \in S$ tels que $s_{init} = s_n$, $s_n \to s_{n-1} \cdots \to s_0$ et $s_0 \geq s_{final}$. On a $s_{init} = s_n \in U_n \subseteq \bigcup_k U_k$ et donc $Init \cap \bigcup_k U_k \neq \emptyset$.

Pour l'autre sens, supposons qu'il existe $s_{init} \in Init$ tel que $s_{init} \in \bigcup_k U_k$ et prouvons que $s_{final} \in Cov_{\mathcal{S}}$. D'après Lemme 1.4.2 on a $U_k \subseteq pre_{\mathcal{S}}^*(\uparrow s_{final})$. Donc on a $s_{init} \in pre_{\mathcal{S}}^*(\uparrow s_{final})$ et il existe $s \in \uparrow s_{final}$ tel que $s_{init} \xrightarrow{*} s \geq s_{final}$. Donc $s_{final} \in Cov_{\mathcal{S}}$ ce qui conclut la preuve. □

Toutefois, dans la plupart des WSTS comme les réseaux de Petri et les LCS, cette suite est constituée d'ensembles infinis. Nous allons donc voir maintenant comment calculer en pratique cette suite.

1.4.2 Algorithme

Nous nous intéressons maintenant à utiliser en pratique cette suite. Il va nous falloir plusieurs conditions d'effectivité. Rappelons que si l'ensemble des minimaux n'est pas forcément fini, ce qui peut se produire lorsque $\leq$ n'est pas un ordre partiel, la Proposition 1.1.3 nous dit qu'il existe pour tout un ensemble $X \subseteq S$ une base minimale de $\uparrow X$.

Définition 1.4.4. *Un WSTS $\mathcal{S} = (S, Init, \to, \leq)$ équipé de deux fonctions calculables $\underline{Cpre}$ et $\underline{Reduce}$ est dit effectif lorsque :*

- *$\leq$ est décidable*
- *pour tout ensemble fini d'états $X \subseteq S$, $\underline{Reduce}(X)$ est une base minimale de $\uparrow X$. De plus, pour deux ensembles finis d'états $X, Y \subseteq S$, si $\uparrow X = \uparrow Y$ alors $\underline{Reduce}(X) = \underline{Reduce}(Y)$*
- *pour tout état $s \in S$, $\underline{Cpre}(s)$ est une base minimale de $\uparrow pre_{\mathcal{S}}(\uparrow s)$*
- *le problème de l'appartenance d'un état $s \in S$ dans $\downarrow Init$ est décidable.*

Ces conditions permettent de travailler avec des bases minimales qui sont des ensembles finis.

On fixe un WSTS effectif $\mathcal{S} = (S, Init, \to, \leq)$ équipé de $\underline{Cpre}$ et $\underline{Reduce}$ qu'on utilisera dans ce chapitre.

Notre définition diffère de la définition classique pour deux raisons. La première est d'imposer à $\underline{Reduce}$ et $\underline{Cpre}$ de calculer des bases *minimales*, contrairement à la définition classique où ce ne sont que des bases. De plus $\underline{Reduce}$ doit être tel que pour deux ensembles finis d'états $X, Y \subseteq S$, si $\uparrow X = \uparrow Y$ alors $\underline{Reduce}(X) = \underline{Reduce}(Y)$. Cette dernière condition est nécessaire pour pouvoir considérer des exécutions de l'Algorithme 1 que l'on va présenter. Dans le cas où c'est un ordre partiel, les éléments minimaux forment une base minimale canonique, on a $\underline{Reduce}(X) = \mathrm{Min}(X)$ pour tout $X \subseteq S$. On remarque que pour un ensemble $X \subseteq S$, $\underline{Reduce}(X)$ n'est pas forcément inclus dans X.

On peut donc maintenant calculer une suite (A_k) d'ensembles finis dans S telle que $U_k = \uparrow A_k$ pour $k \geq 0$. Elle est définie par :

$$\begin{aligned} A_0 &= \{s_{final}\} \\ A_{k+1} &= \underline{Reduce}(A_k \cup \underline{Cpre}(A_k)) \end{aligned}$$

On remarque que les ensembles A_k sont tous finis.

Lemme 1.4.5. *Pour tout $k \geq 0$, A_k est une base minimale de U_k.*

Démonstration. On le prouve par induction sur $k \geq 0$. Pour le cas de base, $A_0 = \{s_{final}\}$ est une base minimale de $U_0 = \uparrow s_{final}$. Supposons que pour un k donné, A_k soit une base minimale de U_k. Par définition de $\underline{Reduce}$, A_{k+1} est une base minimale de $\uparrow(A_k \cup \underline{Cpre}(A_k))$. De plus, $\uparrow A_{k+1} = \uparrow(A_k \cup \underline{Cpre}(A_k)) = \uparrow\underline{Cpre}(A_k) \cup \uparrow A_k$ et par définition de $\underline{Cpre}$ on a $\uparrow\underline{Cpre}(A_k) = \uparrow pre_{\mathcal{S}}(\uparrow A_k)$ ce qui donne $\uparrow A_{k+1} = \uparrow pre_{\mathcal{S}}(\uparrow A_k) \cup \uparrow A_k$. Et par hypothèse de récurrence on a donc $\uparrow A_{k+1} = \uparrow pre_{\mathcal{S}}(U_k) \cup U_k = U_{k+1}$. □

Nous pouvons donc maintenant écrire l'Algorithme 1 qui va décider du problème de couverture en calculant la suite (A_k). On remarque que les lignes 2, 3 et 6 sont réalisables seulement lorsque le WSTS est effectif.

Nous notons A'_k l'ensemble A calculé par l'algorithme après $k \geq 0$ exécutions de la boucle `while`.

Algorithme 1 : $\texttt{BCA}(\mathcal{S}, s_{final})$

Input : Un WSTS $\mathcal{S} = (S, Init, \rightarrow, \leq)$ effectif avec $\leq$ décidable et un état cible $s_{final} \in S$

Output : Existe-t-il un état $s \in S$ et un état initial $s_{init} \in Init$ tels que $s_{init} \xrightarrow{*} s$ et $s \geq s_{final}$.

1 $A \leftarrow \{s_{final}\}$
2 **while** $\downarrow Init \cap A = \emptyset$ **do**
3 $\quad P \leftarrow \underline{Cpre}(A) \setminus \uparrow A$ `/* nouveaux prédécesseurs */`
4 $\quad$ **if** $P = \emptyset$ **then**
5 $\quad\quad$ **return** `False`
6 $\quad A \leftarrow \underline{Reduce}(A \cup P)$
7 **return** `True`

Lemme 1.4.6. *Après $k \geq 0$ exécutions de la boucle* `while`*, $A_k = A'_k$.*

Démonstration. On le prouve par récurrence sur $k \geq 0$. Pour le cas de base, la Ligne 1 nous assure que $A_0 = A'_0 = \{s_{final}\}$. Supposons que pour $k \geq 0$, $A_k = A'_k$ et que l'algorithme exécute au moins $k+1$ fois la boucle `while`. Alors d'après la Ligne 6 et par hypothèse d'induction on a $A'_{k+1} = \underline{Reduce}(A_k \cup (\underline{Cpre}(A_k) \setminus \uparrow A_k))$. Il est clair que $\uparrow(A_k \cup (\underline{Cpre}(A_k) \setminus \uparrow A_k)) = \uparrow(A_k \cup \underline{Cpre}(A_k))$. Et donc on a bien $A'_{k+1} = A_{k+1}$ par définition de $\underline{Reduce}$. □

Nous nous intéressons maintenant à la correction et à la terminaison de l'algorithme.

Lemme 1.4.7. *À l'itération $k \geq 1$ de la boucle* `while`*, $P_k = \emptyset$ si, et seulement si, $U_k = U_{k-1}$.*

Démonstration. Pour $k \geq 1$, $P_k = \emptyset$ est équivalent à $\underline{Cpre}(A'_{k-1}) \subseteq \uparrow A'_{k-1}$ d'après la Ligne 3. Ce qui est équivalent à $\uparrow \underline{Cpre}(A'_{k-1}) \subseteq \uparrow A'_{k-1}$ par définition de $\uparrow$. Ce qui est équivalent à $\uparrow \underline{Cpre}(A_{k-1}) \subseteq \uparrow A_{k-1}$ d'après le Lemme 1.4.6. Ce qui est équivalent à $\uparrow pre_{\mathcal{S}}(\uparrow A_{k-1}) \subseteq \uparrow A_{k-1}$ par définition de $\underline{Cpre}$ ce qui est équivalent à $\uparrow pre_{\mathcal{S}}(U_{k-1}) \subseteq U_{k-1}$ d'après le Lemme 1.4.5, ce qui est équivalent à $U_k = U_{k-1}$ par définition de (U_k).

□

Théorème 1.4.8. *L'Algorithme 1 termine sur chaque instance et est correct.*

Démonstration. Supposons que l'algorithme ne termine pas. D'après le Lemme 1.4.2 il existe $h \in \mathbb{N}$ tel que $U_h = U_{h+1}$ or d'après Lemme 1.4.5 $P_h = \emptyset$ et donc l'exécution aurait du terminer à la ligne 5 à la h-ème exécution de la boucle `while`. C'est une contradiction. L'algorithme termine donc toujours.

Pour la correction, l'algorithme retourne soit `False` à la ligne 5 ou `True` à la ligne 7 après $h \geq 0$ exécutions de la boucle `while`.

— s'il retourne `False` alors $P_h = \emptyset$ et donc d'après le Lemme 1.4.7 la séquence a stabilisé et donc $U_h = pre_S^*(\uparrow s_{final})$. Or, la condition $\downarrow Init \cap A = \emptyset$ de la boucle `while` est vraie. Donc d'après le Lemme 1.4.3 s_{final} n'est pas couvrable.
— s'il retourne `True` alors la condition $\downarrow Init \cap A_h = \emptyset$ de la boucle était fausse. On a donc $Init \cap \uparrow A_h \neq \emptyset$. D'après le Lemme 1.4.5, on a donc $Init \cap \uparrow U_h \neq \emptyset$. Et d'après le Lemme 1.4.3 on a donc $s_{final} \in Cov_S$.

Cela conclut la preuve du théorème. □

Cet algorithme est utilisable pour les réseaux de Petri et les LCS. En effet ces deux systèmes font partie de la classe des WSTS effectifs. Nous verrons cela plus en détail dans les parties 2 et 3.

1.5 Autres approches

Nous abordons maintenant d'autres techniques qui ont été utilisées pour les WSTS.

1.5.1 Expand, Enlarge and Check (EEC)

La technique appelée *Expand, Enlarge and Check* (EEC) Geeraerts *et al.* [2006], calcule, via une procédure itérative, à la fois une sur-approximation et une sous-approximation de l'ensemble des couvrables pour résoudre le problème de couverture dans les WSTS. La méthode est donc composée de deux algorithmes incomplets dans le sens de la terminaison : le premier calcule des sous-approximations et finira si, et seulement si, l'état cible est couvrable, et le second calcule des sur-approximations et finira si, et seulement si, l'état cible n'est pas couvrable.

On présentera la méthode sans rentrer dans les détails. Comme pour tous les algorithmes proposés pour résoudre le problème de couverture, la méthode EEC a besoin de conditions d'effectivité. Un WSTS qui remplit ces conditions est dit EEC-effectif. Ces conditions sont *en avant*, par exemple il faut que la fonction $post_S$ soit effectif, ce qui correspond à la fonction $\underline{Cpre}$ définie dans notre définition d'effectivité. Ces conditions ne sont pas contraignantes. Les réseaux de Petri et les LCS les remplissent Geeraerts *et al.* [2006].

Pour les réseaux de Petri la complexité de la méthode EEC est exponentielle en espace Majumdar et Wang [2013] ce qui correspond à la borne inférieure du problème de couverture.

L'Algorithme 2 est paramétré par des suites (C_i) et (L_i). Elles ont des propriétés particulières qu'on ne détaillera pas. Intuitivement la suite (C_i) représente des ensembles finis d'éléments concrets de S et la suite (L_i) des ensembles d'éléments limites. On verra en pratique comment peuvent être définies ces suites.

Algorithme 2 : $\texttt{EEC}(\mathcal{S}, s_{final})$

Input : Un WSTS $\mathcal{S} = (S, Init, \to, \leq)$ EEC-effectif avec deux suites (C_i) et (L_i) et un état cible $s_{final} \in S$

Output : Existe-t-il un état $s \in S$ et un état initial $s_{init} \in Init$ tels que $s_{init} \xrightarrow{*} s$ et $s \geq s_{final}$.

```
1 i ← 0
2 while True do
3     U ← ↓{s ∈ S | ↓Init ∋ s_0 → s_1 ... → s_n = s, ∀0 ≤ j ≤ n, s_j ∈ C_i}
       /* sous-approximation (Expand)  */
4     O ← graphe AndOr(S, C_i, L_i)
       /* sur-approximation (Enlarge) */
                                          /* Check */
5     if U ∩ ↑s_final ≠ ∅ then
6         return True
7     else if ↑s_final est évitable dans O then
8         return False
9     i ← i + 1
```

Il calcule donc à chaque itération i de sa boucle *while* une sous-approximation $U_i \subseteq Cov_{\mathcal{S}}$ et une sur-approximation $O_i \supseteq Cov_{\mathcal{S}}$ paramétrée par i. Plus i est élevé, plus elles sont précises. Pour les LCS, pour $i \geq 0$, l'ensemble C_i contient tous les états qui ont au plus i messages dans leurs canaux.

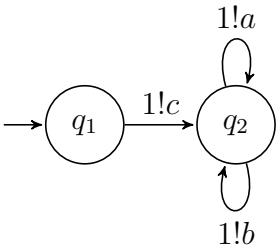

FIGURE 1.6 – Petit exemple de LCS où $(q_2, c \cdot c)$ n'est pas couvrable

On va montrer comment utiliser cette technique avec les LCS sur l'exemple de la Figure 1.6. On a donc des sous-approximations et des sur-approximations à calculer.

On peut utiliser des sous-approximations paramétrées par $i \geq 0$ en limitant le calcul aux exécutions avec des états de C_i, c'est-à-dire qui ont au plus i messages dans leurs canaux. Si une exécution $s_0 \to s_1 \ldots \to s_n \geq s_{final}$ existe avec $s_0 \in Init$, il existera un $k \geq 0$ tel que pour tout $0 \leq j \leq n$, s_j a au plus k messages dans chacun de ses canaux. Cette itération sera donc détectée à l'itération $i = k$, l'algorithme retournera donc vrai au problème de couverture.

Pour les sur-approximations, on peut utiliser des expressions régulières pour représenter le contenu des canaux. Nous ne rentrerons pas dans les détails, mais les objets utilisés sont appelés des *expressions régulières simples* (SRE) Abdulla *et al.* [1998], et serons présentés en détail dans la Section 6.2.1. Pour un $k \geq 0$ donné, on va utiliser des expressions régulières de la forme $e_0 \cdot e_1 \cdots e_k$, pour abstraire le contenu de chaque canal, avec pour tout $0 \leq i \leq k$, e_i une expression régulière de la forme $(a + \varepsilon)$ avec $a \in M$ ou de la forme $(a_1 + a_2 + \ldots + a_n)^*$ avec $a_1, a_2, \ldots a_n \in M$. Ce qui nous donne des *états abstraits* par exemple de la forme $s = (q, (a + \varepsilon), (a + c)^* \cdot (b + \varepsilon))$, qui contient des SRE de taille 1 et 2, et qui appartient donc à O_i pour $i \geq 2$. L'état abstrait s représente tous les états (q, w_1, w_2) avec $w_1 \in (a + \varepsilon)$ et $w_2 \in (a + c)^* \cdot (b + \varepsilon)$.

On a donc un moyen de représenter des ensembles infinis, ce qui nous sera utile pour les sur-approximations. À partir de ces ensembles abstraits, les auteurs utilisent un graphe And-Or Geeraerts *et al.* [2006], composé de noeuds And et de noeuds Or. On ne va pas expliquer en détail la définition de ce graphe, juste en donner l'intuition derrière celui construit à la Figure 1.7. Un noeud And est composé de plusieurs états abstraits. Ses fils sont des noeuds Or qui correspondent à chacun de ces états abstraits. Les fils des noeuds Or sont les calculs de $post_{\mathcal{S}}$ dans l'ensemble des états abstraits. Avec la limitation sur la taille des SRE, il peut y avoir plusieurs moyens de sur-approximer un état abstrait. Par exemple à partir du noeud Or $n_{or} = (q_2, (c + \varepsilon) \cdot (a + \varepsilon))$ on doit effectuer le $post_{\mathcal{S}}$ en utilisant les règles $q_2 \xrightarrow{1!a} q_2$ et $q_2 \xrightarrow{1!b} q_2$. Si la taille limite des canaux est 2, on ne peut pas utiliser $(q_2, (c + \varepsilon) \cdot (a + \varepsilon) \cdot (b + \varepsilon))$ et $(q_2, (c + \varepsilon) \cdot (a + \varepsilon) \cdot (a + \varepsilon))$. On est limité par la taille maximale des SRE. On a deux façons de sur-approximer l'état abstrait correspondant à n_{or}. Ces deux façons donneront les deux noeuds fils de n_{or}, $n_{and-1} = \{(q_2, (c + \varepsilon) \cdot (a + b)^*)\}$ et $n_{and-2} = \{(q_2, (c + a)^* \cdot (b + \varepsilon))\}$. Ces deux noeuds sur-approximent les états qu'on peut atteindre à partir de la configuration n_{or}. On remarque par exemple que $s_{final} = (q_2, c \cdot c)$ est représenté dans le noeud n_{and-2} mais pas dans le noeud n_{and-1}. En notant γ la fonction de concrétisation qui à un état abstrait retourne l'ensemble des états concrets qu'il représente, on a que $post_{\mathcal{S}}(\gamma(n_{or})) \subseteq \gamma(n_{and-1})$ et $post_{\mathcal{S}}(\gamma(n_{or})) \subseteq \gamma(n_{and-2})$.

On peut montrer qu'un état s_{final} est non couvrable si on peut trouver un sous arbre du graphe And-Or tel que tous les fils des noeuds And sont dans le sous-arbre et tel qu'il y a exactement un fils dans le sous-arbre pour chaque noeud Or. On dit qu'on peut *éviter* les mauvais états. Informellement, lorsqu'on passe d'un noeud Or à un de ses noeuds fils And on choisit la façon de sur-approximer l'ensemble $post(\gamma(n_{or}))$. Par exemple on peut prouver que l'état $s_{final} = (q_2, c \cdot c)$ n'est pas couvrable dans le LCS de la Figure 1.6. Pour cela on va donc construire le graphe And-Or avec des SRE de taille 2. Il est représentée en partie dans la Figure 1.7. La partie droite est similaire à la partie gauche. On remarque qu'en choisissant de conserver le noeud $n_{and-1} = \{(q_2, (c+\varepsilon)\cdot(a+b)^*)\}$

mais pas le noeud $n_{and-2} = \{(q_2, (c + a)^* \cdot (b + \varepsilon))\}$, et en faisant de même dans la partie symétrique, on obtient un sous-arbre, en rouge, qui ne contient aucun noeud abstrait qui contient l'état final $(q_2, c \cdot c)$. C'est une preuve de non couverture de s_{final}.

Théorème 1.5.1 (Geeraerts *et al.* [2006]). *L'Algorithme 2 est correct et termine pour chaque instance du problème de couverture.*

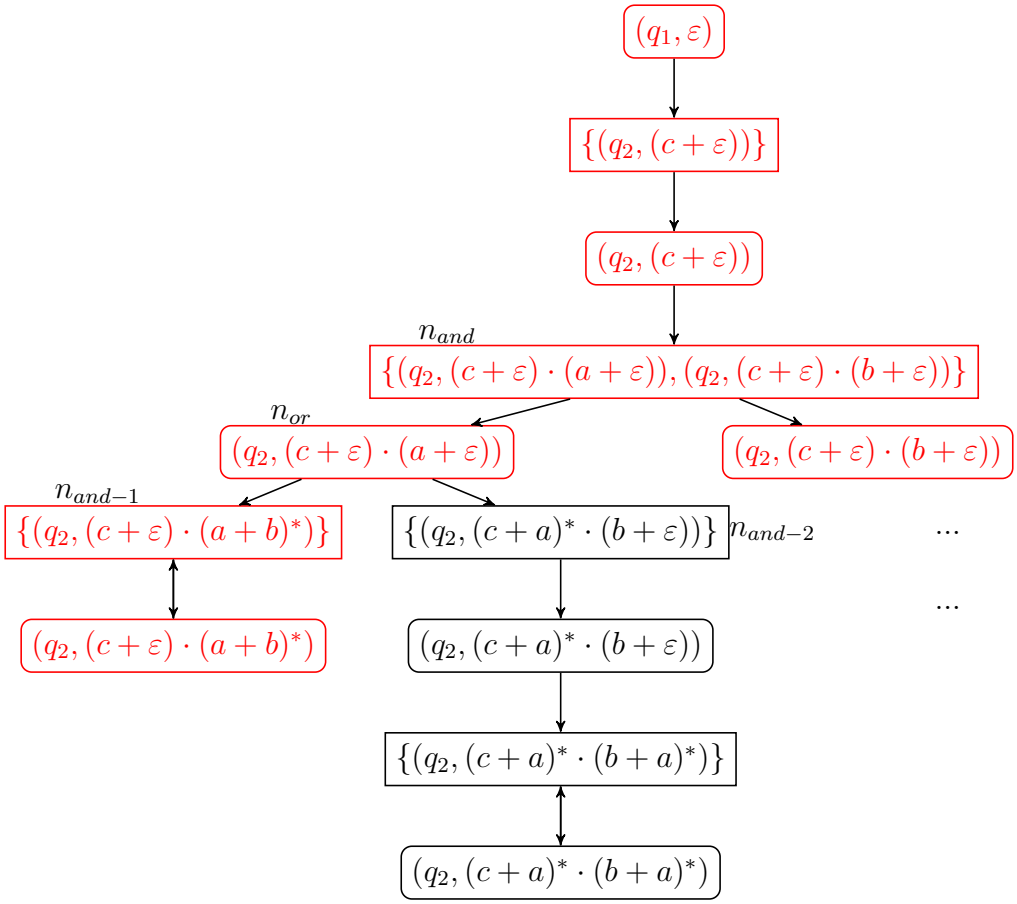

FIGURE 1.7 – Illustration de la méthode EEC. Graphe And-Or du LCS de la Figure 1.6 avec des SRE de taille 2. La partie droite n'est pas détaillée, c'est le symétrique de la partie gauche.

Cette méthode peut donc être utilisée en pratique avec les LCS, mais on peut aussi l'utiliser avec d'autres WSTS.

1.5.2 Incremental, Inductive Coverability (IC3)

La méthode *IC3* (*Incremental, Inductive Coverability*) Bradley [2011] construit itérativement une preuve de non-couverture à l'aide d'invariants in-

ductifs qui contiennent $Cov_{\mathcal{S}}$. Un *invariant* est un ensemble qui contient l'ensemble des états accessibles. Un ensemble inductif I est un ensemble qui est stable par $post_{\mathcal{S}}$: $post_{\mathcal{S}}(I) \subseteq I$. C'est une méthode générale qui propose de construire des invariants inductifs de plus en plus précis. Elle a été appliquée aux WSTS finis par le bas Kloos *et al.* [2013]. Un WSTS *fini par le bas* est un WSTS $\mathcal{S} = (S, Init, \rightarrow, \leq)$ où pour chaque état $s \in S$, $\downarrow s$ est fini. On remarque que les réseaux de Petri et les LCS sont des WSTS finis par le bas.

L'idée est de construire une suite $R_0, R_1, \dots R_N$ d'invariants clos par le bas qui contiennent $Cov_{\mathcal{S}}$ avec $N > 0$ telle que pour tout $0 \leq i < N$:

$$\begin{array}{ll} (I_1) & Init \subseteq R_i \\ (I_2) & post(R_i) \subseteq R_{i+1} \\ (I_3) & R_i \subseteq R_{i+1} \\ (I_4) & R_i \subseteq S \backslash \uparrow s_{final} \end{array}$$

L'algorithme manipule un tuple de taille variable $(R_0, R_1, \dots, R_n, \dots, Q)$ avec $R_0, R_1, \dots, R_n$ des invariants clos par le bas qui respectent les propriétés I_1, I_2, I_3 et I_4 et Q une queue de priorité qui contient des éléments de la forme (s, i) avec $s \in S$ et $i \in \mathbb{N}$. Par exemple si $a_0 \rightarrow a_1 \dots \rightarrow a_n \geq s_{final}$, avec $a_0 \in Init$, alors à la fin de l'exécution on aura $(a_i, i) \in Q$ et $a_i \in R_n$ pour tout $0 \leq i \leq n$. L'analyse va s'arrêter soit lorsque on aura trouvé un invariant inductif qui n'intersecte pas les états cibles $\uparrow s_{final}$, ce qui sera détecté si on trouve $i \geq 0$ tel que $R_i = R_{i+1}$, soit lorsque on aura trouvé un vrai contre exemple : c'est-à-dire un contre exemple qui respecte la relation $\rightarrow$.

Les règles de l'analyse sont décrites dans la Figure 1.8. $\mathbf{R}$ représente le vecteur $R_0, R_1, \dots, R_n$. On ne détaillera pas les fonctions Gen_i, à part que les éléments $b \in Gen_i(a)$ sont tels que $b \leq a$.

On présente rapidement ces règles :

— *Initialize* est la règle qui commence le calcul avec un vecteur vide.
— *Valid* et *Invalid* sont les règles qui correspondent à retourner respectivement `False` et `True`.
— *CandidateNondet* trouve un contre exemple $a \in S$ dans le dernier ensemble du vecteur, c'est-à-dire R_N, et le met dans la queue.
— *DecideNondet* trouve un tuple (a, i) de la queue et construit b, un contre exemple envers R_{i-1}, à partir de a. Il met $(b, i-1)$ dans la queue.
— *Conflict* trouve un tuple (a, i) de la queue tel que a n'est pas couvrable. La règle raffine certains invariants en utilisant cette information.
— *Induction* permet de raffiner les vecteurs lorsqu'il n'y a plus de contre exemples dans la queue et que la structure des vecteurs R_i le permet.
— *Unfold* agrandit le vecteur. Le nouvel ensemble R_N est initialisé à S.

Ces règles induisent un algorithme non déterministe qui commence avec *initialize* et finit avec *valid* ou *invalid* c'est-à-dire non-couvrable ou couvrable. Il termine toujours pour les WSTS finis clos par le bas.

[Initialize] $\dfrac{}{init \rightarrow \downarrow I|\emptyset}$ [Valid] $\dfrac{\downarrow R_i = \downarrow R_{i+1} \ pour \ i<N}{\mathbf{R}|Q \ \rightarrow \ valid}$

[CandidateNonDet] $\dfrac{a \in \downarrow R_N \cap \uparrow s_{final}}{\mathbf{R}|\emptyset \ \rightarrow \ \mathbf{R}|(a,N)}$ [Model] $\dfrac{min(Q)=(a,0)}{\mathbf{R}|Q \ \rightarrow \ invalid}$

[DecideNonDet] $\dfrac{min(Q)=(a,i) \ \ i>0 \ \ b \in pre(\uparrow a) \cap \downarrow R_{i-1} \setminus \uparrow a}{\mathbf{R}|Q \ \rightarrow \ \mathbf{R}|Q.push((b,i-1))}$

[Conflict] $\dfrac{min(Q)=(a,i) \ \ i>0 \ \ pre(\uparrow a) \cap \downarrow R_{i-1} \setminus \uparrow a = \emptyset \ \ \ b \in Gen_{i-1}(a)}{\mathbf{R}|Q \ \rightarrow \ \mathbf{R}[\downarrow R_k \ \leftarrow \ \downarrow R_k \setminus \uparrow b]_{k \in \{1,2,\ldots,i\}}|Q.popmin}$

[Induction] $\dfrac{\downarrow R_i = \Sigma \setminus \uparrow \{r_{i,1},\ldots,r_{i,m}\} \ \ b \in Gen_i(r_{i,j}) \ \ pour 1 \leq j \leq m}{\mathbf{R}|\emptyset \ \rightarrow \ \mathbf{R}[\downarrow R_k \ \leftarrow \ \downarrow R_k \setminus \uparrow b]_{k \in \{1,2,\ldots,i+1\}}}$

[Unfold] $\dfrac{\downarrow R_N \cap \uparrow s_{final} = \emptyset}{\mathbf{R}|\emptyset \ \rightarrow \ \mathbf{R} \cdot \Sigma|\emptyset}$

FIGURE 1.8 – Règles de la méthode IC3 pour résoudre la couverture dans les WSTS finis par le bas Kloos *et al.* [2013].

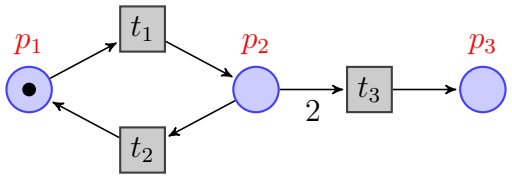

FIGURE 1.9 – Dans ce réseau de Petri simple $s_{final} = p_3$ n'est pas couvrable.

Les réseaux de Petri étant des WSTS finis par le bas, on donne un exemple d'application de ces règles sur le réseau de Petri de la Figure 1.9. Le nombre 2 sur la flèche entrante de la transition t_3 signifie qu'il faut deux jetons dans p_2 pour emprunter cette transition. Nous donnons dans la Figure 1.10 une exécution des règles d'IC3 qui permet de prouver que le marquage $m_{final} = p_3$ n'est pas couvrable.

La preuve de correction de l'algorithme s'obtient en prouvant d'abord (Lemme 1 Kloos *et al.* [2013]) que les propriétés I_1,I_2,I_3 et I_4 sont toujours vraies dans une exécution de IC3 car elles sont vraies à l'initialisation et que les règles *Unfold*, *Induction*, *Conflict*, *CandidateNondet*, et *DecideNondet* les préservent. Il faut ensuite prouver (Lemme 2 Kloos *et al.* [2013]) que si l'algorithme atteint l'état $(\mathbf{R}, Q)$, alors pour tout élément $(s,i) \in Q$, on a $s_{final} \in \downarrow post_S^*(s)$.

Théorème 1.5.2 (Théorème 1 & 2 Kloos *et al.* [2013])**.** *L'algorithme induit par la Figure 1.8 est correct.*

Démonstration. Dans le cas où l'algorithme retourne `invalid` cela implique qu'il existe $(b,i) \in Q$ avec $i = 0$. Pour avoir un tel élément avec $i = 0$ il faut que la règle *DecideNondet* ait eu comme précondition qu'il existait $(a,1) \in Q$ et

rules	**R**	Q	
$\xrightarrow{initialize}$	$(\downarrow Init)$	$\emptyset$	
$\xrightarrow{Unfold}$	$(R_0, \mathbb{N}^P)$	$\emptyset$	
$\xrightarrow{Candidate}$	$(R_0, \mathbb{N}^P)$	$\{(p_3, 1)\}$	$a = p_3$
$\xrightarrow{Conflict}$	$(R_0, \mathbb{N}^P \backslash \uparrow p_3)$	$\emptyset$	$b = p_3$
$\xrightarrow{Unfold}$	$(R_0, \mathbb{N}^P \backslash \uparrow p_3, \mathbb{N}^P)$	$\emptyset$	$\emptyset$
$\xrightarrow{Candidate}$	$(R_0, \mathbb{N}^P \backslash \uparrow p_3, \mathbb{N}^P)$	$\{(p_3, 2)\}$	$a = p_3$
$\xrightarrow{Decide}$	$(R_0, \mathbb{N}^P \backslash \uparrow p_3, \mathbb{N}^P)$	$\{(2 \cdot p_2, 1), (p_3, 2)\}$	$b = 2 \cdot p_2$
$\xrightarrow{Conflict}$	$(R_0, \mathbb{N}^P \backslash (\uparrow p_3 \cup \uparrow (2 \cdot p_2)), \mathbb{N}^P)$	$\{(p_3, 2)\}$	$b = 2 \cdot p_2$
$\xrightarrow{Conflict}$	$(R_0, R_1, \mathbb{N}^P \backslash \uparrow p_3)$	$\emptyset$	$a = p_3$
$\xrightarrow{Induction}$	$(R_0, R_1, \mathbb{N}^P \backslash (\uparrow p_3 \cup \uparrow (2 \cdot p_2)))$	$\emptyset$	$b = 2 \cdot p_2$
$\xrightarrow{Valid}$	$valid$		$R_1 = R_2$

FIGURE 1.10 – Une exécution d'IC3 sur le réseau de Petri de la Figure 1.9 avec le marquage cible $m_{final} = p_3$. ($R_0 = \downarrow Init$, $R_1 = \mathbb{N}^P \backslash (\uparrow p_3 \cup \uparrow (2 \cdot p_2))$ et $R_2 = R_1$)

$b \in (pre_{\mathcal{S}}(\uparrow a) \cap R_0) \backslash \uparrow a$. D'après le lemme 2 Kloos *et al.* [2013],$s_{final} \in post^*_{\mathcal{S}}(a)$. Donc $s_{final} \in post^*_{\mathcal{S}}(b)$ or $b \in R_0 = \downarrow Init$. Donc s_{final} est couvrable.

Dans le cas où l'algorithme retourne `valid`, il existe $i < N$ tel que $R_i = R_{i+1}$. Or d'après la propriété I_2, on a $post_{\mathcal{S}}(R_i) \subseteq R_{i+1} = R_i$. De plus $\downarrow Init \subseteq R_i$, d'après la propriété I_1. On a donc $post^*_{\mathcal{S}}(Init) \subseteq R_i$. La propriété I_4 nous prouve donc que s_{final} n'est pas couvrable. □

Les auteurs donnent dans l'Exemple 1.5.3 un WSTS qui n'est pas fini par le bas et tel que l'algorithme *IC3* ne termine pas.

Exemple 1.5.3. *Soit le WSTS $\mathcal{S} = (S, Init, \rightarrow)$, avec $S = \mathbb{N} \cup \{+\infty\}$ l'ensemble des états qui sont constitués des entiers naturels et de l'état infini $+\infty$, l'ensemble des états initiaux $Init = \{0\}$, et $\rightarrow$ la relation de transitions définie telle que $i \rightarrow i + 1$ pour $i \geq 0$ et $+\infty \rightarrow +\infty$. Avec l'ordre naturel $\leq$ sur $\mathbb{N}$ étendu sur S avec $s \leq +\infty$ pour tout état $s \in S$. Ce WSTS n'est pas fini par le bas car $\downarrow +\infty = \mathbb{N} \cup \{+\infty\}$ est infini. On remarque que l'état cible $s_{final} = +\infty$ n'est pas couvrable. L'algorithme de couverture en arrière va vite pouvoir le vérifier avec le point fixe $A_0 = \{+\infty\} = A_1$ car $\underline{Cpre}(+\infty) = \{+\infty\}$. Toutefois, l'algorithme* IC3 *ne va pas finir. Il va trouver un contre exemple candidat, $+\infty$. Le calcul ne s'arrêtera pas et construira pour tout $n > 0$, une suite $R_0, R_1, \ldots, R_n$ tel que $R_i = \downarrow \{0, 1, \ldots i\}$ pour tout $i \geq 0$.* □

C'est donc une méthode générale qui est applicable à tous les WSTS finis par le bas. Les réseaux de Petri et les LCS sont des finis par le bas.

1.5.3 Algorithme de Karp et Miller

L'algorithme de Karp et Miller Karp et Miller [1969a] qui construit un arbre représentant l'ensemble de couverture a été initialement utilisé pour les réseaux de Petri et a ensuite été généralisé à certains WSTS Finkel [1987]. L'idée est de construire un arbre représentant les états accessibles. Au début une branche représente une exécution. Lorsqu'on détecte un circuit, c'est-à-dire qu'on a une branche $m_0 \xrightarrow{t_1} m_1 \dots \xrightarrow{t_n} m_n$ telle que $m_n \geq m_i$ on comprend que les transitions $t_{i+1}, t_{i+2}, \dots t_n$ sont exécutables un nombre infini de fois à partir de m_i. Par exemple si $m_0 = p_1 + p_2$ et $m_n = p_1 + 2 \cdot p_2 + p_3$, on sait qu'on peut couvrir tous marquages $m = p_1 + x \cdot p_2 + y \cdot p_3$ avec $x, y \geq 0$, en exécutant suffisamment de fois la boucle $t_{i+1}, t_{i+2}, \dots t_n$.

Nous présenterons en détail cet algorithme pour les réseaux de Petri dans la partie 2, à la Section 3.2.1.

Chapitre 2

Algorithme de couverture en arrière avec élagage

Dans le chapitre précédent nous avons présenté le problème de couverture dans les systèmes de transitions bien structurés. L'algorithme de couverture en arrière, qui décide du problème de la couverture, a été présenté en détail dans la Section 1.4. Nous proposons dans ce chapitre d'accélérer en pratique ce calcul en réduisant la taille des bases calculées à chaque étape. Ce cadre est paramétré par des invariants, c'est-à-dire des sur-approximations de l'ensemble des états accessibles, qui doivent contenir l'ensemble des états couvrables. Les invariants sont à calculer de manière indépendante du calcul en arrière. On présentera dans la partie 2 des invariants pour les réseaux de Petri et des invariants pour les LCS dans la partie 3. Les invariants sont obtenus en général par des méthodes en avant.

L'idée de combiner une approche en arrière avec des invariants en avant n'est pas nouvelle. Elle a été décrite pour accélérer le calcul des *covering sharing trees* Delzanno *et al.* [2001] pour le problème de couverture dans les réseaux de Petri. L'invariant était l'équation d'état. La thèse de Laurent Van Begin Begin [2004] utilise aussi une approche similaire pour des variantes de réseaux de Petri comme les réseaux de Petri *auto-modifiants*. Les invariants peuvent aussi combiner l'équation d'état avec d'autres systèmes d'équations plus complets et propres aux variantes des réseaux de Petri. L'accessibilité dans les réseaux de Petri continus a été combinée avec l'algorithme de recherche en arrière Blondin *et al.* [2016] pour le problème de couverture dans les réseaux de Petri. Dans Geffroy *et al.* [2016a], nous avons étendu la méthode à tous les invariants clos par le bas, toujours pour les réseaux de Petri, mais avec des invariants différents. Dans Geffroy *et al.* [2017b] nous avons étendu la méthode, à tous les WSTS sous conditions d'effectivité. Par la même occasion, nous avons démontré que l'approche pouvait aussi être utilisée en pratique pour les LCS en utilisant des invariants spécifiques. Dans la version journal de Geffroy *et al.* [2016a], en cours de publication, nous avons généralisé aux invariants

non pas forcément clos par le bas, mais qui contiennent l'ensemble de couverture. Nous appelons ces invariants des invariants de couverture. Ce que nous présentons dans la première section de ce chapitre est la double généralisation de la méthode : elle fonctionne pour tous les WSTS avec tous les invariants de couverture, même à ceux qui ne sont pas clos par le bas.

Nous présentons aussi des heuristiques pour accélérer le calcul en pratique sans chercher à réduire la taille des bases utilisées par l'analyse en arrière. La première est appelée heuristique d'exploration, Section 2.2.1, et cherche à effectuer les calculs en arrière dans un ordre plus avantageux. Elle utilise pour cela un ordre $\leq_h$ sur les états qui définit une priorité : si deux états s_1 et s_2 sont dans une base B tels que $s_1 \leq_h s_2$ alors on calculera $\underline{Cpre}(s_2)$ avant $\underline{Cpre}(s_1)$. Elle est inspirée d'une heuristique utilisée dans l'outil `QCover` Blondin *et al.* [2016]. La seconde heuristique est appelée heuristique parallèle, et cherche à résoudre un problème qui peut arriver en pratique comme nous le verrons dans la Section 8.4 du chapitre consacré aux expérimentations sur les LCS : effectuer le précalcul d'un invariant, c'est-à-dire calculer toutes les informations qui sont nécessaires pour décider du problème de l'appartenance d'un état à cet invariant, peut prendre beaucoup de temps et éventuellement prendre plus de temps que le calcul classique en arrière sans cet invariant. L'idée est donc ne pas utiliser tout de suite les invariants mais de commencer en parallèle le calcul en arrière et le précalcul des invariants. Et une fois le précalcul d'un invariant terminé, on l'utilise pour réduire les bases du calcul en arrière. Ces heuristiques sont, à notre connaissance, nouvelles.

On commencera par définir la suite (B_k^I), paramétrée par un invariant I. C'est une modification de la suite (A_k) vue à la Section 1.4 du premier chapitre. On présentera l'algorithme qui calcule cette nouvelle suite et on effectuera les démonstrations de correction. Ensuite nous passerons aux heuristiques. Nous commencerons par présenter l'heuristique d'exploration qui change l'ordre du calcul en arrière en utilisant un ordre de priorité sur les états. Puis nous montrerons l'heuristique parallèle qui lance l'analyse en arrière et le précalcul des invariants en parallèle.

2.1 Utiliser des invariants pour accélérer le calcul

Nous allons donc reprendre la méthode vue au chapitre précédent à la Section 1.4. On vise à accélérer le calcul en pratique et pour cela on va modifier la suite (A_k), qui servait à décider du problème de couverture. Le but est d'accélérer le calcul en réduisant les ensembles qui composent cette suite. Pour cela nous utilisons des invariants. Rappelons qu'un *invariant* d'un système de transition est un ensemble qui contient tous les états accessibles. Un *invariant de couverture* I d'un WSTS $\mathcal{S}$ est un ensemble tel que $Cov_{\mathcal{S}} \subseteq I$. C'est une

sur-approximation de l'ensemble de couverture. On remarque qu'un invariant clos par le bas est un invariant de couverture par définition de $Cov_{\mathcal{S}}$.

Nous ne modifierons pas la suite (U_k) car il n'y a pas forcément de moyen simple de faire correspondre les membres de la suite (B_k^I) dans le cas où I n'est pas clos par le bas. L' Exemple 2.1.4 expliquera cela plus en détail. Nous allons donc introduire une suite (B_k^I), paramétrée par un invariant I, qui est une modification de (A_k).

Nous nous fixons un WSTS effectif $\mathcal{S} = (S, Init, \to, \leq)$ pour le reste du chapitre.

2.1.1 Modification de la suite (A_k)

Nous introduisons une suite $B_0^I \subseteq B_1^I \subseteq \cdots$ de clos par le haut de S paramétrée par un invariant I. Elle est définie par :

$$\begin{aligned} B_0^I &= \{s_{final}\} \cap I \\ B_{k+1}^I &= \underline{Reduce}(B_k^I \cup (\underline{Cpre}(B_k^I) \cap I)) \end{aligned}$$

On observe que si l'état cible s_{final} n'est pas dans l'invariant alors $B_0^I = \emptyset$ et il n'y a pas besoin de continuer le calcul, la réponse au problème de couverture est non. On remarque que pour l'invariant trivial $I = S$, la suite (B_k^I) est égale à la suite (A_k).

Une suite (B_k^I) ne permet pas forcément de décider du problème de couverture lorsque l'invariant I est quelconque. L'Exemple 2.1.1 montre le problème sur un réseau de Petri simple. Il faut qu'un invariant I soit de couverture, c'est-à-dire qu'il contienne l'ensemble des états couvrables, pour que la suite (B_k^I) permette de décider du problème de couverture.

Exemple 2.1.1. *Prenons le réseau de Petri $\mathcal{N}$ très simple qui contient une unique place p, une unique transition t telle que $F(p,t) = 1$ et $F(t,p) = 3$, et un marquage initial $s_{init} = p$. Parce que $p \xrightarrow{t} 3p \xrightarrow{t} 5p$, on remarque que le marquage cible $s_{final} = 4p$ appartient à $Cov_{\mathcal{S}}$. On étudie maintenant l'ensemble $I = \{p, 3p, 5p, \ldots\}$. C'est exactement l'ensemble des marquages accessibles de $\mathcal{N}$. Mais comme le marquage cible s_{final} n'est pas dans l'ensemble I, on a que $B_k^I = \emptyset$ pour tout $k \geq 0$. Donc (B_k^I) ne permet pas de décider du problème de couverture.*

Le lemme suivant est central dans la preuve de correction. La première inclusion nous permet de vérifier qu'on garde suffisamment d'éléments, et que donc si un état initial s_{init} n'est pas dans la suite (B_k^I) alors c'est que s_{final} n'est pas dans $Cov_{\mathcal{S}}$. La seconde nous assure que si tout état initial s_{init} est dans un élément de la suite, alors s_{init} est dans $pre_{\mathcal{S}}^*(\uparrow s_{final})$ et donc s_{final} est dans $Cov_{\mathcal{S}}$.

Lemme 2.1.2. *Si* $Cov_{\mathcal{S}} \subseteq I$ *alors*

$$\uparrow pre_{\mathcal{S}}^{\leq k}(\uparrow s_{final}) \cap Cov_{\mathcal{S}} \;\subseteq\; \uparrow B_k^I \;\subseteq\; \uparrow pre_{\mathcal{S}}^{\leq k}(\uparrow s_{final})$$

pour tout $k \in \mathbb{N}$.

Démonstration. On prouve les deux inclusions par induction sur $k \geq 0$. On commence par la seconde inclusion. Le cas de base est vrai car $\uparrow B_0^I \subseteq \uparrow s_{final} = \uparrow pre_{\mathcal{S}}^{\leq 0}(\uparrow s_{final})$. Pour l'étape d'induction, supposons que $\uparrow B_k^I \subseteq \uparrow pre_{\mathcal{S}}^{\leq k}(\uparrow s_{final})$.

On a donc

$$\begin{array}{rcll} \uparrow B_{k+1}^I & = & \uparrow(\underline{Cpre}(B_k^I) \cap I) \;\cup\; \uparrow B_k^I & \text{[définition de } \underline{\text{Reduce}}\text{]} \\ & \subseteq & \uparrow \underline{Cpre}(B_k^I) \;\cup\; \uparrow B_k^I & \\ & = & \uparrow pre_{\mathcal{S}}(\uparrow B_k^I) \;\cup\; \uparrow B_k^I & \text{[définition de Cpre]} \\ & \subseteq & \uparrow pre_{\mathcal{S}}^{\leq k+1}(\uparrow s_{final}) & [\uparrow B_k^I \;\subseteq\; pre_{\mathcal{S}}^{\leq k}(\uparrow s_{final})] \end{array}$$

On prouve maintenant la première inclusion. On considère deux cas pour le cas de base. Si $s_{final} \in I$ alors $(\uparrow pre_{\mathcal{S}}^{\leq 0}(\uparrow s_{final}) \cap Cov_{\mathcal{S}}) \subseteq \uparrow s_{final} = \uparrow B_0^I$. Sinon si $s_{final} \notin I$ alors $s_{final} \notin Cov_{\mathcal{S}}$. De plus comme $Cov_{\mathcal{S}}$ est clos par le bas, $\uparrow s_{final} \cap Cov_{\mathcal{S}}$ est vide. Donc $\uparrow pre_{\mathcal{S}}^{\leq 0}(\uparrow s_{final}) \cap Cov_{\mathcal{S}}$ est vide, et donc il est contenu dans $\uparrow B_0^I$.

Pour l'étape d'induction supposons que $(\uparrow pre_{\mathcal{S}}^{\leq k}(\uparrow s_{final}) \cap Cov_{\mathcal{S}}) \subseteq \uparrow B_k^I$. La définition de (B_k^I) et de $\underline{Reduce}$ implique que $\uparrow B_k^I \subseteq \uparrow B_{k+1}^I$. On doit donc seulement prouver que : $(\uparrow pre_{\mathcal{S}}^{=k+1}(\uparrow s_{final}) \cap Cov_{\mathcal{S}}) \subseteq \uparrow B_{k+1}^I$. Commençons par remarquer que

$$\uparrow pre_{\mathcal{S}}(X) \cap Cov_{\mathcal{S}} \;\subseteq\; \uparrow pre_{\mathcal{S}}(X \cap Cov_{\mathcal{S}})$$

pour tout sous-ensemble $X \subseteq S$. La remarque vient du fait que si $m \in Cov_{\mathcal{S}}$ et $m \to m'$ alors $m' \in Cov_{\mathcal{S}}$.

On a donc que :

$$\uparrow B_k^I \;\supseteq\; \uparrow pre_{\mathcal{S}}^{=k+1}(\uparrow s_{final}) \cap Cov_{\mathcal{S}} \tag{2.1}$$

On a donc que :

$$\begin{array}{rcll} \uparrow B_{k+1}^I & = & \uparrow(\underline{Cpre}(B_k^I) \cap I) \;\cup\; \uparrow B_k^I & \text{[définition de } \underline{\text{Reduce}}\text{]} \\ & \supseteq & \uparrow(\underline{Cpre}(B_k^I) \;\cap\; Cov_{\mathcal{S}}) & [Cov_{\mathcal{S}} \subseteq I] \\ & \supseteq & (\uparrow \underline{Cpre}(B_k^I)) \;\cap\; Cov_{\mathcal{S}} & [Cov_{\mathcal{S}} = {\downarrow} Cov_{\mathcal{S}}] \\ & = & \uparrow pre_{\mathcal{S}}(\uparrow B_k^I) \;\cap\; Cov_{\mathcal{S}} & \text{[définition de Cpre]} \\ & \supseteq & \uparrow pre_{\mathcal{S}}^{k+1}(\uparrow s_{final}) \;\cap\; Cov_{\mathcal{S}} & [\text{Équation 2.1}] \end{array}$$

Cela conclut la preuve du lemme. □

La suite (B_k^I) est, comme la suite (A_k), capable de décider du problème de couverture comme le montre le lemme suivant :

Lemme 2.1.3. $s_{final} \in Cov_{\mathcal{S}}$ *si, et seulement si,* ${\downarrow} Init \cap \bigcup_k B_k^I \neq \emptyset$.

Démonstration. On rappelle que $pre^*_{\mathcal{S}}(\uparrow s_{final}) = \bigcup_k \uparrow pre^{\leq k}_{\mathcal{S}}(\uparrow s_{final})$ Finkel et Schnoebelen [2001]. À partir du Lemme 2.1.2 on déduit que

$$pre^*_{\mathcal{S}}(\uparrow s_{final}) \cap Cov_{\mathcal{S}} \subseteq \bigcup_k \uparrow B^I_k \subseteq pre^*_{\mathcal{S}}(\uparrow s_{final})$$

On rappelle que $s_{final} \in Cov_{\mathcal{S}}$ si, et seulement si, $Init \cap pre^*_{\mathcal{S}}(\uparrow s_{final})$ est non vide. Comme $Init \subseteq Cov_{\mathcal{S}}$, on a que $s_{final} \in Cov_{\mathcal{S}}$ si, et seulement si, $Init \cap \bigcup_k \uparrow B^I_k$ est non vide. La dernière condition est équivalente à dire que $\downarrow Init \cap \bigcup_k B^I_k$ est non vide. □

Exemple 2.1.4. *Nous illustrons avec un exemple pourquoi nous n'avons pas présenté une modification de la suite (U_k) mais directement une modification de la suite (A_k). Prenons le réseau de Petri très simple constitué de deux places p_1 et p_2 d'une unique transition t qui prends deux jetons dans p_1 et mets deux jetons dans p_2. L'ensemble des états initiaux est $Init = \{p_1\}$. L'ensemble de couverture est $Cov_{\mathcal{S}} = \downarrow p_1$. L'état cible, non couvrable, est $s_{final} = \{4 \cdot p_2\}$. Pour un invariant non clos par le bas $I = Cov_{\mathcal{S}} \cup \{2 \cdot p_1 + 3 \cdot p_2\} \cup \{4 \cdot p_2\}$, on a $B^I_0 = \{s_{final}\} = B^I_1$. En effet, $\underline{Cpre}(s_{final}) = \{2 \cdot p_1 + 2 \cdot p_2\} \notin I$. Si on voulait définir une suite de clos par le haut paramétrée par I analogue à (U_k) on pourrait la définir ainsi :*

$$\begin{aligned} V^I_0 &= \uparrow(s_{final} \cap I) \\ V^I_{k+1} &= V^I_k \cup \uparrow(\uparrow pre_{\mathcal{S}}(V^I_k) \cap I) \end{aligned}$$

Toutefois on remarque que $V_0 = \uparrow s_{final}$ et $V_1 = V_0 \cup \uparrow(2 \cdot p_1 + 3 \cdot p_2)$. En effet on a $m = (2 \cdot p_1 + 3 \cdot p_2) \xrightarrow{t} (5 \cdot p_2) \geq s_{final}$ avec $m \in I$. La suite (V^I_k) ne correspond donc pas à la suite $(\uparrow B^I_k)$: elle est moins précise. Avec un invariant non clos par le bas il est plus facile de raisonner directement avec les bases. Si I est un clos par le bas, alors les éléments de la suite (B^I_k) sont les bases des éléments de la suite (V^I_k) Geffroy et al. [2017b]. □

2.1.2 Algorithme

Nous pouvons donc modifier l'Algorithme 1 en un nouvel Algorithme 3 paramétré par un invariant I. Nous allons prouver que cet algorithme calcule la suite (B^I_k) que l'on vient de présenter.

Pour $k \geq 1$, on introduit les ensembles ${B'}^I_k$,N^I_k et P^I_k avec ${B'}^I_0 = B^I_0$:

$$\begin{aligned} N^I_k &= \underline{Cpre}(B^I_k) \setminus \uparrow B^I_k \\ P^I_k &= N^I_k \cap I \\ {B'}^I_{k+1} &= \underline{Reduce}({B'}^I_k \cup P^I_k) \end{aligned}$$

On observe que les ensembles N, P, B calculés par l'algorithme à la k-ième exécution de la boucle `while` sont les ensembles N^I_k, P^I_k, ${B'}^I_k$. Dans le lemme

Algorithme 3 : `BCwP`$(\mathcal{S}, s_{final}, I)$

Input : Un WSTS $\mathcal{S} = (S, Init, \to, \leq)$ effectif équipé avec des fonctions $\underline{Cpre}$ et $\underline{Reduce}$, un état cible $s_{final} \in S$ et un invariant de couverture I pour $\mathcal{S}$

Output : Existe-t-il un état $s \in S$ tel que $s_{init} \xrightarrow{*} s$ et $s \geq s_{final}$.

```
1  if s_final ∈ I then
2  |  B ← {s_final}
3  else
4  |  B ← ∅
5  while ↓Init ∩ B = ∅ do
6  |  N ← Cpre(B) \ ↑B                    /* nouveaux prédécesseurs */
7  |  P ← N ∩ I                  /* élague les états non couvrables */
8  |  if P = ∅ then
9  |  |  return False
10 |  B ← Reduce(B ∪ P)
11 return True
```

suivant on prouve que même si les définitions diffèrent légèrement, les ensembles B_k^I et ${B'}_k^I$ sont toujours identiques après $k \geq 0$ exécutions de la boucle `while`.

Lemme 2.1.5. *Après $k \geq 0$ exécutions de la boucle `while`, $B_k^I = {B'}_k^I$.*

Démonstration. On le prouve par induction sur $k \geq 0$. Le cas de base s'obtient simplement par définition de ${B'}_0^I$. Pour l'étape d'induction supposons que $B_k^I = {B'}_k^I$ pour $k \geq 0$ et que l'algorithme effectue au moins $k+1$ exécutions de la boucle `while`.

On prouve que

$$B \overset{\text{déf}}{=} \uparrow(B_k^I \cup (\underline{Cpre}(B_k^I) \cap I)) = \uparrow(B_k^I \cup ((\underline{Cpre}(B_k^I) \setminus \uparrow B_k^I) \cap I)) \overset{\text{déf}}{=} B'$$

L'inclusion $B' \subseteq B$ est directe. Pour le sens contraire, soit $x \geq y \in (B_k^I \cup (\underline{Cpre}(B_k^I) \cap I))$. Si $y \in B_k^I$ alors on a directement $x \in B'$. Sinon, on a $y \in (\underline{Cpre}(B_k^I) \cap I)$. On a deux cas : soit $y \notin \uparrow B_k^I$ et alors on a directement $y \in B_k^I \cup ((\underline{Cpre}(B_k^I) \setminus \uparrow B_k^I) \cap I)$. Soit $y \in \uparrow B_k^I$ et donc on a aussi $y \in B_k^I \cup ((\underline{Cpre}(B_k^I) \setminus \uparrow B_k^I) \cap I)$. Dans les deux cas on a bien $x \in B'$ et donc on a fini de prouver que $B = B'$.

Par définition de $\underline{Reduce}$ on a donc $B_{k+1}^I = {B'}_{k+1}^I$. Ce qui prouve l'induction. □

Lemme 2.1.6. *$\uparrow B_k^I = \uparrow B_{k+1}^I$ si, et seulement si, $P_k^I = \emptyset$.*

Démonstration. Supposons que $\uparrow B_k^I = \uparrow B_{k+1}^I$. Comme $B_{k+1}^I = \underline{Reduce}(B_k^I \cup P_k^I)$, on a $P_k^I \subseteq \uparrow B_{k+1}^I = B_k^I$. De plus, parce que $P_k^I \subseteq N_k^I$ et $N_k^I \cap \uparrow B_k^I = \emptyset$, on a que $P_k^I \cap \uparrow B_k^I = \emptyset$. On a donc que $P_k^I = \emptyset$.

Inversement, si $P_k^I = \emptyset$ alors $\uparrow B_{k+1}^I = \uparrow(B_k^I \cup P_k^I) = \uparrow B_k^I$. □

Théorème 2.1.7. *La procédure* `BCwP` *termine pour chaque entrée satisfaisant les prémisses, et est correcte.*

Démonstration. Pour la terminaison, la définition de (B_k^I) et celle de $\underline{Reduce}$ impliquent que la suite $(\uparrow B_k^I)$ est croissante. La Proposition 1.1.3 implique que $(\uparrow B_k^I)$ est stationnaire. Il existe donc $k \geq 0$, tel que $\uparrow B_k^I = \uparrow B_{k+1}^I$. Le Lemme 2.1.6 montre que $P_k^I = \emptyset$ et implique que l'algorithme ne peut donc pas exécuter la boucle `while` plus de k fois.

Maintenant prouvons la correction. Si l'algorithme retourne `False` après $k \geq 0$ exécutions de la boucle `while`, cela implique que $P_k^I = \emptyset$ alors le Lemme 2.1.6 montre que $\uparrow B_k^I = \uparrow B_{k+1}^I$. La condition de la boucle `while`, $\downarrow Init \cap B = \emptyset$, est satisfaite et donc $\downarrow Init \cap \bigcup_k B_k^I = \emptyset$. Le Lemme 2.1.3 montre donc que s_{final} n'est pas couvrable.

Si l'algorithme retourne `True` après $k \geq 0$ exécutions de la boucle `while`, alors la condition de la boucle `while`, $\downarrow Init \cap B = \emptyset$, est fausse. Donc, d'après Lemme 2.1.3, s_{final} est couvrable. □

Dans la suite, l'Algorithme 3 sera appelé `ICover`. Cette méthode permet donc de résoudre le problème de couverture dans les WSTS. Elle repose sur l'efficacité des invariants qui lui sont donnés. Des invariants seront proposés dans les parties 2 et 3, respectivement pour les réseaux de Petri et les LCS. L'efficacité des invariants, et donc de cette méthode, sera testée dans les deux parties avec des implémentations pratiques.

2.2 Heuristiques

Nous avons présenté comment réduire la taille de la suite (A_k) présentée au précédent chapitre. On verra dans les parties expérimentales, que cela permet d'accélérer le calcul. Maintenant nous présentons deux heuristiques pour accélérer en pratique le calcul sans chercher à réduire la taille des bases. L'utilité de ces deux méthodes sera étudiée dans les parties expérimentales, Section 5.5 et Section 8.4.

2.2.1 Heuristique d'exploration

La première heuristique est indépendante des invariants et consiste à changer l'ordre dans lequel les marquages seront traités. Elle est inspirée d'une heuristique utilisée dans `QCover` Blondin *et al.* [2016] pour les réseaux de Petri. L'algorithme de couverture en arrière peut être vu comme un algorithme

de parcours en largeur. Cette heuristique vise à changer l'ordre de cette exploration. L'idée repose sur une mesure d'utilité des états. Il nous faut un ordre $\leq_h$ sur les états qui soit simple à calculer. Pour deux états s_1 et s_2 tels que $s_1 \leq_h s_2$, $\underline{Cpre}(s_2)$ sera calculé avant $\underline{Cpre}(s_1)$.

Exemple 2.2.1. *L'ordre $\leq_h$ utilisé par* `QCover` *Blondin et al. [2016] dans les réseaux de Petri est donné par : pour deux marquages m_1 et m_2 on a $m_1 \leq_h m_2$ lorsque que m_1 a au moins autant de jetons, au total. Par exemple $(2 \cdot p_3 + p_4) \leq_h (p_1 + p_2)$.*

En pratique, la Ligne 6 de l'Algorithme 3, $N \leftarrow (\underline{Cpre}(B)) \setminus \uparrow B$, ne calcule pas tout l'ensemble $\underline{Cpre}(B)$. La suite $(\uparrow B_k^I)$ étant croissante, à la Ligne 10, $B \leftarrow \underline{Reduce}(B \cup P)$, des éléments de B peuvent être déjà présents. L'implémentation n'applique donc la fonction $\underline{Cpre}$ que sur les nouveaux éléments. À l'itération k c'est $\underline{Cpre}(P_k^I)$ qui est calculé. On peut voir $W = P_k \setminus \uparrow B_k^I$ comme une liste de travail. Ce sont tous les éléments qu'on va explorer en arrière. En pratique dans l'implémentation, si l'algorithme effectue k itérations de boucle, il utilise donc $k+1$ ensemble $W_0, W_1, \ldots, W_k$ avec $W_0 = \{s_{final}\}$ et $W_k = P_k \setminus \uparrow B_k^I$ pour $k \geq 1$. Le calcul $\underline{Cpre}(W_k)$ est effectué d'un coup sans rien changer au reste de l'algorithme. On propose de passer d'une vision étape par étape à une vision élément par élément. Cela permet d'utiliser une heuristique par rapport à une priorité entre les états pour pouvoir explorer les états les plus prometteurs.

On introduit donc une queue Q composé d'états de S. La queue Q est un multi-ensemble, un même état peut y apparaître plusieurs fois. On notera $set(Q)$ l'ensemble des états qui appartiennent à Q. On introduit deux opérations sur Q, *pop* et *push* telles que :

- *pop* est une fonction qui prend une queue $Q \subseteq S$ et qui retourne (Q', s) avec $s \in Q$ un état et Q' la queue avec l'état s retiré de Q.
- *push* est une fonction qui prend une queue $Q \subseteq S$ ainsi qu'un état $s \in S$ et qui retourne une queue Q' qui contient les éléments de Q et s.

En pratique, la queue est équipée d'un ordre $\leq_h$ sur S. On note $<_h$ l'ordre strict sur S induit par $\leq_h$. Pour les réseaux de Petri et les LCS, la queue Q peut être implémentée à l'aide d'un vecteur de listes FIFO ou piles FILO. Voir Section 5.5. À chaque état est associé un entier naturel correspondant dans le cas des réseaux de Petri au nombre de jetons dans le marquage, et dans le cas des LCS au nombre de message dans les canaux. La fonction *push* ajoute l'élément dans la case du vecteur correspondant à ce nombre. Plus l'indice de la case est faible, plus les états qui sont dedans sont prioritaires. La fonction *pop* parcourt le vecteur jusqu'à trouver une case avec une FIFO (resp. FILO) non vide et en retourne le premier (resp. dernier) élément. Si toutes les cases sont vides, la queue est vide. Avec ces fonctions *push* et *pop* on a la propriété suivante : pour deux états s_1 et s_2 tels que $s_1 <_h s_2$, après avoir effectué les

opérations $push(Q, s_1)$ et $push(Q, s_2)$, $pop(Q)$ ne peut retourner s_1 tant qu'elle n'a pas retourné s_2. L'état s_2 est prioritaire sur s_1. En conclusion, la fonction *pop* utilisée par cette heuristique est un algorithme déterministe qui retourne un état avec la priorité maximale.

Nous pouvons donc présenter l'Algorithme 4, qui implémente ce changement par rapport à l'Algorithme 3.

Algorithme 4 : `BCwP_Exploration`$(\mathcal{S}, s_{final}, I, \leq_h)$

Input : Un WSTS $\mathcal{S} = (S, Init, \rightarrow, \leq)$ effectif équipé avec des fonctions $\underline{Cpre}$ et $\underline{Reduce}$, un état cible $s_{final} \in S$ et un invariant clos par le bas I pour $\mathcal{S}$

Output : Existe-t-il un état $s \in S$ tel que $s_{init} \xrightarrow{*} s$ et $s \geq s_{final}$.

```
1  if s_final ∈ ↓Init then
2      return True
3  B ← ∅
4  Q ← push(∅, ({s_final} ∩ I))
5  while Q ≠ ∅ do
6      (Q, s) ← pop(Q)            /* état avec la priorité maximale */
7      if s ∉ ↑B then
8          N ← Cpre(s) \ ↑B              /* nouveaux prédécesseurs */
9          P ← N ∩ I              /* élague les états non couvrables */
10         if P ∩ ↓Init ≠ ∅ then
11             return True
12         B ← Reduce(B ∪ {s})
13         Q ← push(Q, P)
14 return False
```

On note B_k l'ensemble B de l'Algorithme 4 après $k \geq 0$ exécution de la boucle *while*. La preuve de correction est similaire à celle détaillée dans la Section 2.1. De manière similaire au Lemme 2.1.2, on a la propriété suivante : pour une exécution avec $k \geq 0$ itération, où l'algorithme manipule les ensembles B_k et $set(Q_k)$, et si $Cov_{\mathcal{S}} \subseteq I$, on a :

$$\uparrow pre_{\mathcal{S}}^{\leq k}(\uparrow s_{final}) \cap Cov_{\mathcal{S}} \;\subseteq\; \uparrow B_k \cup \uparrow set(Q_k) \;\subseteq\; \uparrow pre_{\mathcal{S}}^{\leq k}(\uparrow s_{final})$$

pour les mêmes raisons que dans la version classique.

C'était le lemme central dans la preuve de correction de l'Algorithme 3.

Théorème 2.2.2. *L'Algorithme 4 termine et est correct.*

Démonstration. La preuve de correction s'obtient de manière similaire à celle utilisée pour l'Algorithme 3.

Pour la preuve de terminaison, supposons qu'il existe une exécution infinie de l'algorithme. La suite $(\uparrow B_k)$ est une suite de clos par le haut croissante. La Proposition 1.1.3 implique qu'elle est ultimement stationnaire. Il existe donc $n \geq 0$, tel que $\uparrow B_n = \uparrow B_{n+1}$. À l'étape n il y a un nombre fini d'états dans la queue Q. À chaque itération, si $e \notin \uparrow B$ la ligne $B \leftarrow \underline{Reduce}(B \cup \{e\})$ est en contradiction avec $\uparrow B_n = \uparrow B_{n+1}$. Donc, après n itération, $e \in \uparrow B$. À partir de l'itération n aucun état n'est ajouté à Q. La queue est vidé après un nombre fini d'étapes et l'algorithme retourne `True`, contradiction. L'algorithme termine donc toujours. □

Ce changement dans l'algorithme de couverture en arrière avec élagage peut donc être utilisé. Nous montrerons que ce changement aide l'analyse en arrière dans la Section 5.5 qui est consacrée aux expérimentations pour les réseaux de Petri.

Remarque. *L'implémentation de l'outil* `QCover` *pour les réseaux de Petri utilise une heuristique Blondin [2016] qui change aussi l'ordre d'exploration. Le wqo* $\leq$ *induit pour les réseaux de Petri est un ordre partiel et la fonction* $\underline{Reduce}$ *est donc simplement la fonction* Min. *L'idée est de remplacer la Ligne 10, qui s'écrit donc* $B \leftarrow \text{Min}(B \cup P)$, *par la ligne* $B \leftarrow \text{Min}(B \cup \text{Min}_{c,k}(P))$. *La nouvelle fonction* $\text{Min}_{c,k}$ *est paramétrée par deux valeurs* c *et* k *choisies expérimentalement (*$c = 10$ *et* $k = 5$ *dans* `QCover`*). Pour un ensemble* $P \subseteq S$, $\text{Min}_{c,k}(P)$ *retourne les* $c + |P|/k$ *marquages avec le moins de jetons, ce qui correspond à ceux dont la priorité est la plus haute selon* $\leq_h$. *La correction et la terminaison de l'algorithme sont conservées. On remarque que laisser de côté un marquage avec priorité basse peut augmenter le nombre d'itérations nécessaires pour finir le calcul. L'outil* `ICover` *qui est un* fork *de* `QCover` *et qui résout le problème de couverture pour les réseaux de Petri utilise aussi cette heuristique.* □

2.2.2 Heuristique de parallélisation

Les invariants ont besoin d'un temps de précalcul. Ce temps peut être long, c'est le cas par exemple avec l'invariant *CSRE* pour les LCS, présenté dans la Section 7.2. Les résultats expérimentaux de la Section 8.4 montre que le temps de précalcul peut être largement supérieur au temps de calcul de l'analyse en arrière sans aucun invariant. Si on lance le calcul en séquentiel, l'algorithme de calcul en arrière ne commencera pas tant que le précalcul n'a pas fini. L'heuristique qu'on présente maintenant consiste donc simplement à utiliser plusieurs threads : un premier principal pour calculer la suite des bases et décider du problème de couverture et un autre thread pour chaque invariant. Chacun de ces derniers threads sera utilisé pour effectuer le précalcul d'un invariant. Une fois ce précalcul fini, le thread passe cette information au thread principal et termine. Il est possible que pour les premières itérations de la boucle `while` de

l'Algorithme 3 l'ensemble B ne bénéficie pas de l'invariant. En général, pour quelques étapes l'ensemble B reste petit. Ensuite, on peut élaguer cet ensemble une fois qu'on a ce qu'il faut pour calculer le problème de l'appartenance à un nouvel invariant. On illustrera expérimentalement cette technique dans la Section 8.5 du Chapitre 8. La Figure 2.1 schématise la différence entre les deux approches.

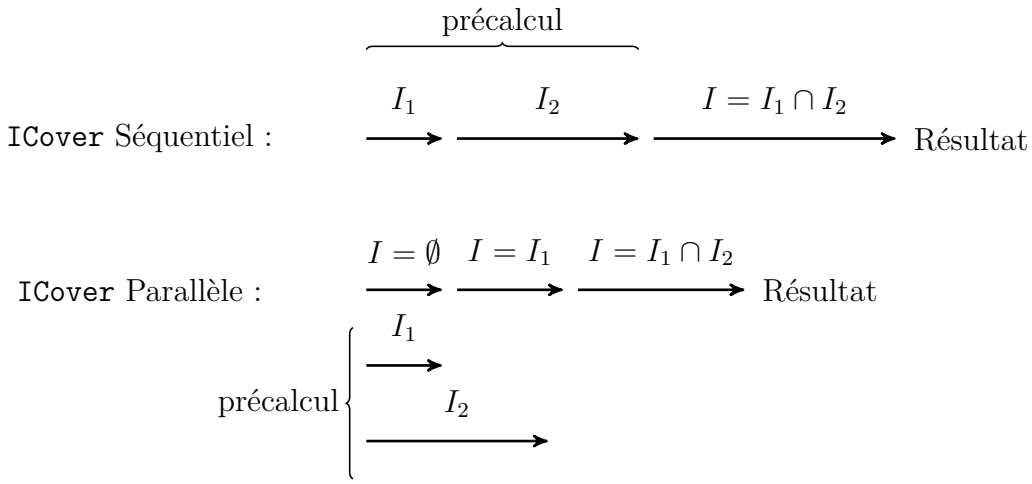

FIGURE 2.1 – Dans la version séquentielle, le précalcul des invariants I_1 et I_2 est d'abord effectué, puis l'analyse en arrière commence. Dans la version parallèle, l'analyse commence sans invariant en même temps que le précalcul des invariants I_1 et I_2. Dès qu'un précalcul a terminé, l'analyse en arrière utilise ce nouvel invariant.

Pour que l'analyse en arrière reste correcte et précise on utilisera des invariants clos par le bas et inductifs. Un ensemble $X \subseteq S$ est dit *inductif* si pour tout $x, x' \in S$ tels que $x \in X$ et $x \rightarrow x'$ on a $x' \in X$. On présente l'Algorithme 5 qui implémente cette heuristique.

Pour les preuves de corrections et de terminaison on présente une suite $(C_k^{(I)})$ paramétrée par une suite décroissante d'invariants $I_1, I_2, \ldots$ clos par le bas et inductifs.

$$C_0^{(I)} = \{s_{final}\} \cap I_0$$

$$C_{k+1}^{(I)} = \underline{Reduce}((I_{k+1} \cap C_k^{(I)}) \cup (\underline{Cpre}(C_k^{(I)}) \cap I_{k+1})))$$

L'Algorithme 5 ne calcule pas directement la suite $(C_k^{(I)})$, mais on prouvera sa correction et sa terminaison à l'aide de cette suite.

Nous allons montrer avec le Lemme 2.2.5 que les bases de la suite $(C_k^{(I)})$ ne perdent pas en précision même en utilisant un invariant seulement plusieurs étapes après le début de l'analyse. Nous avons besoin de plusieurs lemmes techniques avant.

Algorithme 5 : `BCwP_parallel`$(\mathcal{S}, s_{final}, (I_k))$

Input : Un WSTS $\mathcal{S} = (S, Init, \rightarrow, \leq)$ effectif équipé avec des fonctions $\underline{Cpre}$ et $\underline{Reduce}$, un état cible $s_{final} \in S$ et une suite d'invariants de couverture clos par le bas inductifs $I_0, I_1, \ldots$

Output : Existe-t-il un état $s \in S$ tel que $s_{init} \xrightarrow{*} s$ et $s \geq s_{final}$.

```
1  I ← I_0                                   /* premier invariant disponible */
2  if s_final ∈ I then
3  |  C ← {s_final}
4  else
5  |  C ← ∅
6  while ↓Init ∩ C = ∅ do
7  |  I ← I ∩ I_k                           /* nouvel invariant le plus précis */
8  |  C ← C ∩ I_k  /* élagage de la base avec le nouvel invariant
   |   */
9  |  N ← Cpre(C) \ ↑C                      /* nouveaux prédécesseurs */
10 |  P ← N ∩ I                             /* élague les états non couvrables */
11 |  if P = ∅ then
12 |  |  return False
13 |  C ← Reduce(C ∪ P)
14 return True
```

Lemme 2.2.3. *Soit $X \subseteq S$ un ensemble fini et I un invariant clos par le bas et inductif. On a $\uparrow(\underline{Cpre}(X) \cap I) = \uparrow(pre_{\mathcal{S}}(\uparrow X) \cap I)$ et $pre_{\mathcal{S}}(\uparrow X) \cap I = pre_{\mathcal{S}}(\uparrow(X \cap I)) \cap I$. On a donc notamment $\uparrow(\underline{Cpre}(X) \cap I) = \uparrow pre_{\mathcal{S}}(\uparrow(X \cap I)) \cap I$.*

Démonstration. Soit $x \geq y \in (\underline{Cpre}(X) \cap I)$. Par définition de $\underline{Cpre}$, on a $y \geq y' \rightarrow z \geq z' \in X$ avec y', z, z' trois états. Parce que I est clos par le bas on a $y' \in I$ et donc $y' \in (pre_{\mathcal{S}}(\uparrow X) \cap I)$. Ce qui prouve que $x \in \uparrow(pre_{\mathcal{S}}(\uparrow X) \cap I)$.

Soit $x \geq y \in (pre_{\mathcal{S}}(\uparrow X) \cap I)$. On a $y \in \uparrow pre_{\mathcal{S}}(\uparrow X) = \uparrow \underline{Cpre}(X)$ donc il existe $z \in \underline{Cpre}(X)$ tel que $y \geq z$. Parce que I est clos par le bas on a $z \in I$ et donc $z \in \underline{Cpre}(X) \cap I$. On a $x \geq z$ et donc $x \in \uparrow(\underline{Cpre}(X) \cap I)$.

Soit $x \in pre_{\mathcal{S}}(\uparrow X) \cap I$. On a $x \rightarrow y \geq y' \in X$ avec y, y' deux états. Parce que I est inductif, $y \in I$ et parce que I est clos par le bas, $y' \in I \cap X$. On a donc $x \in pre_{\mathcal{S}}(\uparrow(X \cap I)) \cap I$. L'autre inclusion est triviale. □

Lemme 2.2.4. *Soit I et J deux invariants tels que $I \subseteq J$. Si I est inductif, alors pour tout $k \geq 0$, $\uparrow B_k^I = \uparrow(B_k^J \cap I)$.*

Démonstration. On le prouve par induction sur $k \geq 0$. On a $B_0^I = \{s_{final}\} \cap I = (\{s_{final}\} \cap J) \cap I = B_0^J \cap I$ car $I \subseteq J$, ce qui prouve le cas de base. Pour l'étape d'induction supposons que $\uparrow B_k^I = \uparrow(B_k^J \cap I)$ pour $k \geq 0$. On a :

$$\begin{array}{rcll}
\uparrow(B^J_{k+1} \cap I) & = & \uparrow((B^J_k \cup (\underline{Cpre}(B^J_k) \cap J)) \cap I) & \text{[définition de } \underline{\text{Reduce}}\text{]} \\
& = & \uparrow((B^J_k \cap I) \cup (\underline{Cpre}(B^J_k) \cap I)) & [I \subseteq J] \\
& = & \uparrow(B^J_k \cap I) \cup \uparrow(\underline{Cpre}(B^J_k) \cap I) & \text{[développement]} \\
& = & \uparrow B^I_k \cup \uparrow(\underline{Cpre}(B^J_k) \cap I) & \text{[hypothèse de récurrence]} \\
& = & \uparrow B^I_k \cup \uparrow(\underline{Cpre}(\uparrow(B^J_k \cap I)) \cap I) & \text{[Lemme 2.2.3]} \\
& = & \uparrow B^I_k \cup \uparrow(pre_{\mathcal{S}}(\uparrow(B^J_k \cap I)) \cap I) & \text{[Lemme 2.2.3]} \\
& = & \uparrow B^I_k \cup \uparrow(pre_{\mathcal{S}}(\uparrow B^I_k) \cap I) & \text{[hypothèse de récurrence]} \\
& = & \uparrow B^I_k \cup \uparrow(\underline{Cpre}(B^I_k) \cap I) & \text{[Lemme 2.2.3]} \\
& = & \uparrow B^{I_{k+1}}_{k+1} &
\end{array}$$

Ce qui conclut la preuve. □

Lemme 2.2.5. *Si $I_0, I_1, \ldots$ est une suite décroissante d'invariants de couverture clos par le bas inductifs, alors $\uparrow C^{(I)}_k = \uparrow B^{I_k}_k$ pour tout $k \geq 0$.*

Démonstration. On le prouve par induction sur $k \geq 0$. On a

$$\uparrow C^{(I)}_0 = \uparrow(s_{final} \cap I_0) = \uparrow B^{I_0}_0$$

ce qui prouve le cas de base.

Supposons que $\uparrow C^{(I)}_k = \uparrow B^{I_k}_k$ pour un $k \geq 0$. Par définition de $\underline{Reduce}$ on remarque que $C^{(I)}_k = B^{I_k}_k$.

Pour le premier sens, on a :

$$\begin{array}{rcll}
\uparrow C^{(I)}_{k+1} & = & \uparrow(C^{(I)}_k \cap I_{k+1}) \cup \uparrow(\underline{Cpre}(C^{(I)}_k) \cap I_{k+1}) & \text{[développement]} \\
& = & \uparrow(B^{I_k}_k \cap I_{k+1}) \cup \uparrow(\underline{Cpre}(B^{I_k}_k) \cap I_{k+1}) & [C^{(I)}_k = B^{I_k}_k] \\
& = & \uparrow B^{I_{k+1}}_k \cup \uparrow(\underline{Cpre}(B^{I_k}_k) \cap I_{k+1}) & \text{[Lemme 2.2.4]} \\
& = & \uparrow B^{I_{k+1}}_k \cup \uparrow(pre_{\mathcal{S}}(\uparrow(B^{I_k}_k \cap I_{k+1})) \cap I_{k+1}) & \text{[Lemme 2.2.3]} \\
& = & \uparrow B^{I_{k+1}}_k \cup \uparrow(pre_{\mathcal{S}}(\uparrow B^{I_{k+1}}_k) \cap I_{k+1}) & \text{[Lemme 2.2.4]} \\
& = & \uparrow B^{I_{k+1}}_k \cup \uparrow(\underline{Cpre}(B^{I_{k+1}}_k) \cap I_{k+1}) & \text{[Lemme 2.2.3]} \\
& = & \uparrow B^I_{k+1} &
\end{array}$$

Pour l'autre sens, soit $x \in \uparrow B^{I_{k+1}}_{k+1} = \uparrow B^{I_{k+1}}_k \cup \uparrow(\underline{Cpre}(B^{I_{k+1}}_k) \cap I_{k+1})$. Si $x \in \uparrow B^{I_{k+1}}_k$, $x \in \uparrow B^{I_k}_k$ car la suite (I_k) est décroissante. Par hypothèse de récurrence $\uparrow B^{I_k}_k = \uparrow C^{(I)}_k$, donc $x \in \uparrow C^{(I)}_{k+1}$. Sinon si $x \in \uparrow(\underline{Cpre}(B^{I_{k+1}}_k) \cap I_{k+1})$, d'après le Lemme 2.2.3 on a $x \in \uparrow(pre_{\mathcal{S}}(\uparrow B^{I_{k+1}}_k \cap I_{k+1}))$. De même, parce que la suite (I_k) est décroissante, on a $x \in \uparrow(pre_{\mathcal{S}}(\uparrow B^{I_k}_k \cap I_k))$ et donc par hypothèse de récurrence $x \in \uparrow(pre_{\mathcal{S}}(\uparrow C^{(I)}_k \cap I_k)) \subseteq \uparrow C^{(I)}_{k+1}$. □

Nous pouvons maintenant prouver la correction et la terminaison de l'Algorithme 5 à l'aide la suite $(C^{(I)}_k)$.

Théorème 2.2.6. *L'Algorithme 5 termine pour chaque entrée et est correct.*

Démonstration. La suite $(C_k^{(I)})$ ne correspond pas directement aux ensembles C_k calculés par l'Algorithme 5 à chaque itération $k \geq 0$. La suite d'invariants utilisée par l'algorithme n'est pas décroissante, mais l'algorithme utilise en fait la suite $(\cap_k I_k)$ qui est décroissante. De plus, à la Ligne 8, pour l'itération $k+1 \geq 1$, l'algorithme calcule $\underline{Cpre}(C'_k \cap I_{k+1})$ et non $\underline{Cpre}(C'_k)$ car l'ensemble C'_k a été élagué à la Ligne 10. Toutefois on va prouver que $\underline{Cpre}(C'_k \cap I_{k+1}) \cap I_{k+1} = \underline{Cpre}(C'_k) \cap I_{k+1}$. L'inclusion $\subseteq$ est évidente. Pour $\supseteq$, soit $x \in \underline{Cpre}(C'_k) \cap I_{k+1}$. Par définition de $\underline{Cpre}$ on a $x \geq x' \to y \geq y' \in C'_k$, avec x', y et y' trois états. Parce que I_{k+1} est clos par le bas on a $x' \in I_{k+1}$ et parce que I_{k+1} est inductif on a $y \in I_{k+1}$. Donc $x \in \underline{Cpre}(C'_k \cap I_{k+1}) \cap I_{k+1}$.

Après $k \geq 0$ itération de la boucle *while*, la Ligne 13 de l'algorithme calcule donc un ensemble C'_k tel que $C'_k = C_k^{(\cap_k I_k)}$. D'après le Lemme 2.2.5, on a que pour chaque itération $k \geq 0$, $\uparrow C'_k = \uparrow B_k^{I_k}$. Les preuves de correction et de terminaison sont donc similaire à celles du Théorème 2.1.7. □

On verra l'utilité de l'Algorithme 5 dans la partie expérimentale pour les LCS à la Section 8.5.

2.3 Conclusion

Nous avons présenté plusieurs méthodes pour résoudre le problème de couverture dans les WSTS effectifs. Ces méthodes sont paramétrées par des invariants, Section 2.1, ou par un ordre sur les états, Section 2.2.1. On peut appliquer ces méthodes aux réseaux de Petri et aux LCS, car ce sont des WSTS effectifs. Les deux parties suivantes s'intéresseront principalement au choix des invariants mais les deux heuristiques seront elles aussi confrontées à la pratique dans les sections expérimentales.

Deuxième partie

Problème de couverture dans les réseaux de Petri

Chapitre 3

Réseaux de Petri

Dans ce chapitre nous présentons de manière formelle les réseaux de Petri qui ont déjà été présentés rapidement au premier chapitre. Nous donnerons ensuite quelques résultats sur ces systèmes de transitions bien structurés et nous présenterons un état de l'art du problème de couverture dans les réseaux de Petri.

Les réseaux de Petri sont un modèle simple et ancien de systèmes distribués. Un réseau de Petri est composé de *places*, qui peuvent contenir des *jetons*, de *transitions* qui enlèvent et ajoutent des jetons dans certaines places, et d'un ensemble de marquages initiaux. Un *marquage* est un état du système, qui décrit combien il y a de jetons dans chaque place. Le problème de couverture consiste à savoir si on peut passer d'un marquage initial à un marquage qui contient au moins autant de jetons dans chaque place qu'un marquage final. Un algorithme permettant de résoudre ce problème est connu depuis longtemps Karp et Miller [1969a]. Ce modèle a regagné de l'intérêt car il permet de modéliser des systèmes paramétrés en suivant l'approche de German et Sistla [1992]. Les réseaux de Petri permettent de modéliser combien de processus sont dans chaque état possible du système (par exemple une ligne de code précise) en oubliant l'identité ou toute notion d'ordre entre les processus. Des outils permettent de transformer du code source et une propriété de sûreté en un réseau de Petri et un marquage cible, c'est-à-dire en une instance du problème de couverture. Il y a par exemple `Soter` D'Osualdo *et al.* [2013] qui permet de vérifier des propriétés de sûreté dans des programmes écrits en *Erlang*. À l'aide du problème de couverture, on peut vérifier la non-accessibilité de certaines lignes de code, des propriétés d'exclusions mutuelles et si un processus peut avoir plus d'un certain nombre de messages dans sa boîte aux lettres (les messages qui attendent d'être lus par ce processus). L'outil `SATABS` Donaldson *et al.* [2011] permet de faire la même chose avec des programmes concurrents écrits en C. Les modèles produits par `SATABS` sont des *thread transition systems* (TTS) Kaiser *et al.* [2014], qui sont équivalents aux réseaux de Petri. Plusieurs de ces TTS ont été générés pour tester l'outil `BFC` Kaiser *et al.* [2014].

Le problème de couverture est donc un problème intéressant à pouvoir résoudre en pratique. Toutefois sa complexité est exponentielle en espace Lipton [1976]; Rackoff [1978]. Il faut donc des méthodes qui terminent leurs calculs en un temps raisonnable sur les cas générés par les outils d'abstractions comme `Soter` ou `SATABS`. Plusieurs techniques peuvent être appliquées aux réseaux de Petri pour résoudre le problème de couverture, notamment celles qu'on a vus dans la première partie : l'algorithme de couverture en arrière bien sûr mais aussi *Expand, Enlarge and Check* Geeraerts *et al.* [2006] et *Incremental, Inductive Coverability* par Bradley [2011] et adapté par Kloos *et al.* [2013]. L'algorithme de couverture en arrière avec élagage vu au Chapitre 2 avait été utilisé par Blondin *et al.* [2016] avec comme invariant les marquages couvrables dans les réseaux de Petri continus. D'autres méthodes sont plus spécifiques aux réseaux de Petri notamment l'algorithme de Karp et Miller qui construit un arbre représentant l'ensemble des états couvrables. Certaines méthodes ne sont pas complètes, elles ne peuvent que prouver la non-couverture. Il y a par exemple l'inéquation d'état, un invariant de couverture classique, qui consiste en quelque sorte à enlever la contrainte de garder un nombre de jetons positifs dans chaque place le long de l'exécution. Cela permet de calculer une sur-approximation de l'ensemble des marquages couvrables qui est parfois suffisante pour prouver la non-couverture. Il y a aussi une méthode Esparza *et al.* [2014] qui renforce cette inéquation d'état à l'aide de propriétés sur les trappes Desel et Esparza [2005]. Les *trappes* sont des ensembles de places qui piègent les jetons : à partir d'un marquage qui a au moins un jeton dans une place de cet ensemble, tout marquage obtenu en exécutant n'importe quelle transition aura encore au moins un jeton dans cet ensemble. Enfin il y a un algorithme d'élargissement en arrière, où l'idée est d'accélérer le calcul en faisant des hypothèses sur l'impossibilité de couvrir certains marquages puis éventuellement de corriger ces hypothèses si cela ce révèle nécessaire. Ces deux dernières méthodes ainsi que celle appliquée aux réseaux de Petri continus ne sont pas seulement des algorithmes théoriques mais ont été implémentées dans des outils, respectivement `Petrinizer`, `BFC` et `QCover`, qui peuvent être utilisés en pratique.

Nous commencerons par définir formellement ce qu'est un réseau de Petri et nous prouverons que les réseaux de Petri sont des WSTS effectifs. Nous présenterons ensuite un état de l'art de différentes techniques pour résoudre le problème de couverture dans les réseaux de Petri tout en présentant des notions classiques. Seront présentés, l'algorithme de Karp et Miller et ses variantes, l'inéquation d'état, l'algorithme de `Petrinizer` qui utilise la notion de trappe, la méthode de l'élargissement en arrière avec l'outil `BFC` et pour finir l'algorithme pour les réseaux de Petri continus et comment l'utiliser pour résoudre efficacement le problème de couverture dans les réseaux de Petri standard.

3.1 Réseaux de Petri

Nous présentons en détail les réseaux de Petri, qui ont déjà été présentés informellement dans le Chapitre 1.

Un réseau de Petri est un tuple $\mathcal{N} = (P, T, F, Init)$ qui comprend un ensemble fini P de *places*, un ensemble fini T de *transitions* dont l'intersection est vide avec P, une fonction F dite de *flot*, de $(P \times T) \cup (T \times P)$ dans $\mathbb{N}$, et un ensemble $Init \subseteq \mathbb{N}^P$ de marquages *initiaux*. L'ensemble $\mathbb{N}^P$ comprend toutes les applications de P vers $\mathbb{N}$. Les éléments de $\mathbb{N}^P$ sont appelés *marquages*. Ils correspondent aux états du WSTS correspondant. Intuitivement un marquage décrit combien il y a de *jetons* dans chaque place du réseau de Petri. Les jetons sont retirés et émis en exécutant les transitions.

Une transition $t \in T$ peut être exécutée seulement quand il y a au moins $F(p,t)$ jetons dans chaque place du réseau. Exécuter une transition modifie le contenu de chaque place p en retirant d'abord $F(p,t)$ jetons puis en ajoutant $F(t,p)$ jetons. Formellement, on introduit pour chaque transition $t \in T$, la relation binaire en une étape $\xrightarrow{t}$ sur $\mathbb{N}^P$ définie par :

$$m \xrightarrow{t} m' \;\Leftrightarrow\; \forall p \in P : m(p) \geq F(p,t) \wedge m'(p) = m(p) - F(p,t) + F(t,p)$$

La relation binaire $\rightarrow$ est l'union de ces relations binaires pour chaque transition $t \in T$. Formellement, $m \rightarrow m' \Leftrightarrow \exists t \in T : m \xrightarrow{t} m'$. La relation binaire $\xrightarrow{*}$ est la clôture réflexive et transitive de $\rightarrow$.

Pour un marquage $m \in S$ et une place p on note $m(p)$ le nombre de jetons dans la place p dans le marquage m. On notera souvent $m = \Sigma_{p \in P}\ m(p) \cdot p$, par exemple $m = 2 \cdot p_1 + p_2$.

Nous présentons à nouveau dans la Figure 3.1 une modélisation en réseau de Petri du protocole de Lamport Lamport [1986]; Esparza *et al.* [2014] avec deux processus. L'Exemple 3.1.1 montre la formalisation de ce réseau de Petri.

Exemple 3.1.1. *Le réseau de Petri de la Figure 3.1 est décrit formellement par le tuple $\mathcal{N} = (P, T, F, Init)$ avec*

- $P = \{p_1, p_2, p_3, bit_1, \overline{bit_1}, \overline{bit_2}, q_1, q_2, q_3, q_4, q_5\}$
- $T = \{t_1, t_2, t_3, r_1, r_2, r_3, r_4, r_5, r_6\}$
- *la fonction de flot F est telle que*
  - $F(p_1, t_1) = F(\overline{bit_1}, t_1) = F(bit_1, t_1) = F(t_1, p_2) = 1$
  - $F(p_2, t_2) = F(\overline{bit_2}, t_2) = F(t_2, p_3) = F(t_2, \overline{bit_2}) = 1$
  - $F(p_3, t_3) = F(bit_1, t_3) = F(t_3, p_1) = F(t_3, \overline{bit_1}) = 1$
  - $F(q_1, r_1) = F(\overline{bit_2}, r_1) = F(r_1, q_2) = 1$
  - $F(q_2, r_2) = F(bit_1, r_2) = F(r_2, q_3) = F(r_2, bit_1) = 1$
  - $F(q_3, r_3) = F(r_3, q_4) = F(r_3, \overline{bit_2}) = 1$
  - $F(q_4, r_4) = F(\overline{bit_1}, r_4) = F(r_4, q_1) = F(r_4, \overline{bit_1}) = 1$
  - $F(q_2, r_5) = F(\overline{bit_1}, r_4) = F(r_5, q_5) = F(r_5, \overline{bit_1}) = 1$
  - $F(q_5, r_6) = F(r_6, q_1) = F(r_4, \overline{bit_2}) = 1$

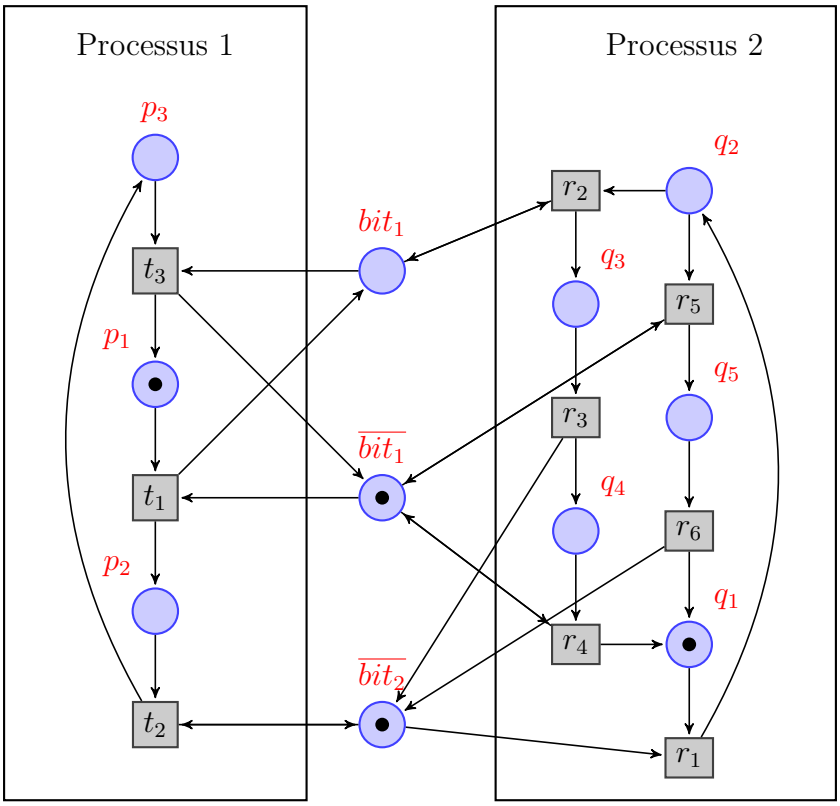

FIGURE 3.1 – Réseau de Petri modélisant le fonctionnement de l'algorithme d'exclusion mutuelle à 1 bit de Lamport

et $F(p,t) = F(t,p) = 0$ *pour toutes les autres combinaisons*

— $Init = \{p_1 + q_1 + \overline{bit_1} + \overline{bit_2}\}$

Un ordre naturel sur les marquages existe pour ces réseaux. On étend l'ordre usuel $\leq$ sur $\mathbb{N}$ en un ordre $\leq$ sur $\mathbb{N}^P$. Deux marquages $m, m' \in \mathbb{N}^P$ sont tels que $m \leq m'$ lorsque pour toute place $p \in P$, $m(p) \leq m'(p)$. On dit que m' *couvre* m quand $m \leq m'$.

Le problème de couverture demande, étant donné un réseau de Petri $\mathcal{N} = (P, T, F, Init)$ et un marquage cible $m_{final} \in \mathbb{N}^P$, s'il existe ou non un marquage $m \in \mathbb{N}^P$ tel que $m_{init} \xrightarrow{*} m$ and $m \geq m_{final}$. Formellement :
Problème : COUVERTURE
Input : Un réseau de Petri $\mathcal{N} = (P, T, F, Init)$ et un marquage cible m_{final}
Output : s'il existe $m_{init} \in Init$ et $m \in \mathbb{N}^P$ tels que $m_{init} \xrightarrow{*} m \geq m_{final}$

Par exemple pour le réseau de Petri du protocole de Lamport, le marquage cible est $m_{final} = p_3 + q_5$, qui n'est pas couvrable.

3.1.1 Résultats de complexités

Karp et Miller [1969b] ont prouvé en 1969 que le problème de couverture est décidable, toutefois leur algorithme n'est pas primitif-récursif. Lipton [1976] a ensuite montré que le problème est au moins exponentiel en espace avant que Rackoff [1978] montre qu'il est exactement exponentiel en espace.

3.1.2 Les réseaux de Petri sont des WSTS effectifs

Nous allons prouver que les réseaux de Petri sont des WSTS effectifs. Les définitions qui suivent sont nécessaires.

On remarque que $\leq$ sur $\mathbb{N}^P$ est un ordre partiel. C'est de plus un wqo par le lemme de Dickson. Le fait que $\leq$ soit un ordre partiel et un wqo implique que pour tout ensemble fini $X \subseteq \mathbb{N}^P$, l'ensemble $\mathrm{Min}(X)$ est l'unique base minimale de $\uparrow X$. La fonction $\mathrm{Min}(X)$ sera donc la fonction *Reduce* introduite dans la définition des WSTS effectifs à la Section 1.4.2.

Pour montrer que les réseaux de Petri sont des WSTS effectifs il nous faut prouver l'existence d'une fonction *Cpre* qui calcule une base minimale des prédécesseurs de $\uparrow m$ pour un marquage m donné. Pour chaque transition t on définit la fonction $Cpre^t$ de $\mathbb{N}^P$ à $\mathbb{N}^P$ telle que pour un marquage $m \in \mathbb{N}^P$ et une place $p \in P$:

$$Cpre^t(m)(p) \;=\; F(p,t) + \max(0, m(p) - F(t,p))$$

Intuitivement, $Cpre^t(m)$ est le plus petit marquage qui peut couvrir m en exécutant t. Formellement, pour toute transition $t \in T$, on définit la fonction pre^t_{S} telle que pour tout ensemble $X \subseteq \mathbb{N}^P$:

$$pre^t_{\mathrm{S}}(X) = \{m \in \mathbb{N}^P \mid \exists m' \in X, m \xrightarrow{t} m'\}$$

Lemme 3.1.2. *Pour toute transition $t \in T$ on a $pre^t_{\mathrm{S}}(\uparrow m) = \uparrow Cpre^t(m)$ pour tout marquage $m \in \mathbb{N}^P$.*

Démonstration. Soit $u \in pre^t_{\mathrm{S}}(\uparrow m)$. Il existe $v \geq m$ tel que $u \xrightarrow{t} v$. Soit une place $p \in P$. Il est vrai que $u(p) \geq F(p,t)$ et $v(p) = u(p) - F(p,t) + F(t,p)$ car $u \xrightarrow{t} v$. On considère deux cas :

- Si $m(p) \leq F(p,t)$ alors $Cpre^t(m)(p) = F(p,t) \leq u(p)$.
- Si $m(p) \geq F(p,t)$ alors $Cpre^t(m)(p) = F(p,t) + m(p) - F(t,p)$. Parce que $v \geq m$, on en déduit que $Cpre^t(m)(p) \leq F(p,t) + v(p) - F(t,p) = u(p)$.

Dans les deux cas, on obtient que $Cpre^t(m)(p) \leq u(p)$. On a donc montré que $u \in \uparrow Cpre^t(m)$.

Dans l'autre sens, soit $u \in \uparrow Cpre^t(m)$. Ce qui implique que $u(p) \geq Cpre^t(m)(p)$ pour toute place $p \in P$. On a $u(p) \geq F(p,t)$ et $u(p) \geq F(p,t) + m(p) - F(t,p)$.

Et donc $u \xrightarrow{t} v$ pour le marquage $v \geq m$ défini par $v(p) = u(p) - F(p,t) + F(t,p)$. On a donc montré que $u \in pre^t_{\mathcal{S}}(\uparrow m)$. □

On étend naturellement la fonction $Cpre^t$ à des ensembles de marquages. On définit la fonction $Cpre$ comme l'extension naturelle de la fonction $Cpre^t$ appliquée à toutes les transitions et telle qu'on ne garde que les éléments minimaux, c'est-à-dire p our tout $M \subseteq \mathbb{N}^P$, $Cpre(M) = \mathrm{Min}(\bigcup_{t \in T} Cpre^t(M))$

Corollaire 3.1.3. *Pour tout sous-ensemble $X \subseteq \mathbb{N}^P$, $pre_{\mathcal{S}}(\uparrow X) = \uparrow Cpre(X)$ et donc $\uparrow pre_{\mathcal{S}}(\uparrow X) = \uparrow Cpre(X)$.*

Nous avons toutes les définitions nécessaires pour prouver que les réseaux de Petri sont des WSTS effectifs. On se restreint toutefois aux réseaux de Petri $\mathcal{N} = (P, T, F, Init)$ où le problème de l'appartenance d'un marquage $m \in \mathbb{N}^P$ dans $\downarrow Init$ est décidable, ce qui est une hypothèse raisonnable. L'ensemble $Init$ est souvent donné sous forme d'une conjonction de contraintes linéaires de la forme $m_{init}(p) \leq x$, avec $x \geq 0$. Dans ce cas le problème de l'appartenance est décidable. Lorsque $Init$ est décrit par une formule de Presburger, l'appartenance à $\downarrow Init$ est aussi décidable Presburger [1929]; Ginsburg et Spanier [1966]; Gastin et Laroussinie [2010].

Théorème 3.1.4. *Un réseau de Petri équipé des fonctions Cpre et* Min *est un WSTS effectif.*

Démonstration. Soit un réseau de Petri $\mathcal{N} = (P, T, F, Init)$. On veut le transformer en un WSTS $\mathcal{S} = (S, Init, \rightarrow, \leq)$. L'ensemble des états S est tout simplement l'ensemble des marquages $\mathbb{N}^P$. L'ensemble des états initiaux est donc naturellement l'ensemble des marquages initiaux. On a déjà vu que l'ordre $\leq$ sur $\mathbb{N}^P$ était un wqo, de plus $\leq$ est décidable. L'ensemble des transitions T et la fonction de flot F induits la relation de transition $\rightarrow$.

Montrons que $\leq$ est compatible avec $\rightarrow$. Soit $s_1, s_2, m_1 \in \mathbb{N}^P$, tels que $s_1 \rightarrow s_2$ et $s_1 \leq m_1$. Le fait que $s_1 \rightarrow s_2$ implique qu'il existe une transition $t \in T$ telle que $s_1 \xrightarrow{t} s_2$. On peut décomposer m_1 en deux marquages, s_1 et m_δ tel que $m_1 = s_1 + m_\delta$. Parce que $s_1 \leq m_1$, on a bien que $m_\delta(p) \geq 0$ pour toute place $p \in P$. On peut exécuter la même transition t depuis m_1 pour obtenir un marquage m_2. Par définition de la fonction F on a $m_2(p) = s_2(p) + m_\delta(p)$. On a donc bien que $m_1 \xrightarrow{*} m_2 \geq s_2$. Ce qui prouve que $\leq$ est compatible avec $\rightarrow$.

Pour tout ensemble $X \subseteq S$, $\mathrm{Min}(X)$ est l'unique base minimale de $\uparrow X$. La fonction Min fait donc office de fonction *Reduce* pour $\mathcal{S}$.

De même, la fonction $Cpre$ est calculable et, d'après le Lemme 3.1.3, pour tout état $s \in S$, $Cpre(s)$ est une base minimale de $\uparrow pre_{\mathcal{S}}(\uparrow s)$. Cela finit de de montrer qu'un réseau de Petri est un WSTS effectif. □

Nous pouvons donc notamment utiliser la méthode décrite dans le Chapitre 2, à la Section 2.1, qui utilise un algorithme de couverture en arrière qui

est accéléré grâce à des invariants de couverture. Cela sera le but du chapitre suivant.

3.2 État de l'art

3.2.1 Algorithmes à la Karp et Miller

L'algorithme de Karp et Miller Karp et Miller [1969a] construit un arbre de couverture pour les réseaux de Petri. Les noeuds de cet arbre permettent de représenter exactement l'ensemble des états couvrables. Cette construction a été généralisée à certains WSTS Finkel [1987, 1990].

L'idée est de commencer avec un arbre qui contient une racine qui représente l'état initial, puis de construire des noeuds fils à partir des transitions en exécutant simplement les transitions possibles. Dans cette forme simple, le calcul peut ne pas finir. Au début une branche représente une exécution. Lorsqu'on détecte un circuit, c'est-à-dire qu'on a une branche $m_0 \xrightarrow{t_1} m_2 \dots \xrightarrow{t_n} m_n$ telle que $m_n \geq m_i$ on comprend que les transitions $t_{i+1}, t_{i+2}, \dots t_n$ sont exécutables un nombre infini de fois à partir de m_i. Par exemple si $m_i = p_1 + p_2$ et $m_n = p_1 + 2 \cdot p_2 + p_3$, on sait qu'on peut couvrir tous les marquages $m = p_1 + x \cdot p_2 + y \cdot p_3$ avec $x, y \geq 0$, en exécutant suffisamment de fois la boucle $t_{i+1}, t_2, \dots t_n$. On note $(\mathbb{N} \cup \{\omega\})^P$ l'ensemble des ω-marquages c'est-à-dire les marquages m tels que $m(p)$ peut être égal à ω, ce qui représente un nombre infini de jetons. L'ordre $\leq$ sur $\mathbb{N}^P$ est naturellement étendu à $(\mathbb{N} \cup \{\omega\})^P$, avec $x < \omega$ pour tout $x \in \mathbb{N}$ et $\omega \leq \omega$.

L'Algorithme 6, dit de Karp et Miller, formalise cet exemple. Il retourne un arbre étiqueté $(N, n_{racine}, E, \Lambda)$ avec (N, n_{racine}, E) l'arbre et Λ la fonction d'étiquetage de N dans $(\mathbb{N} \cup \{\omega\})^P$ qui associe à chaque noeud un ω-marquage. La clôture par le bas de l'ensemble des étiquettes des noeuds de l'arbre représente l'ensemble de couverture du réseau de Petri. L'algorithme utilise la notation $Ancêtres(n)$ pour désigner l'ensemble comprenant tous les noeuds parents du noeud $n \in N$ et le noeud n lui même. Il utilise aussi la fonction d'accélération $Acc(X, m)$ qui à un ensemble de noeuds X et un marquage m associe un marquage qui formalise l'intuition vue au paragraphe précédent, c'est-à-dire calculer l'effet d'une boucle sur marquages en X qui amène à m :

$$\forall p \in P, Acc(X, m)(p) = \begin{cases} \omega & \exists m' \in \Lambda(X), m' < m \wedge m'(p) < m(p) < \omega \\ m(p) & sinon \end{cases}$$

Cet algorithme termine toujours, toutefois il peut calculer plusieurs fois le même sous-arbre. Deux noeuds qui ont le même marquage comme étiquette, donc qui ne sont pas dans la même branche, vont avoir le même sous-arbre. C'est pourquoi une accélération du calcul a été proposée Finkel [1991] pour

Algorithme 6 : `Karp&Miller`$(\mathcal{N}, m_{final})$

Input : Un réseau de Petri $\mathcal{N} = (P, T, F, Init)$ et un marquage cible $m_{final} \in \mathbb{N}^P$

Output : Un arbre étiqueté $(N, n_{racine}, E, \Lambda)$ tel que $\downarrow\Lambda(N) \cap \mathbb{N}^P = Cov_{\mathcal{N}}$

```
1  créer un noeud n_racine tel que Λ(n_racine) = m_init
2  E ← ∅
3  N ← {n_racine}
4  W ← {(n_racine, t) | Λ(n_racine) -t-> ·}
5  while W ≠ ∅ do
6      prendre (n, t) ∈ W et le retirer de W
7      m = post(Λ(n), t)
8      if ∀x ∈ Ancêtres(n), Λ(x) ≠ m then
9          créer un noeud n' tel que Λ(n') = Acc(Ancêtres(n), m)
10         N ← N ∪ {n'}
11         E ← E ∪ {(n, t, n')}
12         W ← W{(n', t') | Λ(n') -t'-> ·}
13 return (N, n_racine, E, Λ)
```

rendre tous les marquages de l'arbre incomparables deux à deux. Malheureusement cette dernière technique n'est pas tout à fait correcte comme démontré dans Geeraerts *et al.* [2010] où les auteurs montrent comment résoudre le problème. Une autre technique d'accélération a été proposée plus récemment Reynier et Servais [2013].

3.2.2 Élargissement en arrière (BFC)

La méthode de l'outil `BFC` Kaiser *et al.* [2014] est comme l'Algorithme 3 une variante de l'algorithme de couverture en arrière, même si les changements par rapport à l'Algorithme 1 sont plus importants que les nôtres. En effet elle est basée sur l'idée d'accélérer le calcul en grossissant les ensemble clos par le haut U_k. Ces ensembles peuvent être donc plus grands que $pre^*_S(\uparrow s_{final})$. Ces grossissements peuvent avoir été faits à tort et dans ce cas il y a besoin de revenir en arrière (*backtrack*).

Ils ont appliqué leur méthode aux *thread transition systems* (TTS) qui sont un modèle équivalent aux réseaux de Petri mais leur méthode peut être généralisée aux WSTS où le wqo $\leq$ est compatible avec $\rightarrow$ *en une étape*. C'est-à-dire que pour tous les états s_1, s_2, s'_1 tels que $s_1 \rightarrow s_2$ et $s_1 \leq s'_1$ il existe un état $s'_2 \in S$ tel que $s'_1 \rightarrow s'_2$ (et pas seulement $s'_1 \xrightarrow{*} s'_2$) et $s_2 \leq s'_2$. C'est une restriction classique. Les TTS et les réseaux de Petri ont cette propriété.

Un avantage de cette méthode est de réduire la *preuve* qu'un état s_{final} n'est pas couvrable dans un système $\mathcal{S} = (S, Init, \rightarrow, \leq)$. En effet, l'analyse en arrière, l'Algorithme 1, calcule un ensemble A tel que $\uparrow A = pre^*_{\mathcal{S}}(\uparrow s_{final})$. Si $\downarrow Init \cap A = \emptyset$, on sait que s_{final} n'est pas couvrable. Un ensemble R tel que $\uparrow pre_{\mathcal{S}}(\uparrow R) \subseteq \uparrow R$, $s_{final} \in \uparrow R$, et $\downarrow Init \cap R = \emptyset$ est une preuve de non couverture. Un outil indépendant peut vérifier qu'un tel ensemble R est une preuve ou non, sans même savoir par quel moyen cet ensemble a été calculé. L'ensemble A calculé par l'Algorithme 1, lorsqu'il retourne `False`, est une preuve de non couverture. On peut vouloir minimiser cette preuve par exemple pour l'analyser plus facilement. C'est ce que fait la méthode `BFC`, avec une réduction jusqu'à 95% en pratique.

La méthode va avoir besoin d'un ensemble $D \subseteq Cov_{\mathcal{S}}$ pour éviter à la recherche en arrière d'être trop imprécise. En pratique ils utilisent un second thread qui exécute en même temps un algorithme de Karp et Miller, et utilise $D = \Lambda(N)$ avec N l'ensemble des noeuds de l'arbre. Cet ensemble N de noeuds grossit et D devient de plus en plus précis, au fur et à mesure du calcul.

L'algorithme remplace l'ensemble A par un arbre dont s_{final} est la racine. Sans élargissement, l'arbre contient les états de A et les états qui ont conduit à ces états.

La méthode utilise deux fonctions, *enlarge* et *backtrack*, dont on décrit rapidement le fonctionnement dans un réseau de Petri $\mathcal{N} = (P, T, F, Init)$ avec un marquage final m_{final}.

Au lieu de directement utiliser un marquage $m = 3 \cdot p_1 + p_2 + p_3$ retourné par la fonction $\underline{Cpre}$, la fonction *enlarge* va chercher un marquage $m' \leq m$ tel que $m' \notin D$. Par exemple le marquage $m' = p_2 \leq m$. Dans ce cas la fonction *enlarge* va ajouter m' à l'arbre. Si $m_1 = p_1 + p_2$ et $m_2 = p_2 + p_3$ sont deux marquages utilisés par l'analyse en arrière classique, on remarque que $\uparrow m_1 \cup \uparrow m_2 \subseteq \uparrow m'$ et donc on peut remplacer m_1 et m_2 par m'. C'est de la que vient la réduction de la preuve.

Dans le cas où m' se révèle être en fait couvrable, la fonction *backtrack* va corriger cette erreur. La méthode détecte cette situation lorsque la fonction $\underline{Cpre}$ retournera un marquage $d \in D$. Ce marquage d a été obtenu à partir de m grâce à une succession d'utilisation de $\underline{Cpre}$ et *enlarge*. L'idée est de réparer cet élargissement trop généreux, et de remplacer les marquages $d \in D$ qui en découlent par des marquages $d' \notin D$ plus précis.

L'utilisation de ces deux fonctions permet d'accélérer la décision du problème de couverture. Dans le cas où l'état cible est non couvrable, on obtient une preuve de non couverture minimisée par rapport à ce que donnerais l'analyse en arrière classique. Et dans le cas où l'état cible est couvrable, grâce aux exécutions de la fonction *enlarge*, l'analyse en arrière a potentiellement eu besoin de moins d'étapes que l'analyse en arrière classique.

3.2.3 Inéquation d'état

Dans cette section on introduit la classique équation d'état. Le problème qui nous intéresse étant celui de couverture et non de celui d'accessibilité, on la présentera sous forme d'inéquation. On peut voir cette inéquation comme une relaxation de la contrainte d'avoir des marquages positifs. Pour couvrir un marquage $m \in \mathbb{N}^P$ on peut passer par des marquages intermédiaires $m' \in \mathbb{Z}^P$ ayant des places avec un nombre négatif de jetons avant de revenir dans les marquages positifs pour couvrir le marquage m. Cette inéquation sera la base d'un invariant présenté dans le chapitre suivant.

On introduit la fonction totale Δ de T à $\mathbb{Z}^P$. Elle est définie telle que pour toute place $p \in P$ et toute transition $t \in T$, $\Delta(t)(p) = F(t,p) - F(p,t)$. On appelle $\Delta(t)$ le *déplacement* d'une transition $t \in T$. Supposons que le marquage m_{final} soit couvrable dans un réseau de Petri $\mathcal{N}$. On a donc l'existence d'un marquage initial m_{init}, des transitions $t_1 \dots t_k$, et un marquage $m \geq m_{final}$ tels que $m_{init} \xrightarrow{t_1} \cdots \xrightarrow{t_k} m$. On a la relation suivante :

$$m_{init} + \Delta(t_1) + \cdots + \Delta(t_k) = m \geq m_{final}$$

En réarrangeant la somme $\Delta(t_1) + \cdots + \Delta(t_k)$, on peut mettre ensemble tous les déplacements $\Delta(t)$ d'une même transition t. On note $\mathbf{x}(t)$ le nombre de fois où apparaît t dans $t_1 \dots t_k$, on a :

$$m_{init} + \sum_{t \in T} \mathbf{x}(t)\Delta(t) \geq m_{final} \tag{3.1}$$

La relation (3.1) est appelée *inéquation d'état* pour le problème de couverture.

On introduit l'ensemble I_S en notant $\mathbb{Q}_{\geq 0}$ l'ensemble des nombres rationnels positifs.

$$I_S = \{m \in \mathbb{N}^P \mid \exists s_{init} \in Init\ \exists \mathbf{x} \in \mathbb{Q}_{\geq 0}^T : s_{init} + \sum_{t \in T} \mathbf{x}(t)\Delta(t) \geq m\} \tag{3.2}$$

Proposition 3.2.1. *Lorsque Init est donné sous la forme d'inéquations linéaires, l'ensemble I_S est un invariant clos par le bas avec un problème de l'appartenance en temps polynomial.*

Cette inéquation peut donc être utilisée avec l'Algorithme 3. La formule créée par l'inéquation d'état peut être résolue par un solveur SMT. En pratique nous utiliserons `Z3` de Moura et Bjørner [2008]. Cet invariant sera testé dans le Chapitre 5 consacré aux expérimentations.

On observe qu'on peut construire de manière similaire un système d'équation pour le problème d'accessibilité, en remplaçant les inéquations par des égalités. Ce système génère lui aussi un invariant. On le note I_E :

$$I_E = \{m \in \mathbb{N}^P \mid \exists s_{init} \in Init\ \exists \mathbf{x} \in \mathbb{Q}_{\geq 0}^T : s_{init} + \sum_{t \in T} \mathbf{x}(t)\Delta(t) = m\} \quad (3.3)$$

Il n'est ni clos par le bas, ni de couverture.

On aura parfois besoin de parler de *solution* de l'équation d'état pour m_{final}. C'est-à-dire qu'on parlera du vecteur $\mathbf{x}$ tel que $m_{init} + \sum_{t \in T} \mathbf{x}(t)\Delta(t) = m_{final}$.

3.2.4 Utilisations des propriétés des trappes avec un solveur SMT (Petrinizer)

Dans Esparza *et al.* [2014] des notions classiques sur les réseaux de Petri, les trappes et les siphons, sont utilisées pour renforcer l'inéquation d'état, qui comme nous l'avons déjà vu est un invariant de couverture. Ce qui veut dire que si cette formule SMT n'est pas satisfiable, alors le marquage cible n'est pas couvrable. Comme l'inéquation d'état, la méthode présentée dans cette section est incomplète : l'algorithme ne peut détecter si un marquage est couvrable.

Une trappe Desel et Esparza [2005], ou en anglais *trap*, est un ensemble de places X tel que s'il y a un jeton dans X à un moment de l'exécution, alors il y en aura toujours au moins un après. Un siphon X est l'inverse d'une trappe, s'il n'y a pas de jeton dans X à un moment de l'exécution, il n'y en aura jamais. Intuitivement lorsqu'on inverse les flèches d'un réseau de Petri une trappe devient un siphon et inversement.

Pour une place $p \in P$, on note ${}^\bullet p$ (resp. $p^\bullet$) l'ensemble des transitions qui émettent (resp. prennent) des jetons dans cette place :

$${}^\bullet p = \{t \in T \mid F(t,p) > 0\}$$

$$p^\bullet = \{t \in T \mid F(p,t) > 0\}$$

Pour une transition $t \in T$, on note ${}^\bullet t$ (resp. $t^\bullet$) l'ensemble des places dans lesquelles t prend (resp. émet) des jetons :

$${}^\bullet t = \{p \in P \mid F(p,t) > 0\}$$

$$t^\bullet = \{p \in P \mid F(t,p) > 0\}$$

On étend ces définitions pour des ensembles de places et de transitions.

On donne les définitions formelles des trappes et des siphons :

Définition 3.2.2 (Trappe)**.** *Un ensemble de places $X \subseteq P$ est une trappe lorsque l'ensemble des transitions qui prennent des jetons dans une place de X, mettent au moins un jeton dans une place de X. Formellement, on a $X^\bullet \subseteq {}^\bullet X$.*

Définition 3.2.3 (Siphon). *Un ensemble de places $X \subseteq P$ est un siphon lorsque l'ensemble des transitions qui mettent des jetons dans une place de X, prennent au moins un jeton dans une place de X. Formellement, on a ${}^\bullet X \subseteq X^\bullet$.*

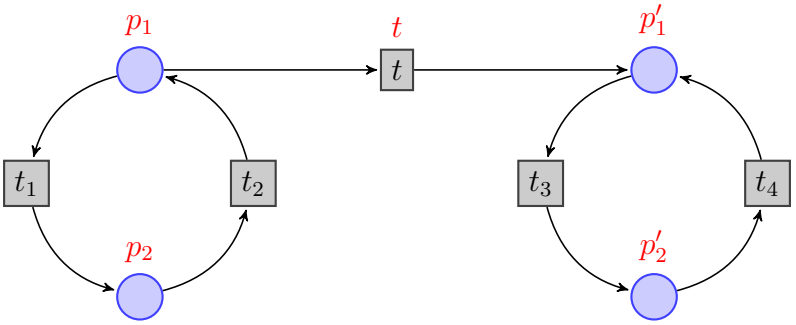

FIGURE 3.2 – Un réseau de Petri simple où $\{p_1, p_2\}$ est un siphon et $\{p'_1, p'_2\}$ une trappe

Exemple 3.2.4. *Dans le réseau de Petri Figure 3.2, l'ensemble $\{p_1, p_2\}$ est un siphon et $\{p'_1, p'_2\}$ une trappe. On remarque aussi que $P = \{p_1, p_2, p'_1, p'_2\}$ est à la fois une trappe et un siphon.*

Exemple 3.2.5. *Dans le réseau de Petri Figure 3.1 Esparza et al. [2014], représenté Chapitre 1, et tiré de l'article dont parle cette section, l'ensemble $\{p_2, q_2, \overline{bit_1}, \overline{bit_2}\}$ est une trappe.*

Pour un ensemble $X \subseteq P$, on dit d'un marquage $m \in \mathbb{N}^P$ qu'il est *marqué* dans X s'il existe $p \in X$ tel que $m(p) > 0$, c'est-à-dire si ce marquage a un jeton dans une place de X. On le note $mark(m, X)$ lorsque c'est le cas et $\neg mark(m, X)$ lorsque m n'est pas marqué dans X. Formellement on a :

$$mark(m, X) = \sum_{p \in X} m(p) \geq 1$$

On note $trap(X)$ la propriété qui est vraie lorsque X est une trappe. Pour une trappe $X \subseteq \mathbb{N}^P$, on a une *contrainte de trappe* qui est la formule suivante $trap(X) \wedge mark(m_{init}, X) \Rightarrow mark(m, X)$.

On utilisera pour la suite un réseau de Petri $\mathcal{N} = (P, T, F, Init)$, tel que $Init$ a un unique marquage m_{init}, c'est-à-dire $Init = \{m_{init}\}$.

En calculant toutes les trappes on peut définir un invariant de couverture clos par le bas qu'on note I_T. C'est la clôture par le bas de l'ensemble des marquages qui satisfont l'équation d'états et les contraintes de trappe pour toutes les trappes $X \subseteq P$.

$$I_T = {\downarrow}\{m \in I_E \mid \forall X \subseteq P, trap(X) \wedge mark(m_{init}, X) \Rightarrow mark(m, X)\}$$

On remarque que cet ensemble est dur à calculer car on doit énumérer tous les ensembles $X \subseteq P$. L'outil `Petrinizer` ne cherche donc pas à calculer directement toutes les trappes, mais va itérativement chercher des trappes pour prouver la non couverture d'un marquage m_{final}.

L'idée est la suivante. On va raffiner l'invariant I_E en lui ajoutant des contraintes de trappe. On cherche à trouver un marquage $m \geq m_{final}$. Supposons que pour un marquage m_{final} on a une solution à l'équation d'état $\mathbf{x}$ telle que $m_{init} + \sum_{t \in T} \mathbf{x}(t)\Delta(t) = m \geq m_{final}$. On peut essayer de trouver un ensemble X telle que $trap(X)$, $mark(m_{init}, X)$ et $\neg mark(m, X)$. Cela prouverait que m n'est pas accessible. Dans ce cas on pourrait continuer le calcul et redemander une autre solution à l'équation d'état. S'il n'y a pas d'autre solution, alors c'est une preuve que m_{final} n'est pas couvrable. Sinon on demande au SMT solveur une nouvelle solution à l'équation d'état et on cherche une nouvelle trappe X'.

On peut itérer ce principe jusqu'à ce que le SMT solveur ne trouve plus de solutions à l'équation d'état qui satisfont aussi les contraintes de trappes qu'on a ajoutées, ce qui prouve la non couverture de m_{final}, ou lorsque que le solveur ne trouve plus de trappe. Cela permet de calculer l'invariant I_T.

Nous avons seulement présenté l'idée de la méthode. L'outil `Petrinizer` effectue d'autres manipulations sur ces équations linéaires pour arriver rapidement au résultat.

3.2.5 Réseaux de Petri continus (QCover)

Une extension des réseaux de Petri appelée réseau de Petri continu a été introduite par David [1987]. La particularité de ces réseaux est qu'on peut exécuter les transitions un nombre rationnel positif de fois. Cela produit des marquages avec un nombre rationnel positif de jetons dans chaque place. Dans la sémantique continue de ces réseaux, le problème d'accessibilité a été prouvé être en temps polynomial Fraca et Haddad [2015a]. Si un marquage n'est pas couvrable avec la sémantique continue, il ne sera pas non plus couvrable avec la sémantique classique. L'ensemble des marquages couvrables en sémantique continue est donc un invariant clos par le bas qui est donc un invariant de couverture. À partir de ce résultat l'outil `QCover` Blondin *et al.* [2016] utilise un encodage en formule SMT du problème d'accessibilité en sémantique continue pour pouvoir utiliser l'Algorithme 3 avec cet invariant.

Comme nous allons le voir, cet invariant est plus précis que l'inéquation d'état. Les résultats expérimentaux du Chapitre 5 montrerons toutefois que cette précision supplémentaire n'est pas forcément nécessaire et conduit à un temps de calcul plus long du problème de l'appartenance à l'invariant.

Soit $\mathcal{N} = (P, T, F, Init)$ un réseau de Petri. La sémantique continue est définie de la façon suivante. Un *marquage continu* est une application $m \in \mathbb{Q}_{\geq 0}^P$ où $\mathbb{Q}_{\geq 0}$ est l'ensemble des rationnels positifs. Pour un rationnel $r \in \mathbb{Q}_{\geq 0}$ et une

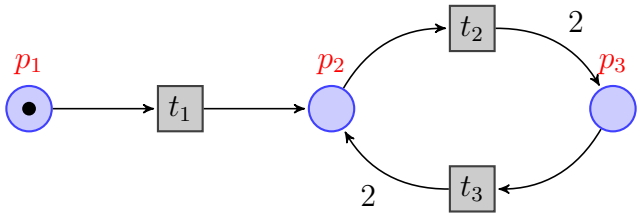

FIGURE 3.3 – Exemple de réseau de Petri continu.

transition t, la relation binaire continue $\overset{rt}{\dashrightarrow}$ sur les marquages continus est définie par :

$$m \overset{rt}{\dashrightarrow} m' \Leftrightarrow \forall p \in P : m(p) \geq r.F(p,t) \wedge m'(p) = m(p) - r.F(p,t) + r.F(t,p)$$

La relation binaire continue en une étape $\dashrightarrow$ est l'union de toutes ces relations $\overset{rt}{\dashrightarrow}$. Formellement, $m \dashrightarrow m'$ lorsqu'il existe $r \in \mathbb{Q}_{\geq 0}$ et $t \in T$ telle que $m \overset{rt}{\dashrightarrow} m'$. La relation binaire continue $\overset{*}{\dashrightarrow}$ est la clôture réflexive transitive de $\dashrightarrow$.

La Figure 3.3 montre un petit exemple de réseau de Petri continu. On peut par exemple faire l'exécution $m_{init} = p_1 \overset{\frac{1}{2}t_1}{\dashrightarrow} \frac{1}{2}p_1 + \frac{1}{2}p_2 \overset{\frac{1}{2}t_2}{\dashrightarrow} \frac{1}{2}p_1 + p_3$

On peut utiliser les réseaux de Petri continus pour prouver la non-couverture d'un marquage dans un réseau de Petri classique. Si on veut savoir si un marquage $m \in \mathbb{N}^P$ d'un réseau de Petri $\mathcal{N} = (P, T, F, Init)$ est couvrable on peut résoudre le problème de couverture dans le réseau de Petri continu $\mathcal{N}_{\mathbb{Q}}$. Ce réseau peut être vu comme un réseau avec des contraintes en moins. Si m n'est pas couvrable dans $\mathcal{N}_{\mathbb{Q}}$ alors il ne l'est pas non plus dans $\mathcal{N}$.

L'ensemble des marquages couvrables en sémantique continue, est l'invariant utilisé par l'outil `QCover` avec l'analyse en arrière de l'Algorithme 3. C'est un invariant clos par le bas, donc un invariant de couverture. Il est défini ainsi :

$$I_C = \{m \in \mathbb{N}^P \mid \exists m_{init} \in Init \; \exists m' \in \mathbb{Q}^P_{\geq 0} : m_{init} \overset{*}{\dashrightarrow} m' \geq m\} \tag{3.4}$$

Pour un réseau de Petri $\mathcal{N}$, on note $\mathcal{N}^{-1}$ le réseau inverse. Intuitivement c'est le réseau dans lequel on a inversé le sens des flèches.

Il faut toutefois pouvoir décider du problème de couverture. La Proposition 3.2.6 montre comment décider du problème de l'accessibilité dans un réseau de Petri continu. Cette proposition fait appel à la notion d'*ensemble d'exécutions*, en anglais *firing sets* notés fs, qui est l'ensemble des ensembles de transitions qui peuvent être empruntées depuis un marquage. Formellement, pour un réseau de Petri continu $\mathcal{N} = (P, T, F, Init)$ et un marquage $m \in \mathbb{N}^P$, on a :

$$fs(\mathcal{N}, m) = \{[\![\pi(\sigma)]\!] \subseteq T \mid \sigma \in (\mathbb{Q}_{\geq 0} \times T)^*, \exists m' \in \mathbb{Q}^P_{\geq 0}, m \overset{\sigma}{\dashrightarrow} m'\}$$

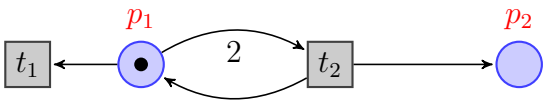

FIGURE 3.4 – Exemple de réseau de Petri continu Blondin *et al.* [2016] où les contraintes de trappes ne permettent pas de montrer que $m_{final} = p_2$ n'est pas accessible.

Le mot σ décrit l'exécution, le vecteur $\pi(\sigma)$ est l'image de Parikh de σ et $[\![\pi(\sigma)]\!]$ est l'ensemble des transitions qui apparaissent dans σ. Par exemple si on effectue $m_{init} = p_1 \xrightarrow{\frac{1}{2}t_1} \frac{1}{2}p_1 + \frac{1}{2}p_2 \xrightarrow{\frac{1}{2}t_2} \frac{1}{2}p_1 + p_3 \xrightarrow{\frac{1}{4}t_1} \frac{1}{4}p_1 + \frac{1}{4}p_2 + p_3$ alors $\sigma = (\frac{1}{2}t_1, \frac{1}{2}t_2, \frac{1}{4}t_1)$, $\pi(\sigma)(t_1) = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}$, $\pi(\sigma)(t_2) = \frac{1}{2}$, $\pi(\sigma)(t_3) = 0$ et $[\![\pi(\sigma)]\!] = \{t_1, t_2\}$.

Proposition 3.2.6. *Fraca et Haddad [2015b] Dans un réseau de Petri $\mathcal{N} = (P, T, F, Init)$, un marquage m_{final} est $\mathbb{Q}$-accessible depuis un marquage $m_{init} \in Init$, si, et seulement si, il existe un vecteur $\mathbf{x} \in \mathbb{Q}^T$ de rationnels positifs, tel que*

(1) $m_{final} = m_{init} + \sum_{t \in T} \mathbf{x}(t)\Delta(t) \quad (m_{final} \in I_E)$
(2) $[\![\mathbf{x}]\!] \in fs(\mathcal{N}, m_{init})$
(3) $[\![\mathbf{x}]\!] \in fs(\mathcal{N}^{-1}, m_{final})$

Exemple 3.2.7. *On peut illustrer la Proposition 3.2.6 avec le réseau de Petri continu de la Figure 3.3. Par exemple le marquage $m_{final} = p_1 + p_2 + p_3$ n'est ni accessible ni couvrable. En effet aucune transition ne donne de jetons dans la place p_1 et il faut forcément enlever au moins un peu de jetons dans p_1 pour en mettre dans les places p_2 et p_3. L'équation d'état trouve comme seule solution le vecteur $\mathbf{x} = (0, 1, 1)$. Son image de Parikh est $[\![\mathbf{x}]\!] = \{t_2, t_3\}$. Toutefois on voit que l'ensemble d'exécutions de m_{init} est $fs(\mathcal{N}, m_{init}) = \{\emptyset, \{t_1\}, \{t_1, t_2\}, \{t_1, t_2, t_3\}\}$ et ne contient pas cet ensemble. La condition (2) de la Proposition 3.2.6 n'est pas remplie. Le marquage $m_{final} = p_1 + p_2 + p_3$ n'est pas $\mathbb{Q}$-accessible dans le réseau de Petri continu $\mathcal{N}$. Ce qui implique qu'il ne l'est pas dans le réseau standard.* □

Pour utiliser l'accessibilité dans les réseaux de Petri en pratique, Blondin *et al.* [2016] transforme cette propriété en une formule booléenne qui pourra être résolue par un solveur SMT. On ne montrera pas comment faire.

Les auteurs de Blondin *et al.* [2016] démontrent que l'accessibilité dans les réseaux de Petri continus est strictement plus précise que les contraintes de trappes. Le Lemme 3.2.8 permet de montrer l'inclusion et l'Exemple 3.2.9 permet de montrer l'inclusion stricte.

Lemme 3.2.8 (Blondin *et al.* [2016])**.** *Dans un réseau de Petri $\mathcal{N} = (P, T, F, Init)$, si un marquage $m \in \mathbb{N}^P$ ne satisfait pas une condition de trappe alors il n'est pas non plus accessible en sémantique continue.*

Exemple 3.2.9. *On va montrer un exemple d'instance du problème de couverture où les propriétés sur les trappes sont insuffisantes pour prouver la non couverture du marquage cible mais où l'accessibilité dans les réseaux de Petri continu l'est. Dans le réseau de Petri de la Figure 3.4 donnée par Blondin et al. [2016], le marquage $m_{final} = p_2$ n'est pas accessible. L'équation d'état a une unique solution $\mathbf{x} = (0, 1)$. Il n'y a qu'une trappe, $\{p_2\}$. Elle n'est pas marquée dans le marquage initial $m_{init} = p_1$. Cette trappe n'est donc pas utile. Toutefois $fs(\mathcal{N}^{-1}, p_2) = \emptyset$, la condition (3) de la Proposition 3.2.6 est donc fausse pour $m_{final} = p_2$ ce qui prouve que m_{final} n'est pas accessible.*

L'accessibilité dans les réseaux de Petri continus est donc un outil puissant pour la couverture. Pour l'Algorithme 3, il faut toutefois un invariant qui soit à la fois précis et dont le problème de l'appartenance est rapidement décidable. Nous verrons dans la partie expérimentale que ce compromis n'est pas toujours en la faveur de l'accessibilité dans les réseaux de Petri continus.

3.3 Conclusion

Nous avons donc vu que les réseaux de Petri étaient des WSTS et que plusieurs techniques pouvaient s'appliquer pour résoudre le problème de couverture. Tout l'enjeu est de réussir à résoudre ce problème en pratique, sur des modèles issus d'exemples concrets de systèmes concurrents ou autres. Le Chapitre 4, le suivant, montrera nos nouvelles techniques pour résoudre ce problème. Au Chapitre 5 nous montrerons notre outil, `ICover`, et nous comparerons les différentes techniques et les différentes implémentations.

Chapitre 4

Invariants pour réseaux de Petri

Dans ce chapitre nous considérons des invariants de couverture. Un invariant est une sur-approximation de l'ensemble des marquages accessibles et un invariant de couverture est un invariant qui contient aussi l'ensemble des marquages couvrables. Au chapitre précédent nous avons déjà vu trois invariants de couverture : l'inéquation d'état, les propriétés de trappes combinées à l'inéquation d'état et l'accessibilité dans les réseaux de Petri continus. Dans ce chapitre nous allons présenter un autre invariant de couverture, l'analyse des signes, et nous allons comparer ces quatre invariants et leur combinaison. De plus nous présentons les *places à oméga*, qui sont une technique qui permet de rendre un invariant de couverture plus précis. L'intérêt des invariants de couverture est de pouvoir les utiliser avec l'Algorithme 3 présenté au Chapitre 2. En effet, nous avons montré que cet algorithme, qui est paramétré par des invariants de couverture, permet de résoudre le problème de couverture dans les WSTS, or les réseaux de Petri sont des WSTS comme nous l'avons vu au chapitre précédent.

Ces invariants de couverture permettent de diminuer la taille de la suite (A_k) qui est calculé dans la version classique de l'algorithme de couverture en arrière comme expliqué dans le Chapitre 1. Cela doit permettre d'accélérer la décision du problème de couverture comme expliquer dans la Section 2.1. Un invariant de couverture se doit d'être suffisamment précis, pour pouvoir réduire la taille des ensembles calculés, mais ne doit pas trop ralentir le calcul en arrière. L'algorithme vu au Chapitre 2 a en effet besoin, à la Ligne 7, $P \leftarrow N \cap I$, de tester si un état est dans l'invariant ou non. Un invariant se doit donc de résoudre un compromis, il faut qu'il soit suffisamment précis pour éliminer des éléments mais ne doit pas être trop coûteux.

Au chapitre précédent nous avons déjà vu l'inéquation d'état qui est une méthode classique souvent combinées avec d'autres méthodes Esparza *et al.* [2014]; Blondin *et al.* [2016]. Nous en avons déjà décrit le principe dans le chapitre précédent. Elle consiste en quelque sorte à ne pas tenir compte de l'ordre dans lequel les transitions sont empruntées. On peut le voir comme

une relaxation de l'obligation des marquages intermédiaires à être positifs. On peut aussi utiliser l'accessibilité dans les réseaux de Petri continus Blondin *et al.* [2016]. Dans ce chapitre nous présentons une analyse des signes. Elle cherche à trouver des places dans lesquelles il n'est pas possible d'avoir des jetons en partant des marquages initiaux.

Ces méthodes peuvent être hiérarchisées selon leur précision. En effet nous avons vu au Chapitre 3, dans la Section 3.2.5, que l'accessibilité dans les réseaux de Petri continus est une technique plus précise que l'utilisation des trappes. Dans la Section 4.2 de ce chapitre nous verrons que la combinaison de l'inéquation d'état avec l'analyse des signes d'un coté et l'accessibilité dans les réseaux de Petri continus de l'autre côté sont en fait deux notions très proches.

Nous finirons par présenter une technique simple pour réduire un peu la taille des réseaux de Petri. Elle consiste à calculer des ensemble de *places à oméga*, c'est-à-dire des ensembles de places Ω telles que le problème de couverture pour un marquage m_{final} ne dépend pas du nombre $m_{final}(p)$ de jetons dans m_{final} dans les places $p \in \Omega$. Ces places sont inutiles pour l'analyse du réseau et sont donc retirées avant de commencer le calcul en arrière. On montrera que cette technique renforce les invariants de couverture.

Nous commencerons par présenter un invariant basé sur l'analyse des signes puis nous comparerons la combinaison de cette analyse et de l'inéquation d'état à la méthode de `QCover` utilisant la sémantique continue pour montrer que cette combinaison est moins précise mais proche sur le plan théorique. Puis nous montrerons qu'il n'y a pas d'inclusions entre cette combinaison et la méthode utilisant les trappes. Enfin nous présenterons les places à oméga et montrerons leur utilité pour renforcer les invariants de couverture.

Dans ce chapitre nous travaillerons avec un réseau de Petri $\mathcal{N} = (P, T, F, Init)$.

4.1 Analyse des signes

Dans cette section, nous cherchons à trouver des places qui ne peuvent pas contenir des jetons en partant des marquages initiaux, même dans le réseau de Petri continu correspondant. L'idée est de pouvoir supprimer des places non accessibles et donc qui ne sont pas utiles pour décider du problème de couverture. Le fait d'enlever des places va aussi permettre d'enlever des transitions : une transition qui doit prendre un jeton dans une place qui ne peut avoir de jeton ne pourra jamais être exécutée, on peut donc l'enlever. Cette analyse calcule un invariant qui peut être utilisé dans l'algorithme de couverture en arrière. En pratique nous procédons différemment, cette analyse sera un précalcul utilisé pour réduire la taille du réseau de Petri avant de commencer l'algorithme de couverture en arrière avec d'éventuels invariants.

L'analyse des signes consiste donc à calculer l'ensemble de places Z, maximale pour l'inclusion, tel que l'ensemble suivant I_Z soit un invariant de cou-

verture inductif :

$$I_Z = \{m \in \mathbb{N}^P \mid \bigwedge_{p \in Z} m(p) = 0\} \tag{4.1}$$

L'unicité de cet ensemble découle du fait que la classe des ensembles Z tels que I_Z soit un invariant est clos par union. Dans la suite de cette section, Z dénotera l'ensemble maximal qui satisfait cette propriété. Cet ensemble peut être construit en temps polynomial grâce à un calcul de point fixe.

On introduit l'opérateur $\text{prop}_t : 2^P \to 2^P$, associé à une transition t, et défini tel que pour tout ensemble de places Q :

$$\text{prop}_t(Q) = \begin{cases} \{q \in P \mid F(t,q) > 0\} & \text{si } \bigwedge_{p \in P \setminus Q} F(p,t) = 0 \\ \emptyset & \text{sinon} \end{cases}$$

Intuitivement, si pour une transition t la condition suivante est respectée $\bigwedge_{p \in P \setminus Q} F(p,t) = 0$, alors pour un marquage avec assez de jetons dans les toutes les places de Q, il est possible d'exécuter t. En particulier, toutes les places qui satisfont $F(t,q) > 0$ ne peuvent être dans Z. Cette propriété est formellement démontré le lemme suivant :

Lemme 4.1.1. *On a* $\text{prop}_t(Q) \subseteq P \setminus Z$ *pour tout* $Q \subseteq P \setminus Z$.

Démonstration. Soit $Q \subseteq P \setminus Z$.

Lorsque $\bigwedge_{p \in P \setminus Q} F(p,t) = 0$ est faux ou qu'il n'existe pas de place $q \in P$ telle que $F(t,q) > 0$ alors la propriété est évidente car l'ensemble $\text{prop}_t(Q)$ est vide.

Dans le cas où $\bigwedge_{p \in P \setminus Q} F(p,t) = 0$ est vraie, et qu'il existe $q \in \text{prop}_t(Q)$ alors on a $q \in F(t,q) > 0$. Nous allons prouver que cette place ne peut pas être dans Z. On introduit les marquages m_t et m'_t définis par $m_t(p) = F(p,t)$ et $m'_t(p) = F(t,p)$ pour tout $p \in P$. Ce sont les marquages minimaux qui satisfassent $m_t \xrightarrow{t} m'_t$. On remarque que pour toutes les places $p \in Z$, on a $p \in P \setminus Q$ car $Q \cap Z$ est vide. On en déduit $F(p,t) = 0$ pour toute place $p \in Z$. Et donc m_t est dans l'invariant inductif I_Z. Comme $m_t \xrightarrow{t} m'_t$, on a $m'_t \in I_Z$. Car $F(t,q) > 0$, on a $m'_t(q) > 0$. Donc $q \notin Z$. □

L'ensemble Z peut être calculé par un calcul de point fixe avec l'aide d'une suite croissante pour l'inclusion $Q_0, Q_1, \ldots$ de places de P définie telle que :

$$Q_0 = \{q \in P \mid \exists s_{init} \in Init\ s_{init}(q) > 0\}$$
$$Q_{k+1} = Q_k \cup \bigcup_{t \in T} \text{prop}_t(Q_k)$$

On remarque que l'ensemble $Q = \bigcup_{k \geq 0} Q_k$ est calculable en temps polynomial. Le lemme suivant nous montre que comme Q est le complément de Z, calculer Q est suffisant pour calculer Z.

Lemme 4.1.2. *On a $Z = P \backslash Q$.*

Démonstration. Le Lemme 4.1.1 permet de prouver par induction sur $k \geq 0$ que $Q_k \subseteq P \backslash Z$ pour tout k. Le cas de base est vrai grâce au fait que $Q_0 \subseteq P \backslash Z$. On a donc que $Q \subseteq P \backslash Z$. On en déduit $Z \subseteq P \backslash Q$.

L'inclusion inverse est obtenue en prouvant que l'ensemble $M = \{m \in \mathbb{N}^P \mid \bigwedge_{p \in P \backslash Q} m(p) = 0\}$ est un invariant inductif. On commence par déduire de $Q_0 \subseteq Q$ qu'il existe $s_{init} \in Init \cap M$. Soit un marquage $m \in M$ et une transition t tels que $m \xrightarrow{t} m'$ pour un marquage m'. On observe que $m(p) \geq F(p,t)$ pour toute place $p \in P$. En particulier, pour $p \in P \backslash Q$, l'égalité $m(p) = 0$ implique $F(p,t) = 0$. Supposons que $m' \notin M$. Dans ce cas, il existe $q \in P \backslash Q$ telle que $m'(q) > 0$. Parce que $m'(q) = m(q) + F(t,q) - F(q,t)$ et $m(q) = 0 = F(q,t)$, on en déduit $F(t,q) > 0$. Et donc $q \in \text{prop}_t(Q)$. Par définition de Q, on a $q \in Q$ et on a une contradiction avec $m' \notin M$. Ce qui prouve que M est un invariant inductif. Par maximalité de Z, on a l'inclusion $P \backslash Q \subseteq Z$. Et donc $Z = P \backslash Q$. □

Corollaire 4.1.3. *L'ensemble Z peut être calculé en temps polynomial.*

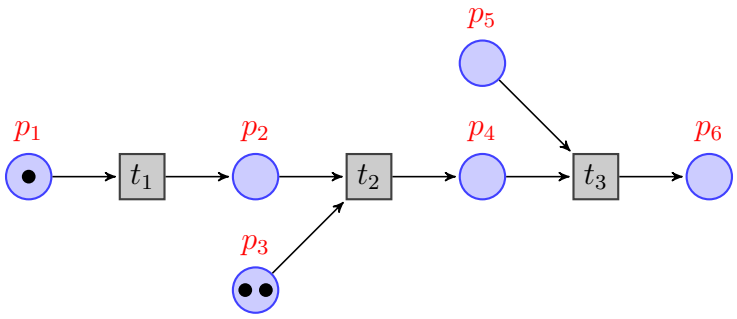

FIGURE 4.1 – Sur ce réseau de Petri, les places de l'ensemble $Z = \{p_5, p_6\}$ sont non accessibles

Remarque. *On peut utiliser cet invariant I_Z comme un précalcul. En effet, on peut enlever toutes les places dans Z, car elles sont non accessibles, et supprimer toutes les transitions qui ont besoin de prendre des jetons dans des places de Z : ces transitions ne pourront jamais être exécutées. Cela permet de diminuer la taille du réseau et donc d'accélérer la décision du problème de couverture.*

Exemple 4.1.4. *Dans le réseau de Petri de la Figure 4.1, avec le marquage initial $p_1 + 2 \cdot p_3$, on a d'abord $Q_0 = \{p_1, p_3\}$, puis, car $\text{prop}_{t_1}(Q_0) = \{p_2\}$, on a $Q_1 = \{p_1, p_2, p_3\}$. Enfin $Q_2 = \{p_1, p_2, p_3, p_4\} = Q_3$. L'ensemble des places inutiles Z est donc $\{p_5, p_6\}$. La transition t_3 ne pourra pas être exécutée dans ce réseau de Petri avec le marquage initial $p_1 + 2 \cdot p_3$. L'analyse des signes va donc enlever les places p_5 et p_6 et la transition t_3.*

4.2 Lien entre accessibilité dans les continus et accessibilité limite

Dans cette section on compare l'invariant I_S de l'inéquation d'état à l'ensemble des marquages couvrables en sémantique continue. Cette sémantique a été présenté à la Section 3.2.5.

On introduit la relation binaire $\overset{\infty}{\dashrightarrow}$ définie sur les marquages continus par $m \overset{\infty}{\dashrightarrow} m'$ lorsqu'il existe une suite $(m_k)_{k\geq 0}$ de marquages continues qui *converge* vers m' avec la topologie classique sur $\mathbb{Q}^P_{\geq 0}$ et telle que $m \overset{*}{\dashrightarrow} m_k$ pour tout k. On dit que m' est *limite-accessible* depuis m. Cette notion d'accessibilité limite est théoriquement très proche de la notion d'accessibilité dans les réseaux de Petri continus. On rappelle que l'outil `QCover` utilise comme invariant I_C, l'ensemble des marquages couvrables en sémantique continue :

$$I_C = \{m \in \mathbb{N}^P \mid \exists s_{init} \in Init\ \exists m' \in \mathbb{Q}^P_{\geq 0} : s_{init} \overset{*}{\dashrightarrow} m' \geq m\} \tag{4.2}$$

Au chapitre précédent, dans la Section 3.2.3, nous avons vu que l'inéquation d'état pouvait être utilisée comme invariant. Dans cette section nous montrons que la combinaison de l'inéquation d'état et de l'analyse des signes, vue comme un précalcul, donne un invariant qui correspond en fait aux marquages limite-accessibles.

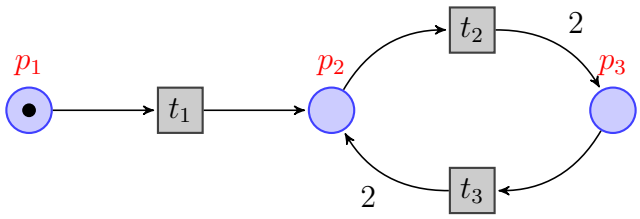

FIGURE 4.2 – Dans ce réseau de Petri simple $m_{final} = p_1 + p_2 + p_3$ n'est pas couvrable mais est limite-accessible.

Exemple 4.2.1. *Regardons le réseau de Petri simple $\mathcal{N}$ de la Figure 4.2. Pour tout entier $k \geq 0$, on a :*

$$(1,0,0) \overset{\frac{1}{k}t_1}{\dashrightarrow} (1-\frac{1}{k},\frac{1}{k},0) \overset{\frac{1}{k}t_2\frac{1}{k}t_3}{\dashrightarrow} (1-\frac{1}{k},\frac{2}{k},\frac{1}{k}) \cdots \overset{\frac{1}{k}t_2\frac{1}{k}t_3}{\dashrightarrow} (1-\frac{1}{k},1+\frac{1}{k},1)$$

On a donc que $(1,0,0) \overset{\infty}{\dashrightarrow} (1,1,1)$. Toutefois on remarque que la relation $(1,0,0) \overset{*}{\dashrightarrow} (1,1,1)$ est fausse. On a $(1,1,1) \notin I_C$ car pour avoir des jetons dans p_2 ou p_3 il faut exécuter un nombre non nul de fois la transition t_1, or il n'y a aucune transition qui permette d'émettre des jetons dans p_1. On ne peut donc pas atteindre un marquage avec un jeton dans p_1 et des jetons dans p_2 ou p_3, même en sémantique continue. □

On prouve maintenant que dans un réseau de Petri dont on aurait effectuer l'analyse des signes comme un précalcul, l'inéquation d'état correspond à l'ensemble des marquages limite-couvrables.

Remarque. *L'énoncé du théorème 7 de Recalde et al. [1999] est faux car il est basé sur une définition trop stricte de l'accessibilité limite. Avec nos notations ce théorème dit que I_S est l'ensemble des marquages m tel qu'il existe une suite $(m_k)_{k\geq 0}$ de marquage continus qui* convergent *vers un marquage $m' \geq m$ avec $m_0 \in Init$ et $m_k \overset{*}{\dashrightarrow} m_{k+1}$ pour tout k. L'Exemple 4.2.1 montre que cette propriété est fausse. Toutefois, leur preuve devient correcte avec nos définitions et notations.*

Théorème 4.2.2 ([Recalde *et al.*, 1999, Theorem 7])**.** *Supposons que $Init$ est un ensemble de marquages de la forme suivante avec $Q \subseteq P$ et $x \in \mathbb{N}^Q$:*

$$Init = \{m_{init} \in \mathbb{N}^P \mid \bigwedge_{q \in Q} m_{init}(q) \leq x(q)\}$$

De plus, on suppose que toutes les transitions sont exécutables dans l'invariant I_Z (Section 4.1). On a :

$$I_S = \{m \in \mathbb{N}^P \mid \exists m_{init} \in Init\ \exists m' \in \mathbb{Q}^P_{\geq 0} : m_{init} \overset{\infty}{\dashrightarrow} m' \geq m\} \tag{4.3}$$

Démonstration. La preuve est organisée ainsi. D'abord, si un marquage m est tel qu'il existe $m_{init} \overset{\infty}{\dashrightarrow} m' \geq m$ pour $m_{init} \in Init$ et $m' \in \mathbb{Q}^P_{\geq 0}$, on en déduit qu'il existe une suite $(m_k)_{k\geq 0}$ de marquages continus, qui *convergent* vers m' telle que $m_{init} \overset{*}{\dashrightarrow} m_k$ pour tout k. Cela implique que pour tout k, m_k satisfait le système d'équations linéaires de I_S. Comme limite, parce que I_S est fermé, on a que m' satisfait le même système. Et parce que I_S est clos par le bas, on a $m \in I_S$

Inversement, soit m un marquage dans I_S, et supposons que toutes les transitions soient exécutable depuis l'invariant clos par le bas I_Z introduit dans la Section 4.1. Pour un mot $\sigma = (r_1 t_1) \dots (r_k t_k)$ avec $(r_j, t_j) \in \mathbb{Q}_{\geq 0} \times T$, on note $\overset{\sigma}{\dashrightarrow}$ la relation binaire sur les marquages continus définie comme la concaténation $\overset{r_1 t_1}{\dashrightarrow} \cdots \overset{r_k t_k}{\dashrightarrow}$. De plus, on dit que σ est exécutable depuis un marquage continu m s'il existe un marquage continu y tel que $x \overset{\sigma}{\dashrightarrow} y$. Étant donné $r \in \mathbb{Q}_{\geq 0}$, on définit $r\sigma$ comme le mot $(rr_1 t_1) \dots (rr_k t_k)$. On observe que si $x \overset{\sigma}{\dashrightarrow} y$ alors $x \overset{\varepsilon\sigma}{\dashrightarrow} x + \varepsilon(y - x)$ pour tout ε tel que $0 \leq \varepsilon \leq 1$. De plus, si $x \overset{\sigma}{\dashrightarrow} y$ alors $rx \overset{r\sigma}{\dashrightarrow} ry$ pour tout $r \geq 0$.

On reprend la suite $(Q_k)_{k\in\mathbb{N}}$ introduite dans Section 4.1. Montrons par induction sur k qu'il existe une suite de marquage continus $(x_k)_{k\in\mathbb{N}}$ telle que $x_0 \in Init$, $x_k \overset{*}{\dashrightarrow} x_{k+1}$, et $x_k(q) > 0$ pour tout $q \in Q_k$. Le cas de base est immédiat par définition de Q_0 et grâce à la forme de $Init$. Supposons que la suite $x_0, \dots, x_k$ est construite. On remarque qu'il existe une suite $t_1, \dots, t_n$ de transitions de T telle que $F(p, t_j) = 0$ pour tout $p \in P \backslash Q_k$ et pour tout

$1 \leq j \leq n$, et telle que $Q_{k+1} \backslash Q_k \subseteq \{q \mid F(t,q) > 0\}$. On observe que pour tout nombre rationnel $\varepsilon > 0$ assez petit, on a $x_k \xrightarrow{\varepsilon t_1 \dots \varepsilon t_n} x_{k+1}$ où x_{k+1} est tel que $x_{k+1}(q) > 0$ pour tout $q \in Q_{k+1}$.

On a donc qu'il existe $x_0 \in \mathit{Init}$, un marquage continu x, un mot w telle que $x_0 \xrightarrow{w} x$ avec $x(q) > 0$ pour tout $q \in Q$ où $Q = \bigcup_k Q_k$. Maintenant, on considère un marquage $m \in I_S$. Il existe $m_{init} \in \mathit{Init}$ et un mot $t_1 \dots t_k$ de transitions tel que $m_{init} + \Delta(t_1) + \cdots + \Delta(t_k) = y \geq m$. Par définition de Init, on peut supposer sans pertes de généralités que $m_{init} \geq x_0$. Soit $\sigma = (1t_1) \dots (1t_k)$ et ε tel que $0 < \varepsilon < 1$. Parce que $x_0 \xrightarrow{w} x$ et $m_{init} \geq x_0$, on en déduit que $m_{init} \xrightarrow{\varepsilon w} (1-\varepsilon)[m_{init} + z_\varepsilon]$ avec $z_\varepsilon = \frac{\varepsilon}{1-\varepsilon}(x + m_{init} - x_0)$. Car $z_\varepsilon(q) > 0$ pour tout $q \in Q$, et car toutes les transitions sont exécutables depuis I_Z, on en déduit qu'il existe $n \geq 1$ suffisamment grand pour qu'un mot $\frac{1}{n}\sigma$ soit exécutable depuis z_ε. Parce que $\frac{1}{n}\sigma$ est exécutable depuis z_ε, en particulier on peut l'exécuter depuis $\frac{k}{n}m_{init} + \frac{n-k}{n}y + z_\varepsilon$ pour tout $0 \leq k < n$. À partir de $m_{init} + \Delta(t_1) + \cdots + \Delta(t_k) = y$, on obtient la relation suivante :

$$\frac{n-k}{n}m_{init} + \frac{k}{n}y + z_\varepsilon \xrightarrow{\frac{1}{n}\sigma} \frac{n-(k+1)}{n}m_{init} + \frac{k+1}{n}y + z_\varepsilon$$

En concaténant les relations précédentes, on en déduit que $m_{init} + z_\varepsilon \xrightarrow{*} y + z_\varepsilon$. Avec $m_{init} \xrightarrow{*} (1-\varepsilon)[m_{init} + z_\varepsilon]$, on en déduit la relation $m_{init} \xrightarrow{*} (1-\varepsilon)[y + z_\varepsilon] = (1-\varepsilon)y + \varepsilon(x + m_{init} - x_0)$. On a donc que, quand ε tend vers 0, on a $m_{init} \xrightarrow{\infty} y$. □

La condition de l'énoncé de ce théorème sur l'ensemble des états initiaux Init est naturelle, l'ensemble des instances de nos expériences du Chapitre 5 la satisfont.

Les deux égalités Équation 3.4 et Équation 4.3 montrent que I_S et I_C sont très proches dans les réseaux de Petri qui satisfont les conditions données dans l'énoncé du Théorème 4.2.2. Ces conditions sont satisfaites lorsqu'on utilise le précalcul qu'on vient de voir à la Section 4.1.

4.3 Les trappes et l'accessibilité limite sont incomparables

Nous avons vu dans la Section 3.2.5 du chapitre précédent que les trappes combinés à l'inéquation d'état étaient moins précises que l'accessibilité dans les réseaux de Petri continus. Nous avons aussi vu à la Section 4.2 précédente, que l'accessibilité limite dans un réseau de Petri qui remplit les conditions du Théorème 4.2.2 était *presque* aussi précise que l'accessibilité dans les réseaux de Petri continus. Toutefois nous allons montrer qu'il n'y a pas de hiérarchie entre les trappes et la accessibilité limite. Il est possible de trouver une instance du problème de couverture, c'est-à-dire un réseau de Petri $\mathcal{N}$ et un un marquage

m_{final}, telle que l'accessibilité limite permette de prouver que m_{final} n'est pas couvrable mais où les propriétés de trappes, même combinées à l'inéquation d'état, ne permettent pas de prouver la non couverture. Et inversement.

L'analyse des signes peut être plus précise que l'utilisation des propriétés de trappes. Par exemple sur le très simple réseau de la Figure 4.3 l'unique trappe $\{p\}$ n'est pas marquée dans le marquage initial $m_{init} = \emptyset$, les propriétés de trappes ne peuvent donc pas prouver la non couverture d'un marquage car $I_T = \mathbb{N}^P$. Mais l'analyse des signes permet facilement de voir que la place p aura toujours 0 jeton. Ce qui permet par exemple de prouver que l'état cible $m_{final} = 2 \cdot p$ n'est pas couvrable.

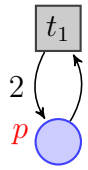

FIGURE 4.3 – Dans ce réseau de Petri simple, il est évident de voir que $m_{final} = \{2 \cdot p\}$ n'est pas couvrable. Mais l'unique trappe $\{p\}$ n'est pas marqué dans le marquage initial $m_{init} = \emptyset$, les propriétés de trappes ne peuvent donc rien prouver.

Avec l'Exemple 4.3.1 on montre que la combinaison des trappes avec l'inéquation d'état peut être suffisante pour prouver la non couverture d'un marquage, alors que l'accessibilité limite échoue.

Exemple 4.3.1. *Dans le réseau de Petri de la Figure 4.4, il n'est pas possible de couvrir $m_{final} = 2 \cdot r$, depuis $m_{init} = p_1 + p_2$. Il y a une solution à l'équation d'état, c'est $\mathbf{x}$ tel que $\mathbf{x}(t_1) = 2$ et $\mathbf{x}(t_2) = 2$. On a donc que $m_{final} \in I_S$ et de plus on observe que toutes les transitions sont exécutables. Toutefois, on remarque que la trappe $\{p_1, p_2\}$ est marqué initialement mais pas dans m_{final}. Ce qui prouve que les trappes peuvent être plus précises que l'accessibilité limite.*

Pour montrer intuitivement le problème de l'accessibilité limite, on construit la suite de marquages (m_k) telle que pour tout $k \geq 0$, $m_k = m_{init} + \sum_{k \geq 0} (\frac{1}{2})^k (r - p_1)$. Prouvons que pour tout $k \geq 0$, $m_{init} \xrightarrow{*} m_k$. Pour $k = 0$, on a $m_0 = p_1 + r$ et $m_{init} \xrightarrow{t_1} p_1 + p_2 \xrightarrow{t_2} p_1 + r$. Supposons que pour un $k \geq 0$, $m_{init} \xrightarrow{*} m_k$. On a $m_k = m_{init} + \sum_{k \geq 0} (\frac{1}{2})^k (r - p_2) \xrightarrow{(\frac{1}{2})^{k+1} t_1} m_k + (\frac{1}{2})^{k+1} (p_2 - p_1) \xrightarrow{(\frac{1}{2})^{k+1} t_2} m_k + (\frac{1}{2})^{k+1} (r - p_1) = m_{k+1}$. Cette suite converge vers m_{final}.

4.4 Places à oméga

Dans cette section nous introduisons les *places à oméga*. Intuitivement pour un ensemble de places à oméga Ω, le problème de couverture pour un marquage

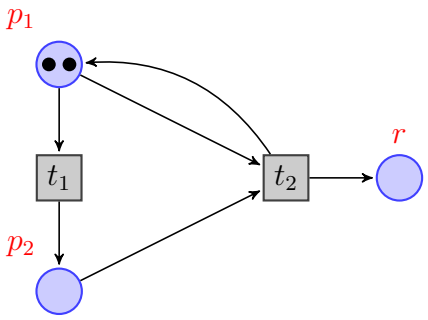

FIGURE 4.4 – Dans ce réseau de Petri, $\{p_1, p_2\}$ est une trappe qui est marquée dans $m_{init} = 2 \cdot p_1$ mais pas dans $m_{final} = 2 \cdot r$. Toutefois $m_{final} = 2 \cdot r$ est limite-accessible.

m_{final} ne dépend pas du nombre $m_{final}(p)$ pour tout $p \in \Omega$. En pratique, ces places peuvent être éliminées. Nous présentons leur définition, comment ces places peuvent être utilisées pour renforcer les invariant de couverture, pourquoi on ne peut espérer toutes les calculer et enfin comment trouver certaines places à oméga à partir d'un calcul en avant simple.

On introduit le vecteur unité $\mathbf{e}_p$ de $\mathbb{N}^P$ défini avec zéro partout à l'exception de l'indice p qui est à un. On appelle une place $p \in \mathbb{N}^P$ une *place à oméga*, si, et seulement si, $Cov_{\mathcal{S}} + \mathbf{e}_p \subseteq Cov_{\mathcal{S}}$.

Nous montrons maintenant avec le lemme suivant que calculer des places à oméga permet de renforcer un invariant de couverture, c'est-à-dire le rendre plus précis.

Lemme 4.4.1. *Soit I un invariant de couverture et Ω un ensemble de places à oméga. L'ensemble $I_\Omega = \{m \in I \mid m + \sum_{p\in\Omega} \mathbb{N} \cdot \mathbf{e}_p \subseteq I\}$ est un invariant de couverture.*

Démonstration. Soit $m \in Cov_{\mathcal{S}}$, on a $m \in I$ car I est un invariant de couverture. Par définition des places à oméga, $m + \sum_{p\in\Omega} \mathbb{N} \cdot \mathbf{e}_p \subseteq Cov_{\mathcal{S}}$ et donc $m \in I_\Omega$. L'ensemble I_Ω est donc un ensemble de couverture. □

Le problème de la place à oméga est de décider si une place est à oméga ou non :
Problème : PLACE À OMÉGA
Input : Un réseau de Petri $\mathcal{N} = (P, T, F, Init)$ et une place $p \subseteq P$
Output : Est-ce que $Cov_{\mathcal{S}} + \mathbf{e}_p \subseteq Cov_{\mathcal{S}}$ (p est une place à oméga) ?

On va montrer qu'on peut éliminer les places à oméga d'un réseau de Petri sans changer la décision du problème de couverture. Dans la suite, U désignera un ensemble de places tel que toutes les places dans $P \backslash U$ sont des places à oméga. L'ensemble U sera donc l'ensemble des places utiles, qui seront conservées pour le calcul. Mais pour le moment, U désignera un ensemble

quelconque de places. Étant donné un marquage $m \in \mathbb{N}^P$, on note $m|_U$ le marquage dans $\mathbb{N}^U$ défini par *restriction* de m sur U, c'est-à-dire formellement défini tel que $m|_U(p) = m(p)$ pour tout $p \in U$. Symétriquement, la restriction de la fonction de flot $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ à U est la fonction de flot $F_U : (U \times T) \cup (T \times U) \to \mathbb{N}$ défini par $F_U(p,t) = F(p,t)$ et $F_U(t,p) = F(t,p)$ pour toute place p dans U et toute transition t dans T. Enfin la restriction d'un réseau de Petri $\mathcal{N} = (P, T, F, Init)$ sur U est le réseau de Petri $\mathcal{N}_U = (U, T, F_U, Init_U)$ où $Init_U = \{s_{init}|_U \mid s_{init} \in Init\}$.

Théorème 4.4.2. *Soit U un ensemble de places d'un réseau de Petri $\mathcal{N}$ tel que les places qui ne sont pas dans U sont des places à oméga. Un marquage m est couvrable dans $\mathcal{N}$ si, et seulement si, $m|_U$ est couvrable dans $\mathcal{N}_U$.*

Démonstration. Dans le but de simplifier les notations, la relation de transition $\xrightarrow{t}$ de $\mathcal{N}_U$ est notée $\xrightarrow{t}_U$. On remarque que si $x \xrightarrow{t} y$ pour deux marquages x, y alors $x|_U \xrightarrow{t}_U y|_U$. Ce qui implique que si un marquage m est couvrable dans $\mathcal{N}$, $m|_U$ est couvrable dans $\mathcal{N}_U$. Nous prouvons maintenant l'autre sens de la propriété.

On commence par prouver par induction sur $n \geq 0$ que pour tout chemin $x_0 \xrightarrow{t_1}_U x_1 \cdots \xrightarrow{t_k}_U x_k$ avec $x_0 \in Init_U$, il existe un marquage $m_k \in Cov_{\mathcal{S}}$ tel que $x_k \leq m_k|_U$. Le cas de base $k = 0$ est immédiat. Supposons que la propriété est vraie pour un $k \geq 0$ et que $x_k \xrightarrow{t_{k+1}}_U x_{k+1}$. Par hypothèse d'induction, il existe un marquage $m_k \in Cov_{\mathcal{S}}$ tel que $x_k \leq m_k|_U$. On observe que dans les places de m_k indexées par $P \backslash U$ on peut mettre un nombre arbitrairement grands de jetons car $P \backslash U$ est un ensemble de places à oméga. Et donc on peut dire que $m_k(p) \geq F(p, t_{k+1})$ pour toute place $p \in P \backslash U$. Cela implique qu'il existe m_{k+1} tel que $m_k \xrightarrow{t_{k+1}} m_{k+1}$. Comme l'ensemble de couverture est un invariant inductif on a $m_{k+1} \in Cov_{\mathcal{S}}$. De plus, on observe que pour toute place $p \in U$ on a $m_{k+1}(p) = m_k(p) - F(p, t_{k+1}) + F(t_{k+1}, p) \geq x_k(p) - F(p, t_{k+1}) + F(t_{k+1}, p) = x_{k+1}(p)$. Ce qui implique que $x_{k+1} \leq m_{k+1}|_U$. Ce qui conclut la preuve par induction.

Maintenant soit m un marquage de $\mathcal{N}$ tel que $m|_U$ soit couvrable dans $\mathcal{N}_U$. Il existe $x_0 \xrightarrow{t_1}_U x_1 \cdots \xrightarrow{t_k}_U x_k$ avec $x_0 \in Init_U$ tel que $m|_U \leq x_k$. D'après le paragraphe précèdent, il existe $m_k \in Cov_{\mathcal{S}}$ tel que $x_k \leq m_k|_U$. On observe que dans les places de m_k indexées par $P \backslash U$ on peut mettre un nombre arbitrairement grand de jetons car $P \backslash U$ est un ensemble de places à oméga. On peut donc dire sans perte de généralité que $m_k(p) \geq m(p)$ pour toute place $p \in P \backslash U$. Cela implique que $m \leq m_k$ et donc que m soit couvrable dans $\mathcal{N}$. □

Les places à oméga peuvent donc être enlevées d'un réseau de Petri sans changer le résultat du problème de couverture. Il est donc intéressant d'en calculer. Mais c'est au moins aussi coûteux de les calculer que de calculer le

problème de couverture. En effet, la question de couverture peut se réduire à connaître si une place est à oméga ou non, comme le montre le lemme suivant :

Lemme 4.4.3. *Le problème de couverture peut se réduire au problème de la place à oméga.*

Démonstration. Soit $\mathcal{N} = (P, T, F, Init)$ un réseau de Petri et m_{final} un marquage cible. On définit un nouveau réseau de Petri $\mathcal{N}_\Omega = (P_\Omega, T_\Omega, F_\Omega, Init_\Omega)$ tel que $P_\Omega = P \cup \{p_{new}\}$ avec $p_{new} \notin P$ une nouvelle place, $T_\Omega = T \cup \{t_{final}, t_{add}\}$ est le nouvel ensemble de transitions, $Init_\Omega$ a toutes les mêmes contraintes que $Init$ plus une nouvelle contrainte : $s_{init}(p_{new}) \leq 0$. De plus la fonction de flot $F_\Omega : (P_\Omega \times T_\Omega) \cup (T_\Omega \times P_\Omega) \to \mathbb{N}$ est telle que :

- pour toute transition $t \in T$, la transition a le même effet que dans $\mathcal{N}$
- t_{add} est une nouvelle transition qui prend un jeton dans p_{new}, et qui met un jeton dans toutes les places $P = P_\Omega \backslash \{p_{new}\}$ ainsi que deux jetons dans p_{new}
- t_{final} est une nouvelle transition qui prend $m_{final}(q)$ jetons pour toute place $q \in P$ et met un jeton dans p_{new}. On remarque que t_{final} ne peut être exécutée qu'à partir d'un marquage $m \geq m_{final}$

On va prouver que m_{final} est couvrable dans $\mathcal{N}$ si, et seulement, si p_{new} est une place à oméga dans $\mathcal{N}_\Omega$.

Pour le premier sens, supposons que m_{final} soit couvrable dans $\mathcal{N}$. Prouvons que tous les marquages dans $\mathbb{N}^{P_\Omega}$ sont couvrables. Parce que m_{final} est couvrable dans $\mathcal{N}$, il est couvrable dans $\mathcal{N}_\Omega$ par définition de F_Ω. Il est donc ensuite possible d'exécuter t_{final} pour couvrir $\mathbf{e}_{p_{new}}$. Puis à partir du marquage $\mathbf{e}_{p_{new}}$ on peut mettre autant de jetons que l'on veut dans toutes les places avec la transition t_{add}. Donc tous les marquages sont couvrables dans $\mathcal{N}_\Omega$. On a donc bien que p_{new} est une place à oméga $\mathcal{N}_\Omega$.

Pour le second sens, si p_{new} est une place à oméga dans $\mathcal{N}_\Omega$ on a en particulier que $\mathbf{e}_{p_{new}}$ est couvrable à moins que $Init = \emptyset$. Il existe donc des marquages $m_0, m_1, \ldots, m_k$ et des transitions $t_1, t_2, \ldots t_n$ dans T_Ω tels que $m_0 \in Init_\Omega$ (et donc $m_0(p_{new}) = 0$), pour tous $0 \leq k < n$, $m_k(p_{new}) = 0$ et enfin $m_0 \xrightarrow{t_1} m_1 \ldots \xrightarrow{*} m_{n-1} \xrightarrow{t_n} m_n \geq \mathbf{e}_{p_{new}}$. Seule la transition t_{final} peut mettre un jeton dans p_{new} quand il n'y aucun jeton dedans. On a donc $t_n = t_{final}$ et pour tous $1 \leq k < n$, $t_k \in T$. De plus cette transition t_{final} peut être exécutée par définition seulement s'il y a au moins $m_{final}(q)$ jetons dans chaque place $q \in \mathbb{N}^P$. On a donc que $m_{n-1} \geq m_{final}$ et $m_0 \xrightarrow{t_1} m_1 \ldots \xrightarrow{*} m_{n-1}$ est un chemin dans $\mathcal{N}$. Cela prouve que m_{final} est couvrable dans $\mathcal{N}$. □

4.4.1 Calculer des places à oméga

On va donc seulement chercher à détecter des places à oméga qui sont faciles à calculer. Nous nous limiterons à un simple calcul de point fixe en avant. On définit la suite d'ensemble de places à oméga telle que :

$$\Omega_0 = \{p \in P \mid \forall s_{init} \in {\downarrow}Init, s_{init} + \mathbf{e}_p \in {\downarrow}Init\}$$
$$\Omega_{k+1} = \Omega_k \cup \bigcup_{\substack{t \in T \\ {}^\bullet t \subseteq \Omega_k}} t^\bullet$$

On rappelle que pour une transition $t \in T$, ${}^\bullet t = \{p \in P \mid F(p,t) > 0\}$ et $t^\bullet = \{p \in P \mid F(t,p) > 0\}$.

Cette suite est croissante et ultimement stationnaire car $\Omega_k \subseteq P$ pour tout $k \geq 0$ et l'ensemble P est fini.

On note $\Omega = \bigcup_k \Omega_k$. On va montrer que Ω est un ensemble de places à oméga.

On va d'abord montrer qu'on peut mettre un jeton dans n'importe quelle place de Ω à partir d'un marquage qui n'a des jetons que dans des places de Ω_0. Et pour cela on a d'abord besoin du lemme technique suivant :

Lemme 4.4.4. *On a $x + y \xrightarrow{*} x' + y'$ pour tous $x \xrightarrow{*} x'$ et $y \xrightarrow{*} y'$.*

Démonstration. On prouve d'abord que si $x \xrightarrow{*} x'$ alors pour tout $z \in \mathbb{N}^P$ on a $x + z \xrightarrow{*} x' + z$. Soit $z \in \mathbb{N}^P$ et supposons que $x \xrightarrow{*} x'$. Il existe $x_0, x_1, \ldots, x_n$ et $t_1, t_2, \ldots, t_n$ tels que $x = x_0 \xrightarrow{t_1} x_1 \ldots \xrightarrow{t_n} x_n = x'$. On a facilement que pour tout $0 \leq k < n$, $x_k + z \xrightarrow{t_k} x_{k+1} + z$. On a alors $x + z \xrightarrow{*} x' + z$. Pour finir, on a $x + y \xrightarrow{*} x' + y \xrightarrow{*} x' + y'$ pour tous $x \xrightarrow{*} x'$ et $y \xrightarrow{*} y'$. □

Lemme 4.4.5. *Pour tout $k > 0$, pour tout $p \in \Omega_k$, il existe un marquage $m_{init} \in \sum_{p \in \Omega_0} \mathbb{N} \cdot \mathbf{e}_p$ tel que $m_{init} \xrightarrow{*} m \geq \mathbf{e}_p$ avec m un marquage.*

Démonstration. On le prouve par induction sur $k > 0$. Pour le cas de base, soit $p \in \Omega_0$, on a $m_{init} = \mathbf{e}_p \geq \mathbf{e}_p$.

Pour l'étape d'induction, soit $k > 0$. Soit $q \in \Omega_{k+1} \backslash \Omega_k$. Par construction de Ω_{k+1}, il existe $t \in T$ tel que $q \in t^\bullet$ et ${}^\bullet t \subseteq \Omega_k$. Par hypothèse d'induction, pour toutes les places $p \in {}^\bullet t \subseteq \Omega_k$, il existe $m_{init,p} \in \sum_{p \in \Omega_0} \mathbb{N} \cdot \mathbf{e}_p$ et m_p tels que $m_{init,p} \xrightarrow{*} m_p \geq \mathbf{e}_p$. Soit $m = \sum_{p \in {}^\bullet t} F(p,t) \cdot m_p$, le marquage minimal qui nous permet d'exécuter t. D'après le Lemme 4.4.4 on a $m_{init} = \sum_{p \in {}^\bullet t} F(p,t) \cdot m_{init,p} \xrightarrow{*} m \geq \sum_{p \in {}^\bullet t} F(p,t) \cdot \mathbf{e}_p$. On peut exécuter t à partir de m : $m \xrightarrow{t} m'$. On a $m'(q) = F(t,q) - F(q,t)$. On a $F(q,t) = 0$ car $q \notin \Omega_k$ et ${}^\bullet t \subseteq \Omega_k$ ce qui implique $q \notin {}^\bullet t$, et $F(t,q) > 0$ car $q \in t^\bullet$. On a donc $m'(q) > 0$ et donc $m_{init} \xrightarrow{*} m' \geq \mathbf{e}_p$. De plus, on peut voir facilement que $m_{init} \in \sum_{p \in \Omega_0} \mathbb{N} \cdot \mathbf{e}_p$. □

Lemme 4.4.6. *Toute place $p \in \Omega$ est une place à oméga.*

Démonstration. Soit $p \in \Omega$. Soit $m \in Cov_S$ un marquage couvrable. Il existe $m_{init} \in Init$ tel que $m_{init} \xrightarrow{*} m' \geq m$. D'après le Lemme 4.4.5, il existe

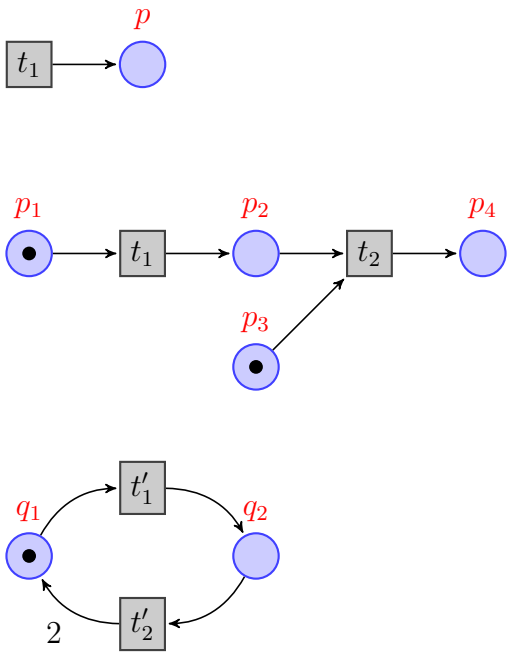

FIGURE 4.5 – Exemple de réseau de Petri où p, q_1 et q_2 sont des places à oméga

$m'_{init} \in \sum_{p\in\Omega_0} \mathbb{N} \cdot \mathbf{e}_p$ tel que $m'_{init} \xrightarrow{*} m'' \geq \mathbf{e}_p$. Par définition de Ω_0 et par induction directe, on a $m_{init} + m'_{init} \in {\downarrow}Init$, et donc il existe $\delta \in \mathbb{N}^P$ tel que $m_{init} + m'_{init} + \delta \in Init$. D'après le Lemme 4.4.4 on a $m_{init} + m'_{init} + \delta \xrightarrow{*} m' + m'' + \delta \geq m + \mathbf{e}_p$. Ce qui implique $m + \mathbf{e}_p$ est couvrable. On a donc $Cov_{\mathcal{S}} + \mathbf{e}_p \subseteq Cov_{\mathcal{S}}$. La place p est une place à oméga. □

On peut donc éliminer les places Ω car elles ne sont pas utiles pour la décision du problème de couverture.

Sur la Figure 4.5 on a représenté un réseau de Petri pour illustrer ce qu'est une place à oméga et comment notre suite $\Omega_0, \Omega_1, \ldots$ en calcule. L'ensemble des marquages initiaux est le singleton $Init = \{p_1 + p_3 + q_1\}$. On observe que p est une place à oméga car on peut exécuter la transition t_1 sans conditions. Les places q_1 et q_2 sont aussi des places à oméga, car on peut exécuter la boucle t'_1, t'_2 un nombre infini de fois. Toutefois notre calcul des places à oméga calculera seulement p avec $\Omega = \{p\}$ car $p \in t^\bullet$ avec ${}^\bullet t = \emptyset \in \Omega_0 = \emptyset$. On aura en effet $\Omega_0 = \emptyset$ et ensuite $\Omega_k = \{p\}$ pour tout $k \geq 1$.

On modifie maintenant l'ensemble des marquages initiaux en $Init = \{m \in \mathbb{N}^P \mid \forall p' \in P, p' \neq p_1 \Rightarrow m(p') = 0\}$, c'est-à-dire tous les marquages qui n'ont pas de jetons dans toutes les places autres que p_1 mais qui peuvent avoir un nombre illimité de jetons dans p_1. Dans ce cas on a $\Omega_0 = \{p_1\}$ et $\Omega_1 = \{p_1, p_2, p\} = \Omega$ car ${}^\bullet t_1 = \{p_1\} \subseteq \Omega_0$ et $t_1^\bullet = \{p_2\}$.

Nous verrons dans la Section 5.4 du chapitre suivant consacré aux expérimentations dans les réseaux de Petri, que cette technique peut être très efficace

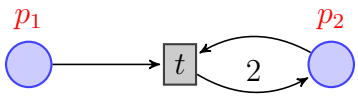

FIGURE 4.6 – Réseau de Petri simple où le précalcul de place à oméga permet d'éviter une explosion combinatoire

même avec un ensemble Ω petit. Même si une unique place est retirée d'un réseau de Petri qui en contient des dizaines, l'analyse en arrière peut être accélérée par un facteur trois dans certains cas. L' Exemple 4.4.7 montre pourquoi on peut avoir un effet important.

Exemple 4.4.7. *Prenons le petit réseau de Petri $\mathcal{N}$ de la Figure 4.6 avec l'ensemble des marquages initiaux $Init = \{m_{init} \in \mathbb{N}^P \mid m_{init}(p_2) \leq 1\}$.*

On veut savoir si le marquage cible $m_{final} = 100 \cdot \mathbf{e}_{p_2}$ est couvrable ou non. Sans le précalcul des places à oméga, la suite (B_k^I) sera pour tout $0 < i < 100$, $B_i^I = \{(j \cdot \mathbf{e}_{p_1} + (100-j) \cdot \mathbf{e}_{p_2}) \in \mathbb{N}^P | \ j \leq i\}$. Et enfin $\downarrow Init \cap B_{99}^I \neq \emptyset$ prouvera que m_{final} est couvrable. La taille de la dernière base B_{99}^I sera 99 alors que le réseau de Petri est de taille très réduite.

Mais notre calcul des places à oméga va calculer $\Omega = \Omega_0 = \{p_1\}$, et donc le réseau de Petri restreint $\mathcal{N}_{P\setminus\Omega}$ contiendra seulement la place p_2, et une transition simple qui prend un jeton dans p_2 et en met deux dans la même place. L'ensemble des marquages initiaux sera donc $Init|_U = \downarrow \mathbf{e}_{p_2}$. Et maintenant la suite sera pour tout $0 \leq i < 100$, $B_i^I|_U = \{(i \cdot \mathbf{e}_{p_1} + (100-i) \cdot \mathbf{e}_{p_2})\}$. Toutes les bases seront de taille une. □

Chapitre 5

Expérimentations

Nous avons vu dans les deux chapitres précédents différentes méthodes pour résoudre le problème de couverture dans les réseaux de Petri. Dans ce chapitre nous nous focaliserons sur les résultats expérimentaux.

Nous présentons notre outil, `ICover`, et ses résultats pratiques sur un certain nombre d'instances utilisées par les autres outils. Il est basé sur l'outil `QCover`, décrit à la Section 3.2.5, et il est disponible en téléchargement Geffroy *et al.* [2016b]. Nous nous intéresserons d'abord au nombre d'instances résolues par les outils. En faisant notamment une comparaison entre les instances *Safe*, où le marquage cible n'est pas couvrable, et les instances *Unsafe*, où le marquage cible est couvrable. Mais nous ferons aussi des comparaisons entre les différentes combinaisons d'invariants, notamment sur l'efficacité des invariants, c'est-à-dire quel est le pourcentage de fois où un marquage est en dehors d'un invariant. Les comparaisons seront notamment entre l'inéquation d'état, combiné ou non à l'analyse des signes, et l'accessibilité dans les réseaux de Petri continus. Ce qui est intéressant à tester avec les invariants de couverture utilisés par l' Algorithme 3 est le compromis entre le nombre de marquages élagués et le temps mis pour tester si les marquages sont dans l'invariant ou non. Nous verrons aussi l'impact important du calcul des places à oméga. Nous testerons aussi l'efficacité de l'heuristique d'exploration présentée dans la Section 2.2.1 du Chapitre 2. C'est l'heuristique qui change l'ordre d'exploration des états, en utilisant une mesure d'ordre de priorité entre deux états.

Dans ce chapitre nous présenterons d'abord l'implémentation de notre outil `ICover`. Puis nous comparerons les différents outils sur des ensembles d'instances du problème de couverture. Une attention particulière sera portée sur la différence de précision entre les invariants. Puis nous verrons l'impact du calcul des places à oméga. Enfin pour finir nous présenterons les résultats de l'heuristique d'exploration qui permet d'accélérer encore le calcul.

5.1 Implémentations

Nous avons développé un outil, `ICover`, qui est en fait un *fork* de `QCover` et qui permet de résoudre le problème de couverture dans les réseaux de Petri avec l'algorithme de couverture en arrière accéléré avec l'inéquation d'état, l'analyse des signes et le calcul des places à oméga. Ces deux dernières techniques sont implémentées sous la forme de précalculs qui réduisent la taille du réseau de Petri. L'outil peut être téléchargé Geffroy *et al.* [2016b]. Il est possible d'activer ou de désactiver les différentes options, pour comparer les différents invariants entre eux, mais aussi de voir l'effet de l'heuristique d'exploration. Il utilise en entrée le format de fichier `spec` introduit par l'outil `mist` Ganty [2004]. Ce format permet entre autres de décrire un réseau de Petri et son marquage cible.

5.2 Comparaisons avec les outils existants

Pour tester les différents outils nous avons utilisés les 176 réseaux de Petri et leur propriété de sûreté déjà utilisés par des outils comme `QCover` et `Petrinizer`. Ils viennent de différentes sources : `Mist` Ganty [2004], `BFC` Kaiser *et al.* [2014], de programmes `Erlang` modélisés sous forme de réseau de Petri D'Osualdo *et al.* [2013], et aussi d'autres exemples appelés `medical` et `bug_tracking` Esparza *et al.* [2014]. On a laissé les programmes fonctionner pendant 2000 secondes, sur une machine munie d'un processeur Intel(R) Core(TM) i7-4770 CPU à 3.40GH et de 16 GB de mémoire vive avec Ubuntu Linux 14.04 en système d'exploitation. Le temps de calcul des différents exemples sera la somme des temps `system` et `user` retournés par la commande `time(1)`.

Sur l'ensemble des instances, `QCover` a résolu le problème 105 fois sur 115 instances non couvrables, et 37 fois sur 61 instances couvrables. `ICover` a été capable de résoudre une instance couvrable supplémentaire. Nous avons aussi utilisé `QCover` avec le réseau de Petri réduit par le précalcul. Cette variante, que l'on appelle `QCover/Pp`, a été capable de résoudre une instance non couvrable de plus que `ICover`. Sur les 142 instances que `QCover` a résolu, il a pris 9847 secondes, `QCover/Pp` a utilisé 6854 secondes et `ICover` a eu besoin de 5464 secondes.

L'utilisation de l'inéquation d'état sans le précalcul a été capable de résoudre 37 instances couvrables mais seulement 98 instances non couvrables. Il y a notamment 6 instances `medical` où la combinaison de l'inéquation d'état et du précalcul est suffisamment précise pour prouver que le marquage cible est non couvrable, sans avoir besoin de lancer l'analyse en arrière. Sur ces 6 instances, l'inéquation d'état sans le précalcul n'est pas suffisante. L'analyse en arrière doit être lancée mais elle ne finit pas dans le temps imparti. Sur

les 135 instances où `QCover` et la version `ICover` sans précalcul ont tous les deux fini, `QCover` a eu besoin de 8171 secondes et la version de `ICover` 9477 secondes.

L'outil `Petrinizer` Esparza *et al.* [2014] est capable de résoudre 95 instances non couvrable mais aucun instance couvrable. L'outil `BFC` Kaiser *et al.* [2014] quand à lui est capable de résoudre seulement 63 instances non couvrables, soit 42 instances de moins que `ICover` et `QCover`. Il est toutefois capable de résoudre 59 instances couvrables contre 37 et 38 pour `QCover` et `ICover`. Au total il est capable de résoudre 122 instances contre 142 et 143 pour `QCover` et `ICover`.

Il y a une erreur de dépassement mémoire avec l'exemple de la suite `Soter`, *ring___single_message_in_mailbox___depth_2*, lors de la conversion du fichier `spec` en structure de donnée `python`. Parce que `ICover` est un fork de `QCover`, toutes les versions de `QCover` et `ICover` sont impactées de la même manière. Aucune autre instance n'a eu de problème, et toutes les versions ont soit terminé, soit commencé l'analyse en arrière.

5.3 Efficacité des invariants

Nous avons un cadre général pour résoudre le problème de couverture dans les réseaux de Petri. Ce cadre est paramétré par des invariants. Dans cette section nous voulons comparer l'efficacité des différents invariants. Nous en avons en fait trois : l'accessibilité dans les réseaux de Petri continus `QCover` Blondin *et al.* [2016], l'inéquation d'état (Section 3.2.3) et l'analyse des signes (Section 4.1) qui est en fait utilisée comme un précalcul. Nous avons donc plusieurs combinaisons à tester.

Figure 5.1(a) montre la comparaison entre `ICover` et `QCover` en temps. La ligne en diagonale représente l'égalité en temps entre les deux outils. Chaque point représente une instance du problème de couverture, c'est-à-dire un réseau de Petri et un marquage cible. Quand un point est en dessous de la ligne, cela indique que `ICover` a été plus rapide que `QCover` et inversement. L'échelle des deux axes est exponentielle. Les instances en bas à gauche ont donc été résolues facilement par les deux outils et sont les moins intéressantes à analyser. Il y a plusieurs instances que `QCover` a résolu très rapidement, en moins d'une seconde, et pour lesquelles `ICover` a eu besoin de plusieurs dizaines de secondes pour conclure. Pour ces instances, l'invariant utilisant la sémantique continue de `QCover` était suffisamment précis pour éliminer les états cibles. L'outil `QCover` n'a donc pas eu besoin de faire l'analyse en arrière. Par contre l'invariant d'`ICover` n'était pas assez précis, et a dû effectuer l'analyse en arrière.

Les Figures 5.1(b) et (c) montrent les comparaisons intermédiaires : `ICover` comparé à `QCover/Pp` et `QCover/Pp` comparé à `QCover`. On observe que le

précalcul a un impact très important sur la différence de performance entre `ICover` et `QCover`.

5.3.1 Précisions des invariants

Nous avons vu au chapitre précédent, dans la Section 4.2, que le précalcul de l'analyse rend l'inéquation d'état proche de l'accessibilité dans les réseaux de Petri continus `QCover` Blondin *et al.* [2016]. La Figure 5.1(d) compare l'efficacité de l'élagage de `ICover` et de `QCover`. Chaque point représente une instance du problème de couverture. On sait que `QCover` élague au moins autant de marquages que `ICover`, grâce à sa formule SMT plus compliquée. Une valeur de 100% indique que `ICover` a été capable d'élaguer exactement les mêmes marquages que `QCover`. Sur la plupart des instances la valeur de 100% est obtenue.

5.3.2 Efficacité de l'analyse des signes

Figure 5.1(e) et Figure 5.1(f) illustrent l'effet du précalcul sur la taille des réseaux de Petri. La première montre le pourcentage de places qui sont considérées comme utiles par l'analyse des signes. Certains réseaux de Petri ont gardé toutes leurs places mais d'autres les ont presque toutes perdues. Par exemple sur `reslockbeh__critical__depth_2` de `soter` le précalcul n'a gardé que 175 places sur les 10194 initiales, soit 1,7% de places gardées. De manière globale, on observe que la plupart des réseaux de Petri ont perdus un nombre non négligeable de places. La seconde figure montre les mêmes statistiques mais pour les transitions, c'est-à-dire combien de transitions étaient encore présentes après le précalcul. On observe que globalement il y a eu moins de transitions que de places qui ont été retirées lors du précalcul. La moitié des réseaux de Petri ont gardé leurs transitions mais par exemple `x0_AA_q1` de `medical` n'a gardé que 227 transitions sur les 5431 initiales, soit 4,2% de transitions non retirées.

5.3.3 Comparaison avec les trappes

Nous avons vu à la Section 3.2.4, qui décrit l'outil `Petrinizer`, qu'on pouvait utiliser les propriétés de trappes pour résoudre le problème de couverture pour les instances non couvrables. On a vu à la Section 4.3 que les propriétés de trappes et l'accessibilité limite, utilisée par `ICover`, sont incomparables.

Il y a 90 instances où l'accessibilité limite seule a été suffisante pour prouver la non couverture de l'état cible, là où `Petrinizer` a été capable de résoudre 95 instances.

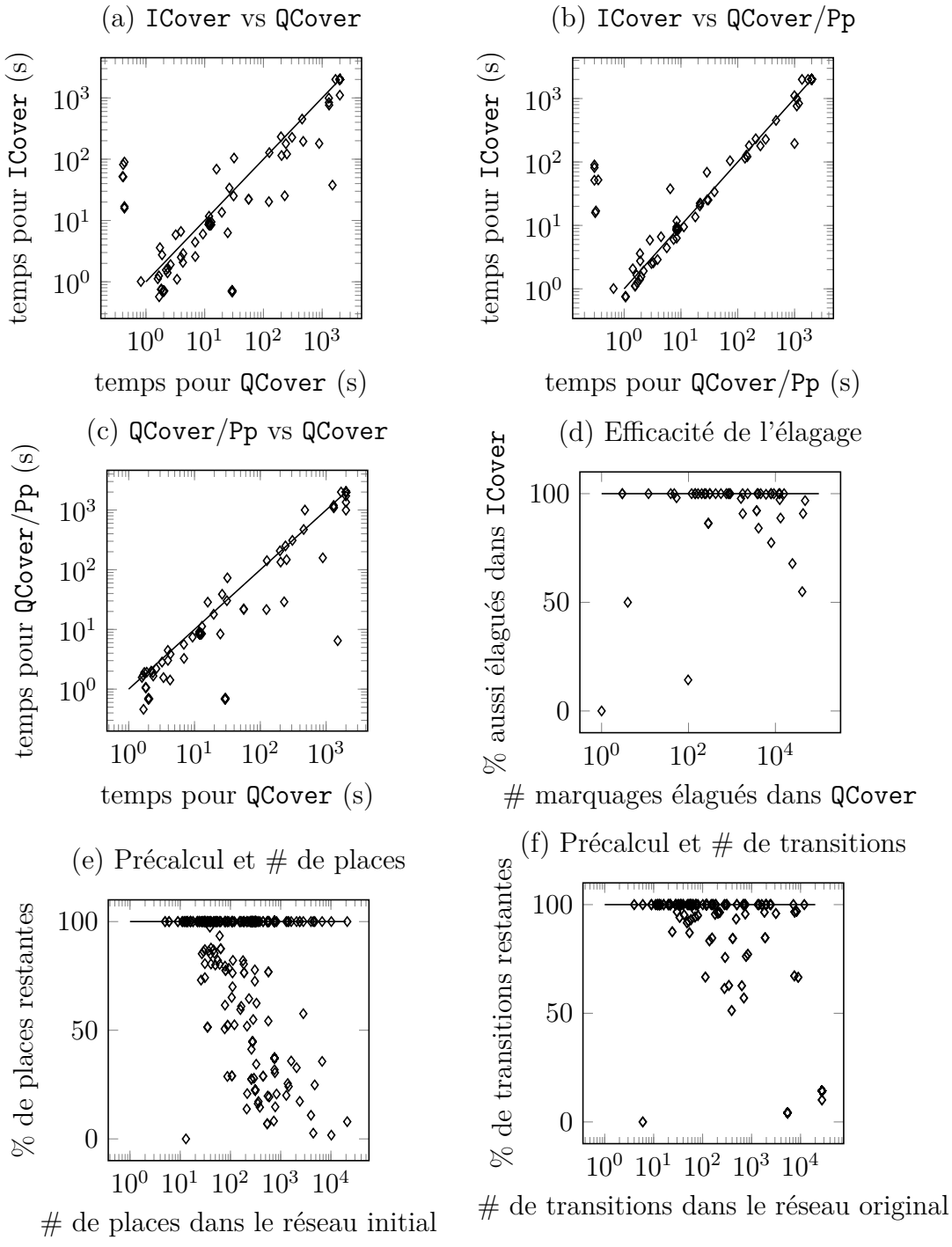

FIGURE 5.1 – Résultats expérimentaux de ICover, QCover et QCover/Pp

5.4 Places à oméga

On va analyser les effets du calcul des places à oméga qui a été décrit à la Section 4.4.

On analyse d'abord combien de places à oméga ont été trouvée, et donc éliminées, avec la méthode décrite dans la Section 4.4. Sur les 176 réseaux de Petri, 112 places à oméga places ont été trouvées avec le calcul de la suite (Ω_k). Et 102 de ces réseaux avaient une unique place à oméga. Ces réseaux de Petri sont en grande partie issus des modèles `tts` générés par `Soter` et `SATABS`. Dans ces réseaux, l'unique place à oméga représente le nombre de threads à l'état initial. Et ce nombre n'a pas de contrainte, ce sont donc des places à oméga. De manière surprenante, pour une instance, `kanban` de l'outil `Mist` Ganty [2004], l'ensemble Ω trouvé par notre précalcul était exactement l'ensemble de *toutes* les places. L'instance a donc été facilement résolue, alors que `QCover` et `ICover` sans le précalcul des places à oméga n'ont pas terminé.

On analyse maintenant l'effet en temps de calcul des places à oméga. Le calcul des places à oméga est utilisé comme un précalcul comme l'analyse des signes. Les deux calculs sont effectués en même temps via le même calcul en avant. C'est une option de `ICover`. On note `ICover` $+\ \Omega$ la procédure qui commence par éliminer des places à oméga, et ensuite qui continue comme la version `ICover` classique comme utilisé jusqu'à maintenant. `ICover`$+\Omega$ a résolu autant d'instances non couvrables que `ICover` mais six instances couvrables de plus. Sur les 143 instances où les deux algorithmes ont conclu, `ICover` a mis 6572 secondes et `ICover` $+\ \Omega$ a pris 2221 secondes. De plus, `ICover` $+\ \Omega$ a mis seulement 2059 secondes pour résoudre les même instances que `QCover` a résolu et où ce dernier a mis 9847 secondes.

Comme dans la section précédente, on présente la comparaison de `ICover`$+\ \Omega$ et `ICover` dans la Figure 5.2(a). On observe un bon impact. Dans la Figure 5.2(b) on compare `ICover` $+\ \Omega$ et `QCover` qui résume les effets de nos modifications : inéquation d'état, analyse des signes et places à oméga. On voit une très nette amélioration : il y a plus de points au dessus de la diagonale.

Même si une unique place est retirée d'un réseau de Petri qui en contient des dizaines, l'effet peut être important : l'analyse en arrière peut être accélérée par un facteur trois dans certains cas. L' Exemple 4.4.7 du Chapitre 4 montre pourquoi on peut avoir cet effet.

5.5 Heuristique d'exploration

Dans le Chapitre 2 nous avons présenté deux heuristiques pour accélérer le calcul en arrière. La seconde, voir Section 2.2.2, est utile dans le cas où le temps de précalcul d'un invariant est élevé. Pour les invariants tels que l'inéquation

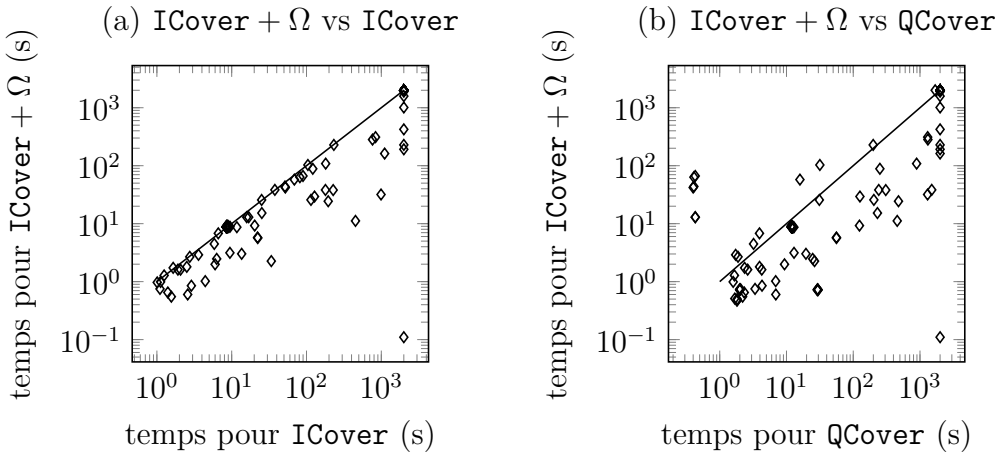

FIGURE 5.2 – Effets expérimentaux du calcul des places à oméga

d'état et l'accessibilité dans les réseaux de Petri continus, le précalcul consiste à construire les formules SMT pour le SMT solveur. Cette tâche ne prend pas beaucoup de temps. Cette heuristique n'est donc pas utile pour ces invariants. L'analyse des signes et le calcul des places à oméga peuvent être vues comme des invariants, toutefois leur objectif est de retirer des places et des transitions. Il est donc compliqué d'utiliser leur résultat après que le calcul en arrière soit lancé. De plus, ces deux précalculs sont rarement long, comme nous le verrons expérimentalement.

Par contre la première heuristique, voir Section 2.2.1, celle sur l'ordre d'exploration, peut être utilisé. Nous l'avons utilisé avec l'ordre $\leq_h$ déjà utilisé par `QCover` Blondin *et al.* [2016] qui compte tout simplement le nombre de jetons dans les marquages : pour deux marquages m_1 et m_2 on a $m_1 \leq_h m_2$ lorsque que $\sum_{p \in P} m_1(p) \geq \sum_{p \in P} m_2(p)$. Informellement, si $m_1 \leq_h m_2$ cela veut dire que m_1 a au moins autant de jetons que m_2 au total. Par exemple $2 \cdot p_3 + p_4 \leq_h p_1 + p_2$.

Pour deux marquages m_1 et m_2 ayant la même priorité, $m_1 \leq_h m_2$ et $m_2 \leq_h m_1$, l'heuristique d'exploration n'impose pas d'ordre sur quel marquage sera exploré avant. En pratique, la queue Q peut être implémentée à l'aide d'un vecteur de listes FIFO ou de piles FILO. On note `Exp`$(fifo)$ et `Exp`$(stack)$ les versions qui utilisent la méthode `ICover` avec l'heuristique d'exploration en utilisant respectivement des FIFO et des FILO.

On va se comparer uniquement à la version `ICover` $+ \Omega$. Celle ci a été capable de résoudre le problème de couverture sur 149 instances. La version `Exp`$(fifo) + \Omega$ a été capable de résoudre 2 instances couvrable de plus et la version `Exp`$(stacks) + \Omega$ une de plus. Toutefois en terme de temps, comme on peut le voir sur la Figure 5.3, il n'y a pas de gain clair et il est même courant que l'heuristique ralentisse le calcul total, surtout en utilisant une FILO.

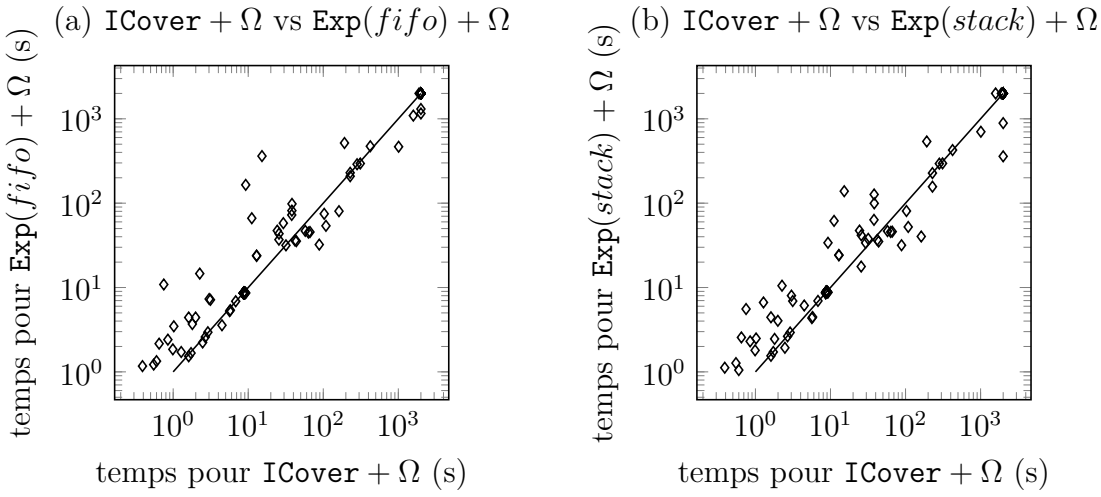

FIGURE 5.3 – Résultats expérimentaux pour $\texttt{Exp}(fifo) + \Omega$ et $\texttt{Exp}(stack) + \Omega$

Notre version d'`ICover` qui arrive a résoudre le plus d'instances est $\texttt{Exp}(fifo) + \Omega$, qui combine l'inéquation d'état, l'analyse des signes, le calcul des places à oméga et l'heuristique d'exploration. Elle en résout 151 contre 142 pour `QCover`.

Troisième partie

Problème de couverture dans les systèmes de canaux à pertes

Chapitre 6

Systèmes de canaux à pertes

Ce chapitre et les deux qui vont suivre reprennent la même structure que ceux de la partie 2 consacrés aux réseaux de Petri. Cette partie s'intéresse aux *systèmes de canaux à pertes*, ou en anglais LCS pour *lossy channel systems*, que nous avons présentés rapidement au Chapitre 1.

Un LCS se veut la représentation d'un groupe de systèmes qui s'échangent des messages via des canaux de communication. La particularité est que ces canaux ne sont pas fiables : des messages peuvent être perdus. Il n'y a pas de contraintes ou de propriétés sur ces pertes. Tous les messages peuvent être perdus ou aucun, ils peuvent être perdus au début, à la fin ou au milieu d'un canal, etc. Ces systèmes doivent donc s'affranchir de ces pertes pour pouvoir effectuer leur calcul en toute sécurité. Le problème de couverture dans les LCS est équivalent à celui de savoir si une localité est accessible ou non.

Les LCS ont été introduit et étudiés dans les années 90 Abdulla et Jonsson [1993]; Abdulla *et al.* [1998]; Annichini *et al.* [2001]. Les LCS ont regagné de l'intérêt parce qu'ils sont liés aux systèmes à mémoire faible Atig *et al.* [2010]; Abdulla *et al.* [2016a]. En effet les systèmes concurrents qui utilise le modèle de mémoire TSO peuvent être simulés par des LCS, et inversement Atig *et al.* [2010]. Des protocoles de service web ont aussi été testés et améliorés pour fonctionner correctement avec des canaux à pertes Ravn *et al.* [2011]. Une autre utilisation des LCS est de les voir comme des sur-approximations des systèmes à canaux sans perte. Si une propriété de sûreté est vérifiée dans le système avec des pertes dans les communications, elle sera aussi vérifiée dans le même système sans perte dans les communications. Le problème de couverture dans les LCS est hyper-Ackermanien Chambart et Schnoebelen [2008].

Contrairement aux réseaux de Petri, il n'y a quasiment pas d'outils spécifiquement développés pour résoudre le problème de couverture dans les LCS. Il n'y a que l'outil `TReX` Annichini *et al.* [2001] qui est spécialement fait pour les LCS, mais il n'est plus disponible et sa terminaison n'est garantie que pour les instances couvrables. Il y a deux outils, `LASH` Boigelot et Godefroid [1999] et `McScM` Heußner *et al.* [2012], qui peuvent être utilisés avec des LCS mais qui

sont à la base conçus pour les systèmes de canaux sans perte.

Dans ce chapitre nous présentons de manière formelle les LCS, nous prouvons que ce sont des systèmes de transitions bien structurés et nous présentons un état de l'art du problème de couverture.

6.1 LCS

Un *système de canaux à pertes*, ou en anglais LCS pour *lossy channel systems*, est un tuple $\mathcal{C} = (Q, q_{init}, M, d, \Delta)$ où Q est un ensemble fini de *localités*, $q_{init} \in Q$ est la *localité initiale*, M est un alphabet fini non vide de *messages* qui peuvent être échangés via les canaux, d est le *nombre de canaux*, et $\Delta \subseteq Q \times Op \times Q$ est l'ensemble des transitions appelées *règles*, où $Op = \{1, \ldots, d\} \times \{!, ?\} \times M$ est l'ensemble des *opérations* sur les canaux. Une opération est :

— soit une *émission* $i!m$ d'un message m dans un canal i,
— soit une *réception* $i?m$ d'un message m depuis un canal i.

La sémantique d'un LCS $\mathcal{C} = (Q, q_{init}, M, d, \Delta)$ est le système de transitions $\mathcal{S} = (S, Init, \to, \leq)$ défini comme suit. L'ensemble des états d'un LCS est $S = Q \times (M^*)^d$. Les états sont appelés des configurations. Une configuration $s = (q, w_1, \ldots, w_d)$ est composée d'une localité q et de mots w_i qui décrivent le contenu du canal i. L'unique configuration initiale est $c_{init} = (q_{init}, \varepsilon, \ldots, \varepsilon)$ c'est-à-dire que le système commence dans la localité initiale avec tous les canaux vides.

Pour chaque règle $\delta = (q, op, q')$ dans Δ, on définit la relation binaire $\xrightarrow{\delta}$ sur S telle que : $s \xrightarrow{\delta} s'$ avec $s = (q, w_1, \ldots, w_d)$ et $s' = (q', w'_1, \ldots, w'_d)$ lorsque :

— soit op est une émission $i!m$, et dans ce cas $w'_i = w_i \cdot m$ et $w'_j = w_j$ pour tout $j \neq i$,
— soit op est une réception $i?m$, et dans ce cas $m \cdot w'_i = w_i$ et $w'_j = w_j$ pour tout $j \neq i$.

Pour définir les pertes, nous définissons la relation $\to_\lambda$ sur S. Formellement, $c \to_\lambda c'$ avec $c = (q, w_1, \ldots, w_d)$ et $c' = (q', w'_1, \ldots, w'_d)$ lorsque les localités q et q' sont les mêmes et qu'il existe un canal $i \in \{1, \ldots, d\}$ tel que w'_i est obtenu à partir de w_i en supprimant ou non un message (i.e., dans le cas où on supprime un message $w_i = u \cdot m \cdot v$ et $w'_i = u \cdot v$, pour un message $m \in M$ et $u, v \in M^*$) et $w'_j = w_j$ pour tout $j \neq i$. Par exemple, $(q, a \cdot b \cdot a) \to_\lambda (q, a \cdot a)$ mais aussi $(q, a \cdot b \cdot a) \to_\lambda (q, a \cdot b \cdot a)$. Le fait que la relation de perte $\to_\lambda$ contient aussi la relation identité est dû à des raisons techniques.

On définit maintenant la relation binaire $\to$ sur S comme l'union de la relation $\to_\lambda$ et des relations $\xrightarrow{\delta}$ où δ est dans Δ.

On présente à nouveau une modélisation du protocole du bit alterné Bartlett *et al.* [1969] vu dans la première partie.

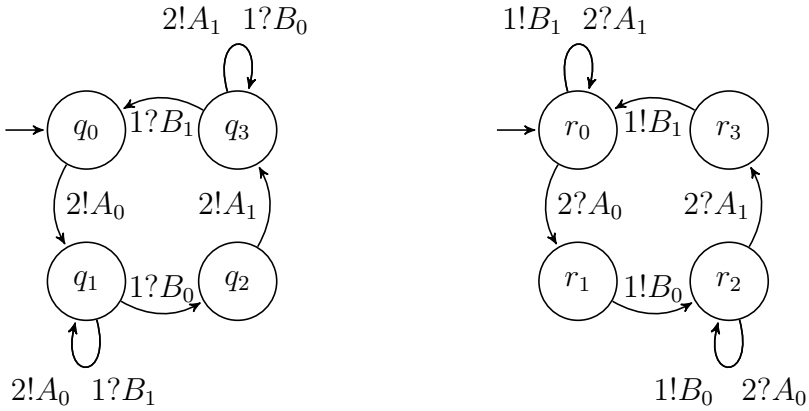

FIGURE 6.1 – Exemple de système de canaux à pertes modélisant le protocole du bit alterné Bartlett *et al.* [1969]

Exemple 6.1.1. *Le LCS de la Figure 6.1 est décrit formellement par le tuple* $\mathcal{C} = (Q, q_{init}, M, d, \Delta)$ *avec*

- $Q = Q_1 \times Q_2$ *avec* $Q_1 = \{q_0, q_1, q_2, q_3\}$ *et* $Q_2 = \{r_0, r_1, r_2, r_3\}$
- $q_{init} = q_0 \times r_0$
- $M = \{A_0, A_1, B_0, B_1\}$
- $d = 2$
- $\Delta = \{(q \times r, op, q' \times r) \mid (q, op, q') \in \Delta_1, r \in Q_2\} \cup \{(q \times r, op, q \times r') \mid (r, op, r') \in \Delta_2, q \in Q_1\}$ *avec* $\Delta_1 = \{(q_0, 2!A_0, q_1), (q_1, 2!A_0, q_1), (q_1, 1?B_1, q_1), (q_1, 1?B_0, q_2), (q_2, 2!A_1, q_3), (q_3, 2!A_1, q_3), (q_3, 1?B_0, q_3), (q_3, 1?B_1, q_0)\}$ *et* $\Delta_2 = \{(r_0, 1!B_1, r_0), (r_0, 2?A_1, r_0), (r_0, 2?A_0, r_1), (r_1, 1!B_0, r_2), (r_2, 1!B_0, r_2), (r_2, 2?A_0, r_2), (r_2, 2?A_1, r_3), (r_3, 1!B_1, r_0)\}$

Étant donné un LCS $\mathcal{C} = (Q, q_{init}, M, d, \Delta)$, on définit la relation $\leq$ sur $S = Q \times (M^*)^d$ comme suit. Pour deux configurations, $c = (q, w_1, \ldots, w_d)$ et $c' = (q', w'_1, \ldots, w'_d)$, on a $c \leq c'$ lorsque $q = q'$ et w_i est un sous-mot de w'_i pour tout $i \in \{1, \ldots, d\}$. Un mot w est un sous-mot de w' si w est obtenu à partir de w' en supprimant des messages. Le lemme suivant est la conséquence directe des définitions de $\rightarrow_\lambda$ et de $\leq$.

Lemme 6.1.2. *Les relations* $\geq$ *et* $(\rightarrow_\lambda)^*$ *sont identiques.*

Le problème de couverture dans les LCS est de savoir s'il existe un chemin depuis la configuration initiale c_{init} jusqu'à une configuration c qui couvre la configuration cible c_{final}.
Problème : COUVERTURE
Input : Un LCS $\mathcal{C} = (Q, q_{init}, M, d, \Delta)$ et une configuration cible c_{final}
Output : s'il existe une configuration $c \in S$ tel que $c_{init} \xrightarrow{*} c \geq c_{final}$

Ce problème est hyper-Ackermanien Chambart et Schnoebelen [2008]. Comme nous l'avons démontré pour les réseaux de Petri, une complexité théorique éle-

vée du problème de couverture ne nous empêche pas de pouvoir le résoudre en pratique sur des exemples non triviaux. On remarque que pour les LCS ce problème est équivalent au problème de l'accessibilité et même équivalent au problème de l'accessibilité d'une localité.

6.1.1 Les LCS sont des WSTS effectifs

Nous allons maintenant montrer que les LCS sont des WSTS effectifs.

Lemme 6.1.3. *L'ordre $\leq$ est un wqo.*

Démonstration. L'ordre sous-mot est un wqo par le lemme de Higman [1952] et parce qu'il y a un nombre fini de localités et de canaux, on peut appliquer le lemme de Dickson pour prouver que $\leq$ est un wqo. □

On remarque que le wqo $\leq$ est un ordre partiel sur S, donc $\mathrm{Min}(X)$ est la base minimale de $\uparrow X$ pour tout ensemble $X \subseteq S$.

Il nous faut maintenant montrer qu'il existe une fonction $\underline{Cpre}$ qui, pour chaque configuration $s \in S$, calcule une base minimale de $\uparrow pre_S(\uparrow s)$. On introduit deux fonctions $cpre_!$ et $cpre_?$ de $M \times M^*$ vers M^* définies telles que :

$$cpre_!(m, w) \;=\; \begin{cases} w' & \text{si } w = w' \cdot m \\ w & \text{sinon} \end{cases}$$

$$cpre_?(m, w) \;=\; m \cdot w$$

Voir Exemple 6.2.

$$\begin{array}{ll} cpre_!(a, abc) = abc & cpre_?(a, \varepsilon) = a \\ cpre_!(a, abca) = abc & cpre_?(c, abb) = cabb \end{array}$$

FIGURE 6.2 – Exemples de calcul de $cpre_!$ et de $cpre_?$.

Pour une configuration $c = (q, w_1, \ldots, w_d)$ et une règle $\delta = (p, i\#m, q) \in \Delta$, avec $\# \in \{!, ?\}$, on pose $Cpre_\delta(s) = (p, w_1, \ldots, w_{i-1}, cpre_\#(m, w_i), w_{i+1}, \ldots w_d)$. La fonction est $Cpre_\delta$ est calculable.

On définit la fonction $Cpre$ telle que :

$$Cpre(s) = \{s\} \;\cup\; \{cpre_\delta(s) \mid \delta = (p, op, q) \in \Delta\}$$

pour toute configuration $s \in S$. En pratique on utilisera $\mathrm{Min} \circ Cpre$.

Le lemme suivant montre que $Cpre$ remplie presque les conditions pour être une fonction $\underline{Cpre}$ dont un WSTS a besoin d'être équipé pour être effectif. La seule condition qui manque est celle de fournir une base minimale, parce que $\leq$ est décidable $\mathrm{Min} \circ Cpre$ remplie toutes les conditions.

Lemme 6.1.4. *Pour toute configuration $c = (q, w_1, \ldots, w_d)$ in S, l'ensemble $Cpre(s)$ est une base de $\uparrow pre_{\mathcal{S}}(\uparrow s)$.*

$$Cpre(s) = \{s\} \ \cup \ \{cpre_\delta(s) \mid \delta = (p, op, q) \in \Delta\}$$

Démonstration. La preuve est technique et découle de la définition de $cpre_!$ et $cpre_?$. On a besoin que pour tout $s \in S$, $\{s\} \subseteq Cpre(s)$ à cause de la définition de la relation de perte $\rightarrow_\lambda$. □

Théorème 6.1.5. *Un LCS équipé des fonctions $\underline{Cpre} = \mathrm{Min} \circ Cpre$ et $\underline{Reduce} = \mathrm{Min}$ est un WSTS effectif.*

Démonstration. Soit un LCS $\mathcal{C}_{lcs}$. On veut prouver que le système de transitions $\mathcal{S} = (S, Init, \rightarrow)$, muni de l'ordre $\leq$, associé à ce LCS et équipé des fonctions $\mathrm{Min} \circ Cpre$ et Min est un WSTS effectif.

On prouve que $\leq$ est compatible avec $\rightarrow$. Soit trois configurations s, s' and t telles que $s \rightarrow s'$ et $s \leq t$. D'après le Lemme 6.1.2, on a que $t \xrightarrow{*} s$. Et donc $t \xrightarrow{*} s \rightarrow s' = t'$ ce qui finit la preuve que $\leq$ est compatible avec $\rightarrow$.

De plus on a :

— $\leq$ est décidable.

— pour un ensemble fini d'états $X \subseteq S$, on a bien que $\mathrm{Min}(X)$ est une base minimale de $\uparrow X$. De plus, parce qu'elle est canonique, pour deux ensembles finis d'états $X, Y \subseteq S$, si $\uparrow X = \uparrow Y$ alors $\mathrm{Min}(X) = \mathrm{Min}(Y)$.

— Lemme 6.1.4 montre que pour tout état $s \in S$, $\mathrm{Min}(Cpre(s))$ est une base minimale de $\uparrow pre_{\mathcal{S}}(\uparrow s)$.

— on peut décider si un état $s \in S$ est dans $\downarrow Init$ ou non.

Ce qui conclut la preuve. □

On peut donc notamment utiliser les techniques pour WSTS vues en première partie. On remarque notamment que les LCS sont des WSTS finis par le bas (pour un état $s \in S$, $\downarrow s$ est fini) et donc la méthode IC3 peut être utilisée. Dans la suite, on utilisera l'Algorithme 3 qui consiste en une analyse en arrière paramétrée avec des invariants de couverture pour élaguer certains états. Le Chapitre 8, le suivant, présentera trois invariants de couverture pouvant être utilisés avec cet algorithme.

6.2 Ensemble de couverture

En général, l'ensemble des états couvrables d'un LCS ne peut être calculé Mayr [2003] même si une représentation via des clos par le bas est toujours possible.

Si on veut résoudre le problème de couverture dans les LCS avec l'Algorithme 3, il nous faudra des invariants calculés en avant qui sur-approximent l'ensemble des configurations couvrables. Ces invariants doivent avoir un moyen

de représenter des ensembles potentiellement infinis. Vu qu'il y a un nombre de localités fini, la difficulté vient du contenu des canaux. Il y a d tuples de mots sur l'alphabet fini M. Une approche naturelle serait d'utiliser des sous-ensembles reconnaissables de $(M^*)^d$ pour représenter les contenus potentiellement infinis des canaux. C'est une limitation raisonnable vu que l'ensemble de couverture d'un LCS, qui est son invariant de couverture le plus précis, est régulier Abdulla et Jonsson [1996]; Cécé *et al.* [1996]. Si nous cherchons seulement des invariants clos par le bas, qui sont tous des invariants de couverture, on peut se concentrer sur le problème de représenter des langages reconnaissables clos par le bas. Ces langages peuvent être représentés par des *expressions régulières simples* Abdulla *et al.* [1998], ou en anglais SRE pour *simple regular expressions*.

6.2.1 Expressions régulières simples (SRE)

Étant donné un alphabet M on définit les expressions régulières simples comme suit. Un *atome* est soit $(m + \varepsilon)$ pour un message $m \in M$, soit $(m_1 + \cdots + m_n)^*$ avec $m_1, \ldots, m_n$ n messages de M avec $n \geq 1$. Un *produit* est une concaténation finie $a_1 \cdots a_n$ d'atomes. Une *expression régulière simple* (SRE) est une somme finie $p_1 + \cdots + p_n$ de produits.

Le *langage d'une SRE* r est son langage régulier associé et sera noté $[\![r]\!]$. Le lemme suivant nous donne une raison d'utiliser des SRE pour représenter les langages reconnaissables clos par le bas.

Lemme 6.2.1 (Abdulla *et al.* [1998])**.** *Un langage $L \subseteq M^*$ est clos par le bas si, et seulement si, il peut être représenté par une SRE.*

En pratique, pour l'efficacité de l'implémentation, on ne veut pas manipuler de SRE avec des doublons, par exemple $(a + b)^*$ et $(a + b)^* \cdot (a + \varepsilon)$ représentent le même langage, mais on préférera le premier car sa représentation est la plus petite possible. On dit d'un produit $p = e_1 \cdots e_n$ qu'il est *en forme normale* Abdulla *et al.* [1998] lorsque pour tout $1 \leq i \leq n$, on a $[\![e_i \cdot e_{i+1}]\!] \not\subseteq [\![e_{i+1}]\!]$ et $[\![e_i \cdot e_{i+1}]\!] \not\subseteq [\![e_i]\!]$. Une SRE $p_1 + \ldots + p_n$ est *en forme normale* si tous ses produits p_i sont en forme normales et s'il n'y a pas deux produits p_i et p_j avec $i \neq j$ telle que $[\![p_i]\!] \subseteq [\![p_j]\!]$.

Lemme 6.2.2 (Abdulla *et al.* [1998])**.** *Pour tout SRE r, il y a une unique SRE (à la commutativité de + près) en forme normale, qui est noté* $\mathrm{nf}(r)$*, telle que* $[\![\mathrm{nf}(r)]\!] = [\![r]\!]$*. De plus,* $\mathrm{nf}(r)$ *peut être calculée depuis r en temps quadratique.*

6.3 État de l'art

6.3.1 Accélérations de boucles (TReX)

L'outil TReX Annichini *et al.* [2001] permet de résoudre le problème de couverture dans les LCS. C'est un algorithme qui ne termine pas forcément lorsque la configuration cible n'est pas couvrable. La méthode calcule en avant des configurations abstraites $c = (q, r_1, r_2, \ldots, r_d)$ constitués d'une localité $q \in Q$ et de SRE r_i pour représenter le contenu du canal i pour $i \geq 0$. Comme nous l'avons vu à la Section 6.2, en utilisant seulement des SRE on ne peut pas, en général, calculer l'ensemble de couverture. Cependant on peut chercher à en calculer une sous-approximation la plus complète possible. Par exemple dans le LCS du bit alterné de la Figure 6.1, on peut calculer les SRE en avant à partir de la configuration $c_{init} = (q_0 \times r_0, \varepsilon, \varepsilon)$ en appliquant les transitions. Ce calcul peut faire l'exécution abstraite suivante à partir de la boucle $q_0 \times r_0 \xrightarrow{1!B_1} q_0 \times r_0$.

$$c_{init} \xrightarrow{1!B_1} (q_0 \times r_0, (B_1 + \varepsilon), \varepsilon) \xrightarrow{1!B_1} (q_0 \times r_0, (B_1 + \varepsilon) \cdot (B_1 + \varepsilon), \varepsilon) \xrightarrow{1!B_1} \ldots$$

On voit bien que le calcul ne va pas terminer. Les auteurs Abdulla *et al.* [1998, 1999] ont introduit les SRE dans le but de résoudre ce problème. En effet on sait qu'on peut représenter le résultat de la boucle $q_0 \times r_0 \xrightarrow{1!B_1} q_0 \times r_0$ par $(q_0 \times r_0, B_1^*, \varepsilon)$. Le même effet peut avoir lieu avec des boucles qui utilisent plusieurs règles. L'outil TReX repère des boucles et cherche à représenter ce que ces boucles changent aux contenus de canaux quand on les exécute plusieurs fois ou à l'infini.

On considère comment appliquer de nombreuse fois une concaténation d'opérations $ops = op_1 \cdot op_2, \cdots, op_n$, sur un produit p. Il y a deux cas (Lemma 7 Abdulla *et al.* [1998]). Soit ce n'est pas possible de répéter ces opérations plus de $m \geq 0$ fois. Soit il existe un produit p' et un entier $m \geq 0$ tel que $p' = \bigcup_{j \geq m} post(p, ops^j)$. On peut donc représenter l'effet d'une boucle. Les auteurs donnent une preuve constructive de leur lemme. On donne des exemples d'application du lemme :

- à partir du produit $p = (a + \varepsilon)$ on ne peut exécuter la concaténation d'opérations $ops = ?a!b$ qu'une unique fois. On a $post(p, ops^2) = \emptyset$ et $post(p, ops^*) = (a + \varepsilon) \cup (b + \varepsilon)$.
- à partir du produit $p = (c + \varepsilon) \cdot (a + b)^*$ on peut exécuter une infinité de fois $ops = ?a!c?b$. On obtient $p' = \bigcup_{j \geq n} post(p, ops^j) = (a + b)^* \cdot c^*$ avec $n = 1$ et $post(p, ops^*) = (c + \varepsilon) \cdot (a + b)^* \cup (a + b)^* \cdot c^*$.
- à partir du produit $p = b^*$ on peut exécuter une infinité de fois $ops = !a?a!a$, on a $p' = \bigcup_{j \geq n} post(p, ops^j) = a^*$ avec $n = 1$ et $post(p, ops^*) = b^* \cup a^*$.
- à partir du produit $p = (a+\varepsilon)$ et de $ops = ?a!a$ on a $p' = \bigcup_{j \geq n} post(p, ops^j) = (a + \varepsilon)$ avec $n = 0$ et $post(p, ops^*) = p'$.

— à partir du produit $p = (a + \varepsilon) \cdot (b + \varepsilon)$ on ne peut exécuter qu'une fois $ops = ?a?b!b!a$. On a donc $post(p, ops^2) = \emptyset$ et $post(p, ops^*) = (a + \varepsilon) \cdot (b + \varepsilon) \cup (b + \varepsilon) \cdot (a + \varepsilon)$.

L'application de ces opérations peut être répétée en cherchant des boucles ou de simples opérations. L'algorithme va s'arrêter s'il trouve une SRE r telle que la configuration cible c_{final} est dans le langage qu'elle représente. Il peut ne pas terminer lorsque c_{final} n'est pas couvrable. L'algorithme terminera lorsqu'il n'y aura plus d'opérations à explorer ou qu'il aura trouvé un point fixe ne contenant pas c_{final}.

6.3.2 Abstract Regular Model Checking

Dans le cas où les communications n'ont pas de perte, les SRE ne permettent pas de représenter exactement les contenus de canaux. Ces contenus peuvent être des langages reconnaissables et donc on peut vouloir utiliser des automates pour les représenter. À partir de d automates $A_1, A_2, \ldots A_d$, on peut utiliser un unique automate A tel $L(A) = L(A_1)\#L(A_2)\ldots\#L(A_d)$ avec $\#$ un nouveau message dans l'alphabet. On peut par exemple utiliser les *Queue-content Decision Diagrams* (QDD) Boigelot et Godefroid [1999].

On peut utiliser des automates pour décider si une configuration est accessible ou non dans un système de canaux sans perte, mais on ne peut pas à la fois calculer de manière exacte, sans sur-approximation, et à la fois terminer à chaque instance. Il faut donc trouver des moyens d'accélérer le calcul en perdant de l'information.

On peut définir une suite d'automates $A_0, A_1, \ldots$ telle que :

$$\begin{aligned} L(A_0) &= Init \\ A_{k+1} &= \alpha(post(A_k) \cup A_k) \end{aligned}$$

avec α une fonction d'approximation telle que $L(A) \subseteq L(\alpha(A))$ pour tout automate A. Cette suite peut converger ou non.

Pour cette fonction α on peut utiliser diverses techniques. On peut par exemple commencer par répartir les états en quatre ensembles Gall *et al.* [2006] : les états initiaux et finaux, les états initiaux mais pas finaux, les états finaux mais pas initiaux et les autres. Si cela n'est pas suffisant, on peut ensuite raffiner cette abstraction, par exemple en considérant deux états équivalents s'ils reconnaissent les mêmes mots de longueurs au plus $k \geq 0$. De manière générale on peut considérer les bisimulations Gall *et al.* [2006] et les classes d'équivalences sur les états Bouajjani *et al.* [2004].

Chapitre 7

Invariants pour systèmes de canaux à pertes

Dans ce chapitre, nous présentons trois invariants de couverture pour les LCS. Un invariant est une sur-approximation de l'ensemble des configurations accessibles et un invariant de couverture est un invariant qui contient aussi l'ensemble des configurations couvrables. Nous allons présenter une variante de l'inéquation d'état adaptée aux LCS, puis deux nouveaux invariants cherchant à comprendre des propriétés sur l'ordre dans lequel les messages peuvent se trouver, les *CSRE* et les *MOF*. Ces trois invariants ont été publiés dans le même article Geffroy *et al.* [2017b].

Ces invariants permettent de diminuer la taille de la suite (A_k) qui est calculée dans la version classique de l'algorithme de couverture en arrière comme expliqué dans le Chapitre 1 à la Section 2.1. Nous avons vu avec les réseaux de Petri que cela permettait d'accélérer la décision du problème de couverture. Un invariant doit être suffisamment précis, pour pouvoir réduire la taille des ensembles calculés, mais ne doit pas trop ralentir le calcul en arrière. Il y a deux coûts en temps associés aux invariants. Le premier est celui du test d'appartenance. L'algorithme vu au Chapitre 2 a en effet besoin à la Ligne 7, $P \leftarrow N \cap I$, de tester si un état est dans l'invariant ou non. Le second coût est celui du précalcul. Un invariant a en pratique toujours besoin d'un calcul à faire avant de commencer l'algorithme. Ce temps peut être négligeable, comme il peut être problématique. Un invariant se doit donc de résoudre un compromis, il faut qu'il soit suffisamment précis pour éliminer des éléments mais ne doit pas être trop coûteux à calculer.

Nous présenterons un invariant qui est une adaptation de l'inéquation d'état pour les LCS. Cette sur-approximation a pour effet d'oublier l'ordre des messages dans les canaux. Elle permet toutefois de compter combien il y a de messages de chaque type dans chaque canal pour chaque localité. Parce que l'ordre dans lequel les messages sont envoyés est important, nous présentons ensuite les expressions régulières simples et compactes, ou en anglais CSRE

pour *compact simple regular expressions*. L'idée est de se baser sur les SRE, qu'on a présentées au chapitre précédent, pour construire un invariant qui regarde dans quel ordre les messages sont envoyés. Cet invariant est une sur-approximation, mais son calcul termine. Les *CSRE* peuvent représenter des propriétés du type : on ne peut pas avoir un message a après un message b dans le canal 2. Nous montrerons que cet invariant peut souffrir d'une explosion combinatoire. C'est pourquoi nous présentons un troisième et dernier invariant, qui a le même objectif que les CSRE mais sans le problème de l'explosion combinatoire. Il s'agit des *MOF*, pour *message ordering flows*, qui sont décrits par un préordre sur les messages.

La structure du chapitre est simple, il sera composé de trois sections, une pour chacun des trois invariants. Nous commencerons par l'inéquation d'état, puis les CSRE et enfin les MOF.

7.1 Inéquation d'état

Nous commençons par présenter un invariant qu'on appelle inéquation d'état même si c'est une version modifiée pour les besoins spécifiques des LCS. L'inéquation d'état, que nous avons vue dans le Chapitre 3, a été utilisée avec succès par Esparza *et al.* [2014]; Blondin *et al.* [2016]; Geffroy *et al.* [2016a] pour les réseaux de Petri dans diverses méthodes et notamment dans l'idée de l'utiliser comme invariant dans l'Algorithme 3. Nous avons vu dans le Chapitre 5, consacré aux expérimentations, que cet invariant bien que simple était très efficace. Pour les LCS, l'intérêt de cet invariant est de pouvoir avoir des informations du type : on peut avoir au plus trois messages a dans le canal 1 et au plus un message b dans le canal 2 mais aucun autre message lorsqu'on est dans la localité q. Toutefois cet invariant oublie complètement l'ordre dans lequel les messages ont été envoyés : par exemple, il ne distingue pas le cas où avoir le mot $a \cdot b$ est possible mais pas le mot $b \cdot a$.

L'inéquation d'état est une conjonction d'inégalités composées de contraintes de localités et de contraintes de canaux. Pour un LCS et une configuration $c_{final} = (q_{final}, w_1, \ldots, w_d)$, un système d'inégalités doit être satisfait si c_{final} est couvrable. Ce système est défini sur un vecteur $\mathbf{x}$ de variables libres dans $\mathbb{Z}^{\Delta}$. On rappelle que Δ est l'ensemble des règles. Intuitivement, $\mathbf{x}(\delta)$ indique le nombre de fois où on a exécuté la règle δ pour couvrir la configuration c_{final}.

On commence par construire un système d'équations sur $\mathbf{x}$, qu'on a appelée *contraintes de localités* qui compte pour chaque localité, combien de fois on y est entré et sorti dans une même exécution. Pour cela on introduit le vecteur $\mathbf{e}_i$ dans $\mathbb{Z}^{Q}$ défini avec des zéros partout sauf pour l'indice i où il vaut un. On appelle $LC_{c_{final}}(\mathbf{x})$ le système d'équations suivant :

$$\mathbf{e}_{q_{init}} + \sum_{\delta=(p,op,q)\in\Delta} \mathbf{x}(\delta)(-\mathbf{e}_p + \mathbf{e}_q) = \mathbf{e}_{q_{final}} \tag{7.1}$$

On construit ensuite un système d'inéquations sur $\mathbf{x}$, qu'on appelle *contraintes de canaux* qui compte le nombre de fois chaque message m est reçu ou envoyé dans chaque canal i. Pour cela on introduit la matrice $\mathbf{e}_{i,m}$ définie comme une matrice de $\mathbb{Z}^{\{1,\dots,d\}\times M}$ qui vaut zéro partout sauf à l'indice i, m où la matrice vaut un. Étant donné un mot w de messages, on note $|w|_m$ le nombre d'occurrence de m dans w. On appelle $CC_{c_{final}}(\mathbf{x})$ le système d'équations suivant, où les inégalités $\geq$ sur les matrices sont définies composante par composante :

$$\sum_{\delta=(p,i!m,q)\in\Delta} \mathbf{x}(\delta)\mathbf{e}_{i,m} - \sum_{\delta=(p,i?m,q)\in\Delta} \mathbf{x}(\delta)\mathbf{e}_{i,m} \geq \sum_{\substack{m\in M \\ i\in\{1,\dots,d\}}} |w_i|_m \mathbf{e}_{i,m} \tag{7.2}$$

Enfin, on introduit le système d'inéquations $SI_{c_{final}}(\mathbf{x})$ défini par la conjonction $LC_{c_{final}}(\mathbf{x}) \wedge CC_{c_{final}}(\mathbf{x})$. La preuve du lemme suivant est immédiate en observant que si c_{final} est couvrable alors il existe une exécution de c_{init} vers c_{final}. En notant $\mathbf{x}(\delta)$ le nombre de fois où la règle δ est utilisée dans cette exécution, on obtient un vecteur $\mathbf{x}$ qui satisfait $SI_{c_{final}}(\mathbf{x})$.

Lemme 7.1.1. *Le système $SI_{c_{final}}(\mathbf{x})$ est satisfiable pour toute configuration couvrable c_{final}.*

Corollaire 7.1.2. *L'ensemble clos par le bas $I_{LCS} = \{c_{final} \mid \exists \mathbf{x}\ SI_{c_{final}}(\mathbf{x})\}$ est un invariant de couverture.*

Cet invariant peut donc être utilisé avec l'Algorithme 3. Nous verrons dans le Chapitre 8, le suivant, comment il se comporte en pratique.

7.2 Expressions régulières simples et compactes (CSRE)

Dans cette section nous présentons un invariant dont le but est de conserver des informations à propos de l'ordre dans lesquels les messages peuvent apparaître dans un canal. Contrairement à l'inéquation d'état, nous ne cherchons pas ici à compter le nombre de messages de chaque sorte dans chaque canal.

Même si des messages peuvent être perdus lors du calcul, tous les messages qui restent dans un canal sont dans le même ordre que celui dans lequel ils ont été envoyés. Si un système envoie des messages a puis des messages b dans un canal, ce canal ne peut pas contenir un mot qui contient un message b suivi d'un message a. L'invariant que l'on propose dans cette section retient des propriétés du type : ce n'est pas possible d'avoir un message a après un message b.

On présente un moyen de calculer un invariant qui utilise une sous-classe des SRE, rappelées dans la Section 6.2.1, qu'on a appelée *expression régulière simple et compacte* Geffroy *et al.* [2017b], ou en anglais CSRE pour *compact*

$$\begin{array}{lcl} csre_1 & = & (a^* \cdot b^*) \ + \ (a^* \cdot c^*) \\ csre_2 & = & ((a+b)^*) \ + \ ((c+d)^* \cdot e^*) \end{array} \qquad \begin{array}{lcl} csre_3 & = & \varepsilon \\ csre_4 & = & \emptyset \end{array}$$

FIGURE 7.1 – Exemples d'expressions régulières simples et compactes

simple regular expressions. L'idée est d'abstraire le contenu des canaux avec des expressions régulières simples de la forme $(a^* \cdot b^*) + (a^* \cdot (c+d)^*)$. Formellement, une *expression régulière simple et compacte* (CSRE) est une SRE $\sum p_i$ où chaque produit p_i est un produit compact. Un *produit compact* est de la forme $e_1 \cdots e_n$ avec chaque atome a_i de la forme $(m_1 + \cdots + m_n)^*$ et tel que dans chaque paire d'atomes e_i et e_j, avec $i \neq j$, les messages sont différents (i.e., $[\![e_i]\!] \cap [\![e_j]\!] = \{\varepsilon\}$). Voir Exemple 7.1. On remarque que tous les produits compacts sont en forme normale. Toutefois, tous les CSRE ne sont pas en forme normales. Par exemple, $(a+b)^* + a^*$ n'est pas en forme normale. On note $CSRE_{nf}$ l'ensemble des CSRE en forme normale. Cet ensemble est fini. Tous les CSRE de l'Exemple 7.1 sont en forme normale.

On définit la relation binaire $\sqsubseteq$ sur $CSRE$ tel que $r_1 \sqsubseteq r_2$ lorsque $[\![r_1]\!] \subseteq [\![r_2]\!]$. Cette relation est réflexive et transitive et donc c'est un préordre. La relation $\sqsubseteq$ est aussi antisymétrique pour les CSRE en forme normale, donc $\sqsubseteq$ est un ordre partiel sur $CSRE_{nf}$. On observe que $\sqsubseteq$ n'est pas antisymétrique sur la classe complète de CSRE (par exemple, $[\![(a+b)^*]\!] = [\![(a+b)^* + a^*]\!]$).

On va exhiber une connexion de Galois entre $(\mathcal{P}(M^*), \subseteq)$ et $(CSRE_{nf}, \sqsubseteq)$. Les deux lemmes suivants sont nécessaires pour cela.

Lemme 7.2.1. *Soit deux sous-ensembles de messages $A_1, A_2 \subseteq M$. Pour tous langages $L_1 \subseteq (M \setminus A_1)^*$ et $L_2 \subseteq (M \setminus A_2)^*$, on a $(A_1^* \cdot L_1) \cap (A_2^* \cdot L_2) = (A^* \cdot (L_1 \cap B_2^* \cdot L_2)) \cup (A^* \cdot (L_2 \cap B_1^* \cdot L_1))$ avec $A = A_1 \cap A_2$, $B_1 = A_1 \backslash A_2$ et $B_2 = A_2 \backslash A_1$.*

Démonstration. On prouve seulement le sens $\subseteq$ car l'autre inclusion est immédiate. Soit un mot $w \in (A_1^* \cdot L_1) \cap (A_2^* \cdot L_2)$. Il existe un mot $u_1 \in A_1^*$, $v_1 \in L_1$, $u_2 \in A_2^*$ et $v_2 \in L_2$ tel que $w = u_1 \cdot v_1 = u_2 \cdot v_2$. On suppose que la taille du mot u_1 est plus petite ou égale à celle de u_2. Il existe un mot u_2' tel que $u_2 = u_1 \cdot u_2'$. On a donc ensuite que $u_1 \in (A_1 \cap A_2)^* = A^*$. On observe qu'on a aussi $v_1 = u_2' \cdot v_2$. Parce que $v_1 \in L_1 \subseteq (M \setminus A_1)^*$, on sait que $u_2' \in (A_2 \backslash A_1)^* = B_2^*$, et donc que $v_1 \in (B_2^* \cdot L_2)$. On a montré que $w \in (A^* \cdot (L_1 \cap B_2^* \cdot L_2))$. De manière symétrique, si la longueur du mot u_1 est plus grande ou égale à celle du mot u_2, alors $w \in (A^* \cdot (L_2 \cap B_1^* \cdot L_1))$. □

Lemme 7.2.2. *L'ensemble des langages représentables par une CSRE est clos par intersection.*

Démonstration. On montre que l'intersection de deux langages représentés par deux produits compacts peut être représentée par une CSRE. La preuve du

$$a^* \cdot b^* \cap b^* \cdot a^* \;=\; a^* \cup b^* \qquad\qquad a^* \cdot b^* \cap (a+b)^* \;=\; a^* \cdot b^*$$

FIGURE 7.2 – Exemples d'intersections de produits compacts.

lemme en découle par distributivité de l'intersection sur l'union et par clôture des CSRE pour l'addition. Pour un produit compact $p = a_1 \cdots a_n$, on note $|p|$ le nombre n d'atomes qui composent p. On prouve par induction sur $k \geq 0$, que pour tous produits compacts p_1 et p_2 tels que $|p_1| + |p_2| \leq k$, il existe une CSRE r telle que $[\![p_1]\!] \cap [\![p_2]\!] = [\![r]\!]$. Pour le cas de base, on a $p_1 = p_2 = \varepsilon$ et $[\![p_1]\!] \cap [\![p_2]\!] = [\![\varepsilon]\!]$. Pour l'étape d'induction, soit p_1 et p_2 deux produits compacts tel que $|p_1| + |p_2| = k + 1$. Si l'un des deux est égal à ε alors $[\![p_1]\!] \cap [\![p_2]\!] = [\![\varepsilon]\!]$. Sinon, p_1 et p_2 peuvent être décomposés comme les produits compacts $p_1 = A_1^* \cdot p_1'$ et $p_2 = A_2^* \cdot p_2'$ avec A_1 et A_2 deux sous-ensembles de M. On remarque que $[\![p_1']\!] \subseteq (M \setminus A_1)^*$ et $[\![p_2']\!] \subseteq (M \setminus A_2)^*$. On en déduit avec le Lemme 7.2.1 que $[\![p_1]\!] \cap [\![p_2]\!] = (A^* \cdot ([\![p_1']\!] \cap [\![B_2^* \cdot p_2']\!])) \cup (A^* \cdot ([\![p_2']\!] \cap [\![B_1^* \cdot p_1']\!]))$ où A, B_1 et B_2 sont définis comme dans le lemme. On remarque que $|p_1'| + |B_2^* \cdot p_2'| = |p_2'| + |B_1^* \cdot p_1'| = k$. D'après l'hypothèse d'induction il existe deux CSRE r_1 et r_2 telles que $[\![r_1]\!] = ([\![p_1']\!] \cap [\![B_2^* \cdot p_2']\!])$ et $[\![r_2]\!] = ([\![p_2']\!] \cap [\![B_1^* \cdot p_1']\!])$. On obtient que $[\![p_1]\!] \cap [\![p_2]\!] = [\![A^* \cdot r_1 + A^* \cdot r_2]\!]$. Il est facile de vérifier que $[\![r_1]\!]$ et $[\![r_2]\!]$ sont contenus dans $(M \setminus A)^*$. Et donc, $A^* \cdot r_1$ et $A^* \cdot r_2$ peuvent être réécrit comme des CSRE par distributivité du produit sur la somme. □

Parce que tout *CSRE* peut être réduit en un $CSRE_{nf}$, $CSRE_{nf}$ est clos par l'union et par l'intersection.

Voir les exemples d'intersections de CSRE dans l'Exemple 7.2.

On introduit la fonction de concrétisation $\gamma : CSRE_{nf} \to \mathcal{P}(M^*)$ définie par $\gamma = [\![\cdot]\!]$. La fonction d'abstraction $\alpha : \mathcal{P}(M^*) \to CSRE_{nf}$ est définie par $\alpha(L) = \bigcap\{r \in CSRE_{nf} \mid L \subseteq [\![r]\!]\}$. La fonction α est bien définie car $CSRE_{nf}$ est clos par intersection, d'après le Lemme 7.2.2, et est fini. On note que $\alpha(L)$ est calculable quand L est un langage régulier. On remarque que (α, γ) est une connexion de Galois entre $(\mathcal{P}(M^*), \subseteq)$ et $(CSRE_{nf}, \sqsubseteq)$. Pour pouvoir appliquer le cadre général de l'interprétation abstraite Cousot et Cousot [1977], il nous reste à montrer qu'on peut faire les opérations abstraites sur les canaux, aussi bien les émissions que les réceptions.

Pour un langage $L \subseteq M^*$, soit $m^{-1}L = \{w \subseteq M^* \mid m \cdot w \in L\}$. En suivant le cadre classique de l'interprétation abstraite, une émission abstraite est définie par $(!m)^\#(r) = \alpha([\![r]\!] \cdot m)$, une réception abstraite est définie par $(?m)^\#(r) = \alpha(m^{-1}[\![r]\!])$ et une perte abstraite est définie par $(\lambda)^\#(r) = r$. Montrons comment calculer $(!m)^\#(r)$ et $(?m)^\#(r)$ à partir d'une CSRE r en forme normale. On a seulement besoin de regarder le cas où r est un produit compact car $(!m)^\#(p_1 + \cdots + p_n) = \mathrm{nf}((!m)^\#(p_1) + \cdots + (!m)^\#(p_n))$, et de même pour $(?m)^\#$.

Pour un produit compact $p = a_1 \cdots a_n$, on note $merge(a_1 \cdots a_n)$ l'atome qui contient exactement les messages qui apparaissent dans $a_1, \ldots, a_n$. Par exemple $merge((a+b)^* \cdot c^* \cdot (d+e)^*) = (a+b+c+d+e)^*$. Formellement, $merge(a_1 \cdots a_n)$ est l'atome A^* où $A = \{m \in M \mid m \in [\![a_1]\!] \cup \cdots \cup [\![a_n]\!]\}$. Pour un produit compact $p = a_1 \cdots a_n$ et un message $m \in M$ on a :

$$(!m)^\#(p) \;=\; \begin{cases} a_1 \cdots a_{k-1} \cdot merge(a_k \cdots a_n) & \text{si } m \in [\![p]\!] \\ p \cdot m^* & \text{sinon} \end{cases}$$

$$(?m)^\#(p) \;=\; \begin{cases} a_k \cdots a_n & \text{si } m \in [\![p]\!] \\ \emptyset & \text{sinon} \end{cases}$$

où, dans les deux cas, k est l'unique indice tel que $m \in [\![a_k]\!]$.

Par une construction standard, on peut étendre la connexion de Galois (α, γ) en une connexion de Galois entre $(\mathcal{P}(S), \subseteq)$ et le domaine abstrait qui consiste en l'ensemble des applications de Q vers $CSRE^d_{nf}$, partiellement ordonné par l'extension composante par composante de $\sqsubseteq$. Et ainsi nous pouvons utiliser le cadre général de l'interprétation abstraite Cousot et Cousot [1977] pour calculer un invariant clos par le bas, qui est donc un invariant de couverture, grâce à un calcul de point fixe sur ce domaine abstrait. Puisque l'ensemble $CSRE_{nf}$ est fini, le calcul de point fixe est garanti de converger, et donc de terminer. On peut donc utiliser cet invariant de couverture pour accélérer l'Algorithme 3.

Pour comprendre le calcul de point fixe nous présentons son calcul étape par étape dans l'Exemple 7.2.3 pour le LCS simple décrit dans la Figure 7.3. Pour montrer que calcul est aussi utile pour des exemples non triviaux, nous montrons le point fixe calculé par cette méthode pour le LCS du protocole du bit alterné Bartlett *et al.* [1969], vu à la Section 6.1, dans l' Exemple 7.2.4.

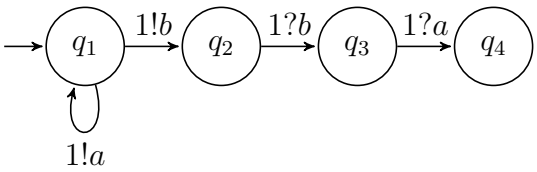

FIGURE 7.3 – Exemple simple de LCS où le calcul de point fixe avec les CSRE permet de vérifier que (q_4, ε) n'est pas couvrable.

Exemple 7.2.3. *Dans le LCS de la Figure 7.3, on peut effectuer le calcul de point fixe, c'est ce qui est décrit Figure 7.4. La configuration initiale est (q_1, ε), le point fixe commencera donc avec pour la localité q_1 la CSRE contenant dans l'unique canal un unique produit compact vide, représenté par ε. Pour toutes les autres localités $q \neq q_1$, il y aura dans le canal la CSRE vide, ne contenant*

itération	q_1	q_2	q_3	q_4
0	ε	$\emptyset$	$\emptyset$	$\emptyset$
1	a^*	$\emptyset$	$\emptyset$	$\emptyset$
2	a^*	$a^* \cdot b^*$	$\emptyset$	$\emptyset$
3	a^*	$a^* \cdot b^*$	b^*	$\emptyset$
4	a^*	$a^* \cdot b^*$	b^*	$\emptyset$

FIGURE 7.4 – Itérations du calcul de point fixe avec des CSRE pour le LCS de la Figure 7.3.

aucun produit compact, qui représente le langage vide. À la première itération, on exécute de manière abstraite l'opération $!a$ *et on a* $(!a)^{\#}(\varepsilon) = a^*$*. La seconde itération rappelle que l'analyse ne compte pas les messages. Il est évident pour le lecteur humain qu'il ne peut y avoir qu'un unique message* b *dans* q_2 *mais on aura l'opération abstraite :* $(!b)^{\#}(a^*) = a^* \cdot b^*$*. La troisième itération est intéressante car elle permet de voir la précision du calcul abstrait. L'analyse commence avec* $a^* \cdot b^*$ *dans* q_2 *mais va* comprendre *qu'il faut perdre tous les messages* a *qui sont avant des messages* b *pour pouvoir aller dans la localité* q_3*. Formellement on a* $(?a)^{\#}(a^* \cdot b^*) = b^*$*. La quatrième itération n'arrivera pas à changer les CSRE. L'analyse a atteint un point fixe. Ce point fixe permet par exemple de montrer que la configuration* $c_{final} = (q_4, \varepsilon)$ *n'est pas couvrable.*

Exemple 7.2.4. *Dans le LCS de l'ABP Bartlett et al. [1969], donné par la Figure 1.4, la configuration initiale est* $(q_0 \times r_0, \varepsilon, \varepsilon)$*. Le point fixe commencera donc avec dans la localité* $q_0 \times r_0$ *la CSRE contenant dans chaque canal un unique produit compact vide, représentant* ε*. Et pour toutes les autres localités* $q \times r$*, dans chaque canal il y aura la CSRE vide, ne contenant aucun produit compact, qui représente le langage vide. Nous ne détaillerons pas le point fixe de toutes les localités. Pour la localité initiale* $q_0 \times r_0$ *on a la CSRE* B_1^* *dans le canal 1 et la CSRE* A_1^* *dans le canal 2. Dans la localité* $q_3 \times r_0$ *ce sont les CSRE* $B_0^* \cdot B_1^*$ *et* A_1^**. Cela permet de prouver certaines propriétés, par exemple le fait de ne jamais avoir de messages du type* $A_0 \cdot A_1 \cdot A_0$ *dans les canaux.* □

L'Exemple 7.2.5 montre un problème avec les CSRE : il peut y avoir une explosion combinatoire. Or dans notre approche en arrière, un invariant est utilisé pour accélérer le calcul. Si son calcul est long il échoue sa mission. Le problème se situe lors du précalcul, c'est-à-dire avant même de commencer l'analyse en arrière. Ce problème a été observé en pratique comme nous le verrons dans le chapitre suivant consacré aux expérimentations. Le but de la section suivante est de présenter un invariant qui n'ait pas ce problème mais qui soit quand même suffisamment précis pour être utilisé en pratique dans l'Algorithme 3.

Exemple 7.2.5. *Soit M l'alphabet fini qui contient les messages $x_1, y_1, z_1, \ldots, x_n, y_n, z_n$. Soit $L \subseteq M^*$ le langage*

$$L \;=\; {x_1}^* \cdot ({y_1}^* + {z_1}^*) \cdot {x_2}^* \cdot ({y_2}^* + {z_2}^*) \cdots {x_n}^* \cdot ({y_n}^* + {z_n}^*)$$

Ce langage peut être représenté par une CSRE. Mais la CSRE minimale r telle que $L = [\![r]\!]$ est la somme de tous les produits compacts de la forme ${x_1}^* \cdot {m_1}^* \cdot {x_2}^* \cdot {m_2}^* \cdots {x_n}^* \cdot {m_n}^*$ où $m_i \in \{y_i, z_i\}$ pour tout $i \in \{1, \ldots, n\}$. Ces produits compacts sont en nombre exponentiel . On remarque que r est en forme normale.

7.3 Flux de messages ordonnés

Soit M un alphabet fini d'un LCS pour lequel on veut calculer un invariant de couverture. Un *flux de messages ordonnés*, ou en anglais MOF pour *message ordering flow* est une paire (A, R) où $A \subseteq M$ et R est un préordre sur A. Intuitivement pour deux messages $a, b \in A$, si aRb alors le message b peut apparaître après le message a. On note MOF l'ensemble des MOF. On définit l'ordre partiel $\sqsubseteq$ sur MOF tel que $F_1 \sqsubseteq F_2$ avec $F_1 = (A_1, R_1)$ et $F_2 = (A_2, R_2)$ lorsque $A_1 \subseteq A_2$ et $R_1 \subseteq R_2$.

La borne supérieure de deux MOF $F_1 = (A_1, R_1)$ et $F_2 = (A_2, R_2)$ est $F_1 \sqcup F_2 = (A, R)$ où $A = A_1 \cup A_2$ et $R = (R_1 \cup R_2)^+$, c'est-à-dire la clôture transitive de $R_1 \cup R_2$. La borne inférieure de F_1 et F_2 est $F_1 \sqcap F_2 = (A, R)$ où $A = A_1 \cap A_2$ et $R = R_1 \cap R_2$.

Comme pour l'invariant basé sur les CSRE présenté à la section précédente, nous voulons exhiber une connexion de Galois entre $(\mathcal{P}(M^*), \subseteq)$ et $(MOF, \sqsubseteq)$. On appelle $\preceq$ l'ordre sous-mot sur M^*. On commence avec la fonction d'abstraction $\alpha : \mathcal{P}(M^*) \to MOF$. Pour un mot $w \in M^*$, $\alpha(w) = (A, R) \in MOF$ tel que $A = \{m \in M \mid m \preceq w\}$ est l'ensemble de tous les messages qui apparaissent dans w et $R = \{(a, b) \in A \times A \mid a = b \vee ab \preceq w\}$. Pour un langage non vide $L \subseteq M^*$, on définit $\alpha(L)$ comme la borne supérieure $\sqcup$ de $\{\alpha(w) \mid w \in L\}$. La fonction de concrétisation $\gamma : MOF \to \mathcal{P}(M^*)$ est définie par $\gamma(A, R) = \{w \in A^* \mid \forall ab \preceq w, (a, b) \in R\}$. On observe que $\gamma(A, R) = \bigcup\{L \subseteq M^* \mid \alpha(L) \sqsubseteq (A, R)\}$.

Il est aisé de voir que la concrétisation d'un MOF est un langage clos par le bas. Mais pour l'instant l'ensemble $\gamma(MOF)$ des langages qui peuvent être représentés par un MOF ne contient pas le langage vide, qui est pourtant essentiel. On étend donc l'ensemble MOF avec un nouvel élément, noté $\bot$, dont la concrétisation est $\gamma(\bot) = \emptyset$. Dans l'autre sens, l'abstraction du langage vide est $\alpha(\emptyset) = \bot$. L'ordre $\sqsubseteq$ est étendue de manière naturelle : pour tout MOF F, on a $\bot \sqcup F = F$ et $\bot \sqcap F = \bot$.

Par construction, (α, γ) est une connexion de Galois entre $(\mathcal{P}(M^*), \subseteq)$ et $(MOF, \sqsubseteq)$. Nous devons, comme pour les CSRE, montrer que

nous pouvons calculer les opérations abstraites sur les canaux, aussi bien les émissions que les réceptions.

En suivant le cadre usuel pour l'interprétation abstraite on définit $(!m)^\# : MOF \to MOF$ et $(?m)^\# : MOF \to MOF$ par $(!m)^\#(F) = \alpha(\gamma(F) \cdot m)$ et $(?m)^\#(F) = \alpha(m^{-1} \cdot \gamma(F))$. Montrons que les fonctions $(!m)^\#$ et $(?m)^\#$ sont calculables.

- $(!m)^\#(\bot) = (?m)^\#(\bot) = \bot$.
- $(!m)^\#(A, R) = (A', R')$ où $A' = A \cup \{m\}$ et $R' = R \cup (A \times \{m\}) \cup (B \times B)$ où $B = \{b \in A \mid (m, b) \in R\}$ est l'ensemble de tous les messages b qui apparaissent après m.
- $(?m)^\#(A, R) = \bot$ si $m \notin A$, sinon $(?m)^\#(A, R) = (A', R')$ où $A' = \{b \in A \mid (m, b) \in R\}$ est l'ensemble de tous les messages b qui apparaissent après m et $R' = R \cap (A' \times A')$.

Comme pour les CSRE, on peut étendre la connexion de Galois (α, γ) en une connexion de Galois entre $(\mathcal{P}(S), \subseteq)$ et le domaine abstrait consistant en l'ensemble des applications de Q vers MOF^d, qui sont partiellement ordonnées par $\sqsubseteq$ pour chaque composante. De la même façon, on peut générer un invariant clos par le bas et inductif avec un calcul classique de point fixe sur ce domaine abstrait. Comme l'ensemble MOF est fini, le calcul de point fixe est assuré de converger. On pourra utiliser cet invariant de couverture avec l'Algorithme 3 décrit au Chapitre 2.

L'ensemble des langages représentés par des MOF est inclus dans celui des langages représentés par des CSRE. Par exemple $L = a^* \cdot b^* \cup b^* \cdot a^*$ peut être représenté par une CSRE mais l'abstraction de L sera $\gamma(L) = (A, R)$ avec $A = \{a, b\}$ et R le préordre où a et b sont équivalents. On aura donc que $\alpha(A, R) = (a + b)^*$. Les *CSRE* sont donc plus précis que les *MOF*. Toutefois, l'Exemple 7.3.1 montre que les *MOF* permettent d'éviter l'explosion combinatoire qu'on avait avec les CSRE sans forcément perdre en précision. Nous le verrons en pratique dans le prochain chapitre avec les expérimentations.

Exemple 7.3.1. *On reprend l'Exemple 7.2.5 qui illustrait l'explosion combinatoire possible avec les CSRE.*

On reprend le même langage L défini par

$$L \;=\; {x_1}^* \cdot ({y_1}^* + {z_1}^*) \cdot {x_2}^* \cdot ({y_2}^* + {z_2}^*) \cdots {x_n}^* \cdot ({y_n}^* + {z_n}^*)$$

Ce langage peut être représenté par le MOF $F = (A, R)$ avec A l'ensemble contenant exactement les messages $x_1, y_1, z_1, \ldots, x_n, y_n, z_n$ et R le préordre illustré dans la Figure 7.5, c'est-à-dire la clôture réflexive et transitive illustrée par les flèches dans la figure. Contrairement à la CSRE minimale représentant L, qui était de taille exponentielle en n, la taille de F est quadratique en n. □

Nous disposons donc maintenant d'un invariant de couverture lié à l'ordre des messages qui peut être utilisé avec l'Algorithme 3. Le temps de calcul

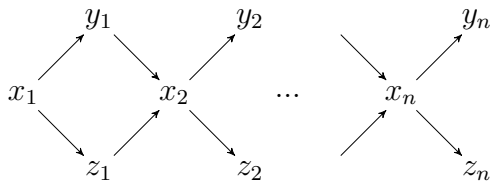

FIGURE 7.5 – MOF pour le langage L de l'Exemple 7.2.5.

de cet invariant est raisonnable tout comme l'est sa taille. Nous le verrons expérimentalement au prochain chapitre.

Chapitre 8

Expérimentations

Ce chapitre est consacré aux expérimentations sur le problème de couverture pour les LCS. Nous avons présenté dans le dernier chapitre trois invariants de couverture. Nous les utiliserons avec l'Algorithme 3. Nous avons déjà vu que cette méthode était efficace pour les réseaux de Petri, nous montrons qu'elle l'est aussi pour les LCS.

Les trois invariants de couverture sont l'inéquation d'état, les *CSRE* et les *MOF*. Nous avons développé en `OCaml` un nouvel outil, `BML`, qui implémente l'Algorithme 3 pour les LCS. Il est paramétré par des invariants et les trois invariants ont été implémentés. Nous allons voir que l'inéquation d'état et les *MOF* permettent d'accélérer fortement le calcul. L'outil est disponible en téléchargement Geffroy *et al.* [2017a].

L'invariant *CSRE* a un problème d'explosion combinatoire sur certaines instances. Pour pouvoir résoudre le problème de l'appartenance à cet invariant, il faut précalculer un certain nombre d'informations avec un calcul de point fixe, comme nous l'avons vu au chapitre précédent. Ce précalcul peut être trop long comme nous allons le voir dans les résultats expérimentaux. Le temps qu'il met peut largement dépasser le temps mis par l'analyse en arrière. C'est pour cet invariant que nous avons présenté dans la Section 2.2.2 du Chapitre 2, l'heuristique de parallélisation, qui permet de ne pas ralentir l'analyse en arrière par le précalcul d'un invariant. Nous avons montré qu'utiliser un invariant après avoir déjà effectué plusieurs itérations de l'analyse en arrière n'a pas d'impact sur la taille des bases calculées. Nous illustrerons cet effet par une preuve de concept de cette heuristique.

Contrairement aux réseaux de Petri, nous n'avons pas pu nous comparer avec plusieurs outils. Il y a l'outil `TReX` Annichini *et al.* [2001], qui a été conçu pour résoudre le problème de couverture dans les LCS mais il n'est plus disponible. Ce n'est pas un algorithme complet, il peut ne pas terminer sur les instances non couvrables. Nous avons pu nous comparer avec l'outil `McScM` Heußner *et al.* [2012], même si celui-ci n'a pas été conçu pour les systèmes à pertes, il permet de les émuler via une option.

Nous commencerons par donner des détails sur l'outil `BML`, puis nous comparerons ses résultats avec l'outil `McScM`. Nous détaillerons l'effet en pratique des différents invariants que sont l'inéquation d'état (`SI`), *CSRE* et *MOF*. Enfin nous présenterons notre preuve de concept pour l'heuristique parallèle.

8.1 Implémentation

Nous avons implémenté la recherche en arrière avec invariants telle que décrite dans le Chapitre 2 et le calcul des trois invariants décrits dans le Chapitre 7. Ces trois invariants sont l'inéquation d'état (`SI`), les SRE compactes (*CSRE*) et les MOF. Cela a donné naissance à un outil, `BML`, qui peut être téléchargé Geffroy *et al.* [2017a]. Il a été écrit en `Ocaml` et utilise le SMT solveur `Z3` pour l'inéquation d'état. Si le but principal de l'outil était de tester l'Algorithme 3 et les invariants `SI`, *CSRE* et *MOF* sur les LCS, l'architecture de l'outil a tenu compte du caractère général de la méthode : l'algorithme principal, l'Algorithme 3, peut être rapidement réutilisé pour d'autres WSTS. De même l'interprétation abstraite est modulaire et seules les fonctions propres à *MOF* et *CSRE*, par exemple la réception ou l'émission abstraite ou encore l'union, sont différentes entre les deux invariants. L'outil lit des fichiers au format `scm` comme utilisé dans l'outil `McScM` Heußner *et al.* [2012].

8.2 Résultats pratiques

Il nous a été impossible de nous comparer à l'outil `TReX` Annichini *et al.* [2001] car celui-ci n'est plus disponible. Nous avons pu nous comparer avec `McScM` Heußner *et al.* [2012], même si cet outil n'est pas à la base fait pour les systèmes avec pertes comme les LCS, une option permet d'interpréter les fichiers comme des LCS.

Les exemples sont ceux utilisés par l'outil `McScM` Heußner *et al.* [2012]. Nous avons sélectionné 13 instances du problème de couverture. Ces instances sont d'origines et de difficultés diverses. Les exemples *BAwCC*, *BAwPC*, *BAwCC_enh* et *BAwPC_enh* sont ce qui s'appelle des *business protocols* et ont été modélisés par Ravn *et al.* [2011]. Les deux premiers sont des instances couvrables, c'est-à-dire non sûre, et les deux autres sont des versions améliorées des protocoles pour les rendre non couvrables et donc sûre. Parmi les autres exemples il y a deux versions du protocole de Peterson, un avec trois processus et l'autre avec quatre, une version du protocole `pop3`, un protocole `tcp` avec et sans une erreur volontairement ajoutée et pour finir un protocole en anneau.

La taille des exemples varie de quelques états et transitions pour les plus simples jusqu'à 164 états et 352 transitions pour le protocole *Peterson4*. On verra dans la suite que l'invariant `SI` arrive à résoudre cette instance en moins

TABLE 8.1 – Temps en secondes pour résoudre le problème de couverture pour chaque instance. La colonne *Meilleur* donne le plus petit temps parmis les quatre versions (avec un des trois invariants et sans invariant).

	$\emptyset$	SI	*CSRE*	*MOF*	Meilleur	McScM
Peterson3	947,6	1,9	21,5	22,1	1,9	9,3
Peterson4	>96h	2934,4	>96h	>96h	2934,4	erreur
pop3	157,9	0,3	>96h	0,9	0,3	0,6
BAwPC	0,4	4,3	0,0	0,0	0,0	0,1
BAwCC	1,9	11,1	0,1	0,1	0,1	0,1
BAwCC_enh	2,3	13,7	0,1	0,1	0,1	1,6
BAwPC_enh	0,7	3,5	0,0	0,0	0,0	0,2
BRP	0,2	0,1	0,3	0,3	0,1	1,2
server	0,1	1,3	0,6	0,4	0,1	5,0
server_err	0,0	0,0	0,0	0,0	0,0	0,0
ring2	0,2	0,2	0,4	0,4	0,2	2,6
tcp	0,0	0,0	0,0	0,0	0,0	0,1
tcp_err	0,0	0,1	0,0	0,0	0,0	0,0

d'une heure. On rappelle que le problème de couverture est hyper-Ackermanien dans les LCS.

Remarque. *Les instances du problème de couverture n'ont rarement qu'une seule configuration cible. Cela ne change pas le fonctionnement de l'Algorithme 3. Au lieu d'avoir* $B_0^I = \{c_{final}\} \cap I$*, on fait* $B_0^I = C_{final} \cap I$*, avec* C_{final} *l'ensemble des configurations cibles. Ce grand nombre de configurations cibles vient en partie du fait de prendre le produit cartésien : pour un système à deux automates, si la propriété est qu'il ne faut pas aller dans un état* q_{final} *du premier automate, alors toutes les configurations* $(q_{final} \times r, \varepsilon, \varepsilon, \ldots, \varepsilon)$ *avec* r *un état du second automate seront des configurations cibles.*

L'analyse en arrière classique a été capable de résoudre 12 instances sur 13. L'Algorithme 3 avec l'inéquation d'état a été capable de tous les résoudre, d'en résoudre 12 sur 13 avec *MOF* et seulement 11 sur 13 avec *CSRE*. L'outil McScM a 4 *moteurs de vérification*, le meilleur pour ces 13 instances a été le moteur `cegar`, qui a résolu 10 instances. La Table 8.1 compare les temps utilisés pour résoudre le problème de couverture avec aucun invariant, noté $\emptyset$, et avec les invariants SI, *CSRE* et *MOF*. Le meilleur temps est mis dans la cinquième colonne. Dans la dernière colonne, il est indiqué le temps utilisé par McScM avec le meilleur moteur pour chaque exemple. On peut voir que SI accélère énormément le calcul dans les plus gros exemples. L'invariant *MOF* a le même effet sauf pour le protocole *Peterson4* où *MOF* n'a pas terminé même après 96 heures.

Remarque. *L'implémentation du calcul de point fixe pour CSRE et MOF*

n'est pas optimale. Elle utilise une liste de toutes les localités possibles, or nous utilisons le produit cartésien. Par exemple pour l'exemple Peterson4, qui a 164 états répartis en 4 automates, il y a 2 825 761 localités différentes en utilisant le produit cartésien. L'implémentation est naïve et parcourt plusieurs fois une liste de cette taille pour mettre à jour ou non les CSRE ou les MOF correspondant à ces localités. □

Pour expliquer l'accélération en temps permise par les invariants, on peut regarder le nombre de configurations qui ont été manipulées par l'analyse en arrière. C'est le nombre de configurations cibles additionnées à chaque étape avec le nombre de configurations calculées avec $\underline{Cpre}$ à la ligne 6, $N \leftarrow \underline{Cpre}(B) \setminus {\uparrow}B$. Par exemple pour *Peterson3*, on voit que de 359131 configurations manipulées par la version sans invariant, on passe à seulement 2127 configurations.

TABLE 8.2 – Nombre de configurations manipulées par l'analyse en arrière.

	$\emptyset$	SI	*CSRE*	*MOF*
Peterson3	359131	2127	67	67
Peterson4	>5113486	852630	-	-
pop3	241595	813	-	17842
BAwPC	15266	15266	1587	1587
BAwCC	31693	31693	2480	2480
BAwCC_enh	38218	38218	59	59
BAwPC_enh	19193	12824	44	44
BRP	9343	470	9132	9343
server	8873	8873	7720	8873
server_err	105	1	1	1
ring2	11141	772	11141	11141
tcp	1519	37	1437	1519
tcp_err	1423	218	1290	1364

Nous expliquons dans la section suivante comment les invariants permettent de manipuler beaucoup moins de configurations. De plus nous les comparerons entre eux.

8.3 Précisions des invariants

Dans la section précédente, on a vu que les invariants présentés dans le Chapitre 7 permettaient de réduire l'analyse en arrière. On s'intéresse dans cette section à comparer les invariants entre eux. Comme pour les réseaux de Petri, une mesure intéressante est la précision des invariants. On veut savoir le pourcentage d'états qui sont éliminés par l'invariant. La Table 8.3 montre, pour chaque invariant, dans la colonne de gauche le nombre de fois où on a

demandé si une configuration était ou non dans l'invariant de couverture et dans la colonne de droite le pourcentage de fois où la réponse était non, et donc où on a pu élaguer la configuration.

TABLE 8.3 – Nombre de configurations testées et pourcentage de configurations élaguées.

	SI		*CSRE*		*MOF*	
	#	%	#	%	#	%
Peterson3	1970	72,2	67	100,0	67	100
Peterson4	745009	72,0	-	-	-	-
pop3	324	61,8	-	-	5433	35,0
BAwPC	3645	0,0	1069	81,4	1069	81,4
BAwCC	7186	0,0	1718	80,6	1718	80,6
BAwCC_enh	7671	0,0	59	100,0	59	100,0
BAwPC_enh	2834	0,4	44	100,0	44	100,0
BRP	172	27,3	2692	3,8	2650	0,0
server	2037	0,0	1910	5,3	2037	0,0
server_err	1	100,0	1	100,0	1	100,0
ring2	490	71,2	2393	0,1	2393	0,0
tcp	37	81,1	751	3,1	768	0,0
tcp_err	173	47,4	671	8,0	686	3,8

On remarque que pour tous les LCS, au moins un invariant a été capable d'éliminer des configurations. Dans certains cas, *Peterson3*, *Peterson4*, *BAwCC_enh*, *BAwPC_enh* et *server_err*, les invariants *CSRE* et *MOF* ont été capables de résoudre le problème de couverture sans même avoir besoin de l'analyse en arrière.

On sait que *MOF* est strictement moins précis que *CSRE*, toutefois en pratique on remarque que, sur sept LCS, les deux invariants ont fait jeu égal et sur les quatre autres où *CSRE* a fait mieux que *MOF* l'écart est faible, le plus grand écart était $5,3\%$ pour *CSRE* et 0% pour *MOF*.

8.4 Coûts des invariants

L'utilité des invariants est de diminuer le nombre de configurations manipulées par l'analyse en arrière. Toutefois les invariants prennent du temps de calcul supplémentaire. Ils ont deux types de surcoût : le premier est celui du précalcul, par exemple le calcul de point fixe pour les invariants *CSRE* et *MOF*, le second est celui du test de l'appartenance ou non dans l'invariant. Ce sont ces deux coûts qui sont représentés dans la Table 8.4, pour chaque invariant.

On remarque que le type de coût est différent entre d'un côté SI et de l'autre *CSRE* et *MOF*. L'inéquation d'état a besoin de peu de temps pour le

TABLE 8.4 – Coûts en secondes de l'utilisation des invariants.

	SI		*CSRE*		*MOF*	
	pre	∈	pre	∈	pre	∈
Peterson3	0.0	1.9	21.5	0.0	22.1	0.0
Peterson4	0.0	1895.2	>96h	-	>96h	-
pop3	0.0	0.3	>96h	-	0.1	0.0
BAwPC	0.0	3.8	0.0	0.0	0.0	0.0
BAwCC	0.0	9.3	0.1	0.0	0.1	0.0
BAwCC_enh	0.0	11.4	0.1	0.0	0.1	0.0
BAwPC_enh	0.0	3.1	0.0	0.0	0.0	0.0
BRP	0.0	0.1	0.1	0.0	0.1	0.0
server	0.0	1.2	0.6	0.0	0.3	0.0
server_err	0.0	0.0	0.0	0.0	0.0	0.0
ring2	0.0	0.2	0.2	0.0	0.2	0.0
tcp	0.0	0.0	0.0	0.0	0.0	0.0
tcp_err	0.0	0.1	0.0	0.0	0.0	0.0

précalcul, il faut juste écrire les inéquations et les donner au solveur SMT. Par contre le test de l'appartenance est plus long. Chaque test peut prendre environ une milliseconde, ce qui lorsque le nombre de tests est grand devient vite un surcoût important. Les invariants *CSRE* et *MOF* qui utilisent un point fixe ont au contraire un temps de précalcul qui peut être très long, notamment *CSRE*, mais une fois l'analyse commencée, il est facile de tester si une configuration est dans l'invariant ou non. En effet, les configurations manipulées ont souvent des mots d'au plus deux ou trois messages dans leurs canaux. Le surcoût de ces deux derniers invariants est donc concentré dans le précalcul.

On remarque que les mauvais résultats de `SI` avec les protocoles business s'expliquent par le surcoût des tests de l'appartenance. Ce surcoût n'est pas compensé, comme par exemple pour *Peterson4*, par une réduction des bases.

Remarque. *Nous avons testé l'heuristique d'exploration avec les LCS. Nous avons utilisé comme ordre $\leq_h$, le nombre de messages au total dans une configuration. Pour $c_1 = (q, w_1, w_2, \ldots, w_d)$ et $c_2 = (q', w'_1, w'_2, \ldots, w'_d)$ on a $c_1 \leq_h c_2$ lorsque $\sum_{i\in\{1,\ldots,d\}} |w'_i| \leq \sum_{i\in\{1,\ldots,d\}} |w_i|$. Toutefois en pratique cette heuristique n'a pas été efficace. Elle ralentit même le calcul dans plusieurs cas. Un autre ordre à tester pourrait être la distance à la localité initiale : plus la localité d'une configuration est proche, en nombre de transition, de la localité initiale, plus cette configuration est prioritaire. On remarque toutefois que pour les réseaux de Petri, l'heuristique a été principalement utile pour les instances couvrables, or les plus gros exemples que nous avons pour les LCS, les deux protocoles de Peterson et* `pop3`*, sont des instances non couvrables.* □

8.5 Utilisation heuristique parallélisation

On a vu que le calcul de point fixe comme celui de *CSRE* et même celui de *MOF* pouvait prendre trop de temps. Dans la version classique le calcul en arrière ne commence qu'après la fin de tous les précalculs pour les invariants. C'est pour cela que nous avons proposé dans le Chapitre 2, à la Section 2.2.2, l'heuristique dite de parallélisation qui lance le calcul en arrière sans invariant et éventuellement utilise des invariants si leurs précalculs sont terminés. L'idée est de disposer de l'invariant `SI` dès le début car son précalcul est trivial mais de commencer le calcul en arrière avant d'avoir fini le précalcul de *MOF*.

En regardant la Table 8.1 on voit qu'une telle technique permettrait de faire mieux que `SI` ou *MOF* : *Peterson3* serait résolu en 1.9 secondes grâce à `SI` et la résolution des exemples *BAwCC*, *BAwPC*, *BAwCC_enh* and *BAwPC_enh* ne serait plus autant ralentie grâce à *MOF*. Nous n'avons pas directement implémenté cette méthode car *OCaml* ne propose pas de parallélisation avec des threads. L'utilisation de processus nous est apparu trop lourde pour un gain marginal sur nos exemples. Nous avons par contre implémenté une preuve de concept avec une version légèrement modifiée de l'implémentation de classique de l'Algorithme 3 que nous avons utilisé jusque là. Cette implémentation a pour but d'émuler l'Algorithme 5, présenté à la Section 2.2.2. Dans `BML`, l'algorithme de recherche en arrière utilise une liste d'invariants dont il utilise la conjonction pour effectuer la ligne Ligne 7, $P \leftarrow N \cap I$. Dans notre preuve de concept cette liste est remplacée par une paire d'éléments, le premier étant un invariant, comme précédemment, et le second est un entier positif. Pour indiquer qu'on veut utiliser un invariant I à partir de l'itération 5 de la boucle *while* on met donc une paire composé de I et de 5 dans la liste. L'algorithme va se comporter comme s'il n'avait pas d'invariant pour les quatre premières étapes et ensuite va intersecter sa base B avec l'invariant I au début de la cinquième itération puis va continuer son calcul avec l'invariant I. On peut vérifier qu'on obtient bien exactement la même base à partir de la cinquième itération. On ne peut utiliser cette heuristique et garder la même précision seulement avec des invariants inductifs clos par le bas, ce qui est le cas de nos trois invariants. L'Exemple 8.5.1 illustre l'utilisation de cette preuve de concept.

Exemple 8.5.1. *Pour tester notre preuve de concept nous avons choisis l'instance pop3 avec l'invariant MOF. Il n'y pas d'amélioration en terme de temps, nous voulons juste illustrer l'utilisation de l'heuristique parallèle et le Lemme 2.2.5. En utilisant MOF à partir de la cinquième étape, l'analyse en arrière est donc imprécise et la base B contient 229 configurations au début la cinquième contre 147 configurations pour la version* `BML` *classique avec l'invariant MOF. Puis notre version d'émulation de l'heuristique parallèle utilise enfin cet invariant. La taille de la base passe d'un coup de 229 à 147. Les bases sont ensuite identiques de l'itération* 6 *à la dernière itération. Les deux versions finissent avec 2277 configurations dans leur base contrairement à l'analyse sans invariant qui*

en a 40412. □

Nous espérons avoir convaincu le lecteur que l'heuristique parallèle est une technique qui peut se révéler utile pour les invariants dont le temps de calcul peut être très long. De plus l'implémentation de cette heuristique ne change pas le code principal ni du calcul en arrière, ni du précalcul des invariants.

Conclusion

La vérification des systèmes concurrents est intéressante. Cette vérification peut être effectuée à l'aide du problème de couverture dans les réseaux de Petri et les LCS. L'analyse en arrière classique Finkel [1987]; Abdulla *et al.* [2000]; Finkel et Schnoebelen [2001] permet de résoudre ce problème dans les WSTS effectifs. Les réseaux de Petri et les LCS sont tous les deux des WSTS effectifs. L'objectif de la thèse était de fournir des outils pour vérifier efficacement en pratique le problème de couverture dans les réseaux de Petri et les LCS.

1. Bilan

Nous avons présenté une méthode générale pour les WSTS effectifs. Cette méthode accélère l'analyse en arrière classique Finkel [1987]; Abdulla *et al.* [2000]; Finkel et Schnoebelen [2001] en utilisant des invariants de couverture, c'est-à-dire des sur-approximations de l'ensemble de couverture. Ensuite nous avons présenté une heuristique d'exploration qui utilise un ordre de priorité entre les états dans le but d'accélérer l'analyse en arrière. Enfin, une autre technique, l'heuristique parallèle, permet de ne pas attendre le calcul d'un invariant pour commencer l'analyse en arrière.

Pour les réseaux de Petri nous avons introduit un nouvel invariant, l'analyse des signes qui calcule des places qui restent vides. Avec les trois invariants déjà connus, l'inéquation d'état, les trappes combinées à l'inéquation d'état, et la sémantique continue, cela fait quatre invariants que nous avons comparés. On a vu que la combinaison de l'analyse des signes et de l'inéquation d'état était proche de la sémantique continue. En pratique l'analyse des signes est utilisée comme un précalcul. Ce précalcul accélère fortement la décision du problème de couverture. Nous avons cherché d'autres précalculs. C'est pourquoi nous avons introduit les places à oméga. Une place à oméga est une place p telle que le problème de couverture pour un marquage m_{final} ne dépend pas du nombre de jetons $m_{final}(p)$. Ces places peuvent être éliminées. Le calcul des places à oméga est aussi implémenté sous forme de précalcul.

Nous avons implémenté un outil, `ICover`, basé sur `QCover` Blondin *et al.* [2016] qui nous a permis de réaliser des expérimentations. Nous avons pu constater que l'analyse des signes utilisée comme un précalcul et combinée avec l'inéquation d'état est efficace. Cette combinaison de deux méthodes simples est en pratique plus efficace que l'utilisation de la sémantique continue Blondin

et al. [2016], bien que cette dernière soit plus précise. Le calcul des places à oméga, qui est aussi utilisé comme un précalcul, s'est révélé être lui aussi très efficace sur les instances testées et ce même s'il y a peu de places à oméga calculées sur ces instances. L'heuristique d'exploration permet de résoudre quelques instances supplémentaires dans la collection de réseaux de Petri utilisée pour les expérimentations.

Pour les systèmes de canaux à pertes (LCS), nous avons présenté trois invariants : une adaptation de l'inéquation d'états, et les invariants *CSRE* et *MOF* qui capturent l'ordre d'émissions des messages.

On a développé un outil, `BML`, pour tester ces trois invariants. Oublier l'ordre d'exécution et oublier l'ordre des messages, ce que fait l'inéquation d'état, est compétitif. Les invariants *CSRE* et *MOF* permettent d'élaguer suffisamment de marquages non couvrables lors de l'analyse en arrière, ce qui a pour effet de réduire la taille des bases manipulées. Le premier a toutefois un problème d'explosion combinatoire qui l'empêche d'être performant en temps d'exécution. Le second, *MOF*, bien que moins précis, est plus performant et cet invariant est en pratique presque aussi précis que le premier, *CSRE*. En effet, dans nos expérimentations, *MOF* élague presque autant de configurations que *CSRE*. Une preuve de concept pour l'heuristique parallèle permet de démontrer l'utilité de cette méthode. En effet lors du choix d'un invariant, qui est un compromis entre rapidité et précision, cette méthode permet d'enlever en partie la contrainte du temps de précalcul.

2. Perspectives

Pour vérifier efficacement le problème de couverture, un moyen est de s'intéresser aux structures de données pour représenter des clos par le bas et des clos par le haut. On peut adapter les structures de Piipponen et Valmari [2016].

Nos expérimentations pour les réseaux de Petri utilisent les mêmes instances que les outils `QCover` Blondin *et al.* [2016] et `Petrinizer` Esparza *et al.* [2014]. Sur les 115 instances non couvrables, 105 sont résolues par `ICover` et `QCover`. De plus, sur les 61 instances couvrables, 44 instances ont été générées par `SATABS` Donaldson *et al.* [2011]. Un effort pour trouver de nouvelles instances intéressantes aussi bien couvrables que non couvrables permettrait de mieux comparer les différents outils.

Pour capturer des propriétés sur l'ordre des messages dans les LCS, on peut étudier les *view abstractions* Abdulla *et al.* [2016b]. Cette technique utilise des abstractions de plus en plus précise, paramétré par $k \geq 1$. Les *MOF* sont proches des *view abstractions* avec $k = 2$. Il serait intéressant comparer ces deux techniques. De plus on peut étudier l'impact qu'aurait un $k > 2$ sur la précision de l'invariant. Est ce qu'avec un k grand, l'invariant serait suffisamment précis pour résoudre le problème de couverture sans commencer l'analyse en arrière ?

Les systèmes à mémoire faible sont équivalents aux LCS Atig *et al.* [2010].

Il serait intéressant d'adapter l'analyse en arrière et les trois invariants pour les LCS, dans le but de vérifier le bon fonctionnement de ces systèmes.

Nos expérimentations pour les LCS sont basées sur seulement 13 instances. Il faudrait augmenter le nombre d'instances et notamment le nombre d'instances qui ne sont pas résolues par l'analyse en arrière classique.

Bibliographie

1993. *Proceedings of the Eighth Annual Symposium on Logic in Computer Science (LICS '93), Montreal, Canada, June 19-23, 1993*. IEEE Computer Society. ISBN 0-8186-3140-6.
URL http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=899

2003. *10th International Symposium on Temporal Representation and Reasoning / 4th International Conference on Temporal Logic (TIME-ICTL 2003), 8-10 July 2003, Cairns, Queensland, Australia*. IEEE Computer Society. ISBN 0-7695-1912-1.
URL http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=8624

2008. *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*. IEEE Computer Society. ISBN 978-0-7695-3183-0.
URL http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4557886

Abdulla, P-A. et Jonsson, B., 1996. Verifying programs with unreliable channels. *Inf. Comput.*, 127(2) :91–101.

Abdulla, Parosh Aziz, Annichini, Aurore et Bouajjani, Ahmed, 1999. Symbolic verification of lossy channel systems : Application to the bounded retransmission protocol. Dans Cleaveland [1999], pages 208–222. doi : 10.1007/3-540-49059-0_15.
URL https://doi.org/10.1007/3-540-49059-0_15

Abdulla, Parosh Aziz, Atig, Mohamed Faouzi, Bouajjani, Ahmed et Ngo, Tuan Phong, 2016a. The benefits of duality in verifying concurrent programs under TSO. Dans Desharnais et Jagadeesan [2016], pages 5 :1–5 :15. doi : 10.4230/LIPIcs.CONCUR.2016.5.
URL https://doi.org/10.4230/LIPIcs.CONCUR.2016.5

Abdulla, Parosh Aziz, Bouajjani, Ahmed et Jonsson, Bengt, 1998. On-the-fly analysis of systems with unbounded, lossy FIFO channels. Dans Hu

et Vardi [1998], pages 305–318. doi :10.1007/BFb0028754.
URL https://doi.org/10.1007/BFb0028754

Abdulla, Parosh Aziz, Cerans, Karlis, Jonsson, Bengt et Tsay, Yih-Kuen, 2000. Algorithmic analysis of programs with well quasi-ordered domains. *Inf. Comput.*, 160(1-2) :109–127. doi :10.1006/inco.1999.2843.
URL https://doi.org/10.1006/inco.1999.2843

Abdulla, Parosh Aziz, Haziza, Frédéric et Holík, Lukás, 2016b. Parameterized verification through view abstraction. *STTT*, 18(5) :495–516. doi : 10.1007/s10009-015-0406-x.
URL https://doi.org/10.1007/s10009-015-0406-x

Abdulla, Parosh Aziz et Jonsson, Bengt, 1993. Verifying programs with unreliable channels. Dans DBL [1993], pages 160–170. doi :10.1109/LICS.1993.287591.
URL https://doi.org/10.1109/LICS.1993.287591

Abdulla, Parosh Aziz et Leino, K. Rustan M., rédacteurs, 2011. *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, tome 6605 de *Lecture Notes in Computer Science*. Springer. ISBN 978-3-642-19834-2. doi : 10.1007/978-3-642-19835-9.
URL https://doi.org/10.1007/978-3-642-19835-9

Alur, Rajeev et Peled, Doron A., rédacteurs, 2004. *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings*, tome 3114 de *Lecture Notes in Computer Science*. Springer. ISBN 3-540-22342-8. doi :10.1007/b98490.
URL https://doi.org/10.1007/b98490

Annichini, Aurore, Bouajjani, Ahmed et Sighireanu, Mihaela, 2001. Trex : A tool for reachability analysis of complex systems. Dans Berry *et al.* [2001], pages 368–372. doi :10.1007/3-540-44585-4_34.
URL https://doi.org/10.1007/3-540-44585-4_34

Atig, Mohamed Faouzi, Bouajjani, Ahmed, Burckhardt, Sebastian et Musuvathi, Madanlal, 2010. On the verification problem for weak memory models. Dans Hermenegildo et Palsberg [2010], pages 7–18. doi :10.1145/1706299.1706303.
URL http://doi.acm.org/10.1145/1706299.1706303

Ball, Thomas, Cook, Byron, Levin, Vladimir et Rajamani, Sriram K., 2004. SLAM and static driver verifier : Technology transfer of formal

methods inside microsoft. Dans Boiten *et al.* [2004], pages 1–20. doi : 10.1007/978-3-540-24756-2_1.
URL https://doi.org/10.1007/978-3-540-24756-2_1

Ball, Thomas, Levin, Vladimir et Rajamani, Sriram K., 2011. A decade of software model checking with SLAM. *Commun. ACM*, 54(7) :68–76. doi : 10.1145/1965724.1965743.
URL http://doi.acm.org/10.1145/1965724.1965743

Bartlett, Keith A., Scantlebury, Roger A. et Wilkinson, Peter T., 1969. A note on reliable full-duplex transmission over half-duplex links. *Commun. ACM*, 12(5) :260–261. doi :10.1145/362946.362970.
URL http://doi.acm.org/10.1145/362946.362970

Begin, Laurent Van, 2004. Efficient verification of counting abstractions for parametric systems.

Bergerand, Jean-Louis, 1986. *LUSTRE : un langage déclaratif pour le temps réel. (LUSTRE : a real time declarative language)*. Thèse de doctorat, Grenoble Institute of Technology, France.
URL https://tel.archives-ouvertes.fr/tel-00320006

Berry, Gérard, Comon, Hubert et Finkel, Alain, rédacteurs, 2001. *Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings*, tome 2102 de *Lecture Notes in Computer Science*. Springer. ISBN 3-540-42345-1. doi :10.1007/3-540-44585-4.
URL https://doi.org/10.1007/3-540-44585-4

Biere, Armin et Bloem, Roderick, rédacteurs, 2014. *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, tome 8559 de *Lecture Notes in Computer Science*. Springer. ISBN 978-3-319-08866-2. doi :10.1007/978-3-319-08867-9.
URL https://doi.org/10.1007/978-3-319-08867-9

Blondin, Michael, 2016. *Algorithmique et complexité des systèmes à compteurs. (Algorithmics and complexity of counter machines)*. Thèse de doctorat, University of Paris-Saclay, France.
URL https://tel.archives-ouvertes.fr/tel-01359000

Blondin, Michael, Finkel, Alain, Haase, Christoph et Haddad, Serge, 2016. Approaching the coverability problem continuously. Dans Chechik et Raskin [2016], pages 480–496. doi :10.1007/978-3-662-49674-9_28.
URL https://doi.org/10.1007/978-3-662-49674-9_28

Boigelot, Bernard et Godefroid, Patrice, 1999. Symbolic verification of communication protocols with infinite state spaces using qdds. *Formal Methods in System Design*, 14(3) :237–255. doi :10.1023/A:1008719024240.
URL https://doi.org/10.1023/A:1008719024240

Boiten, Eerke A., Derrick, John et Smith, Graeme, rédacteurs, 2004. *Integrated Formal Methods, 4th International Conference, IFM 2004, Canterbury, UK, April 4-7, 2004, Proceedings*, tome 2999 de *Lecture Notes in Computer Science*. Springer. ISBN 3-540-21377-5. doi :10.1007/b96106.
URL https://doi.org/10.1007/b96106

Bolognesi, Tommaso et Brinksma, Ed, 1987. Introduction to the ISO specification language LOTOS. *Computer Networks*, 14 :25–59. doi : 10.1016/0169-7552(87)90085-7.
URL https://doi.org/10.1016/0169-7552(87)90085-7

Bouajjani, Ahmed, Habermehl, Peter et Vojnar, Tomás, 2004. Abstract regular model checking. Dans Alur et Peled [2004], pages 372–386. doi : 10.1007/978-3-540-27813-9_29.
URL https://doi.org/10.1007/978-3-540-27813-9_29

Bradley, Aaron R., 2011. Sat-based model checking without unrolling. Dans Jhala et Schmidt [2011], pages 70–87. doi :10.1007/978-3-642-18275-4_7.
URL https://doi.org/10.1007/978-3-642-18275-4_7

Brand, Daniel et Zafiropulo, Pitro, 1983. On communicating finite-state machines. *J. ACM*, 30(2) :323–342. doi :10.1145/322374.322380.
URL http://doi.acm.org/10.1145/322374.322380

Cécé, G., Finkel, A. et Iyer, S.P., 1996. Unreliable channels are easier to verify than perfect channels. *Inf. Comput.*, 124(1) :20–31.

Chambart, Pierre et Schnoebelen, Philippe, 2008. The ordinal recursive complexity of lossy channel systems. Dans DBL [2008], pages 205–216. doi : 10.1109/LICS.2008.47.
URL https://doi.org/10.1109/LICS.2008.47

Chechik, Marsha et Raskin, Jean-François, rédacteurs, 2016. *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, tome 9636 de *Lecture Notes in Computer Science*. Springer. ISBN 978-3-662-49673-2. doi :10.1007/978-3-662-49674-9.
URL https://doi.org/10.1007/978-3-662-49674-9

Chehaibar, Ghassan, Garavel, Hubert, Mounier, Laurent, Tawbi, Nadia et Zulian, Ferruccio, 1996. Specification and verification of the powerscale^{tm} bus arbitration protocol : An industrial experiment with LOTOS. Dans Gotzhein et Bredereke [1996], pages 435–450.

Clarke, Edmund M., 2003. Counterexample-guided abstraction refinement. Dans DBL [2003], page 7. doi :10.1109/TIME.2003.1214874.
URL https://doi.org/10.1109/TIME.2003.1214874

Cleaveland, Rance, rédacteur, 1999. *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS '99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22-28, 1999, Proceedings*, tome 1579 de *Lecture Notes in Computer Science*. Springer. ISBN 3-540-65703-7. doi :10.1007/3-540-49059-0.
URL https://doi.org/10.1007/3-540-49059-0

Cousot, P. et Cousot, R., 1977. Abstract interpretation : A unified lattice model for static analysis of programs by construction or approximation of fixpoints. Dans *POPL*, pages 238–252. ACM.

Cousot, Patrick, Cousot, Radhia, Feret, Jérôme, Miné, Antoine, Rival, Xavier, Blanchet, Bruno, Monniaux, David et Mauborgne, Laurent, 2001. Astrée. http://www.astree.ens.fr.

D'Argenio, Pedro R. et Melgratti, Hernán C., rédacteurs, 2013. *CONCUR 2013 - Concurrency Theory - 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings*, tome 8052 de *Lecture Notes in Computer Science*. Springer. ISBN 978-3-642-40183-1. doi :10.1007/978-3-642-40184-8.
URL https://doi.org/10.1007/978-3-642-40184-8

David, René, 1987. Continuous petri nets. Dans *8th European Workshop on Application and Theory of Petri nets*, tome 275.

de Moura, Leonardo Mendonça et Bjørner, Nikolaj, 2008. Z3 : an efficient SMT solver. Dans *TACAS*, pages 337–340. Springer.

Delzanno, Giorgio, Raskin, Jean-François et Begin, Laurent Van, 2001. Attacking symbolic state explosion. Dans Berry *et al.* [2001], pages 298–310. doi :10.1007/3-540-44585-4_28.
URL https://doi.org/10.1007/3-540-44585-4_28

Delzanno, Giorgio, Raskin, Jean-François et Begin, Laurent Van, 2004. Covering sharing trees : a compact data structure for parameterized verification. *STTT*, 5(2-3) :268–297. doi :10.1007/s10009-003-0110-0.
URL https://doi.org/10.1007/s10009-003-0110-0

Desel, Jorg et Esparza, Javier, 2005. *Free choice Petri nets*, tome 40. Cambridge university press.

Desharnais, Josée et Jagadeesan, Radha, rédacteurs, 2016. *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, tome 59 de *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-017-0.
URL http://www.dagstuhl.de/dagpub/978-3-95977-017-0

Donaldson, Alastair F., Kaiser, Alexander, Kroening, Daniel et Wahl, Thomas, 2011. Symmetry-aware predicate abstraction for shared-variable concurrent programs. Dans Gopalakrishnan et Qadeer [2011], pages 356–371. doi :10.1007/978-3-642-22110-1_28.
URL https://doi.org/10.1007/978-3-642-22110-1_28

Donatelli, Susanna et Kleijn, H. C. M., rédacteurs, 1999. *Application and Theory of Petri Nets 1999, 20th International Conference, ICATPN '99, Williamsburg, Virginia, USA, June 21-25, 1999, Proceedings*, tome 1639 de *Lecture Notes in Computer Science*. Springer. ISBN 3-540-66132-8. doi : 10.1007/3-540-48745-X.
URL https://doi.org/10.1007/3-540-48745-X

D'Osualdo, Emanuele, Kochems, Jonathan et Ong, C.-H. Luke, 2013. Automatic verification of erlang-style concurrency. Dans Logozzo et Fähndrich [2013], pages 454–476. doi :10.1007/978-3-642-38856-9_24.
URL https://doi.org/10.1007/978-3-642-38856-9_24

Dufourd, Catherine, Finkel, Alain et Schnoebelen, Philippe, 1998. Reset nets between decidability and undecidability. Dans Larsen *et al.* [1998], pages 103–115. doi :10.1007/BFb0055044.
URL https://doi.org/10.1007/BFb0055044

Durand-Gasselin, Antoine et Habermehl, Peter, 2010. On the use of non-deterministic automata for presburger arithmetic. Dans Gastin et Laroussinie [2010], pages 373–387. doi :10.1007/978-3-642-15375-4_26.
URL https://doi.org/10.1007/978-3-642-15375-4_26

Erdogmus, Hakan et Havelund, Klaus, rédacteurs, 2017. *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, Santa Barbara, CA, USA, July 10-14, 2017*. ACM. ISBN 978-1-4503-5077-8.
URL http://dl.acm.org/citation.cfm?id=3092282

Esparza, Javier, Ledesma-Garza, Ruslán, Majumdar, Rupak, Meyer, Philipp J. et Niksic, Filip, 2014. An smt-based approach to coverability analysis. Dans Biere et Bloem [2014], pages 603–619. doi :10.1007/

978-3-319-08867-9_40.
URL https://doi.org/10.1007/978-3-319-08867-9_40

Esparza, Javier et Melzer, Stephan, 2000. Verification of safety properties using integer programming : Beyond the state equation. *Formal Methods in System Design*, 16(2) :159–189. doi :10.1023/A:1008743212620.
URL https://doi.org/10.1023/A:1008743212620

Estelle, ISO, 1988. A formal description technique based on an extended state transition model. international standard 9074. *International Organization for Standardization/ Information Processing Systems/ Open Systems Interconnection, Gen eve.*

Finkel, A. et Schnoebelen, P., 2001. Well-structured transition systems everywhere ! *Theor. Comput. Sci.*, 256(1-2) :63–92.

Finkel, Alain, 1987. A generalization of the procedure of karp and miller to well structured transition systems. Dans Ottmann [1987], pages 499–508. doi :10.1007/3-540-18088-5_43.
URL https://doi.org/10.1007/3-540-18088-5_43

Finkel, Alain, 1990. Reduction and covering of infinite reachability trees. *Inf. Comput.*, 89(2) :144–179. doi :10.1016/0890-5401(90)90009-7.
URL https://doi.org/10.1016/0890-5401(90)90009-7

Finkel, Alain, 1991. The minimal coverability graph for petri nets. Dans Rozenberg [1993], pages 210–243. doi :10.1007/3-540-56689-9_45.
URL https://doi.org/10.1007/3-540-56689-9_45

Flanagan, Cormac et König, Barbara, rédacteurs, 2012. *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, tome 7214 de *Lecture Notes in Computer Science*. Springer. ISBN 978-3-642-28755-8. doi :10.1007/978-3-642-28756-5.
URL https://doi.org/10.1007/978-3-642-28756-5

Fraca, Estíbaliz et Haddad, Serge, 2015a. Complexity analysis of continuous Petri nets. 137(1) :1–28.

Fraca, Estíbaliz et Haddad, Serge, 2015b. Complexity analysis of continuous petri nets. *Fundam. Inform.*, 137(1) :1–28. doi :10.3233/FI-2015-1168.
URL https://doi.org/10.3233/FI-2015-1168

Gall, Tristan Le, Jeannet, Bertrand et Jéron, Thierry, 2006. Verification of communication protocols using abstract interpretation of FIFO queues.

Dans Johnson et Vene [2006], pages 204–219. doi :10.1007/11784180_17.
URL https://doi.org/10.1007/11784180_17

Ganty, P., 2004. Mist – A safety checker for petri nets and extensions. http://github.com/pierreganty/mist.

Gastin, Paul et Laroussinie, François, rédacteurs, 2010. *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, tome 6269 de *Lecture Notes in Computer Science*. Springer. ISBN 978-3-642-15374-7. doi :10.1007/978-3-642-15375-4.
URL https://doi.org/10.1007/978-3-642-15375-4

Geeraerts, Gilles, Raskin, Jean-François et Begin, Laurent Van, 2006. Expand, enlarge and check : New algorithms for the coverability problem of WSTS. *J. Comput. Syst. Sci.*, 72(1) :180–203. doi :10.1016/j.jcss.2005.09.001.
URL https://doi.org/10.1016/j.jcss.2005.09.001

Geeraerts, Gilles, Raskin, Jean-François et Begin, Laurent Van, 2010. On the efficient computation of the minimal coverability set of petri nets. *Int. J. Found. Comput. Sci.*, 21(2) :135–165. doi :10.1142/S0129054110007180.
URL https://doi.org/10.1142/S0129054110007180

Geffroy, T., Leroux, J. et Sutre, G., 2017a. coverability checker for lcs. http://dept-info.labri.u-bordeaux.fr/~tgeffroy/lcs/.

Geffroy, Thomas, Leroux, Jérôme et Sutre, Grégoire, 2016a. Occam's razor applied to the petri net coverability problem. Dans Larsen *et al.* [2016], pages 77–89. doi :10.1007/978-3-319-45994-3_6.
URL https://doi.org/10.1007/978-3-319-45994-3_6

Geffroy, Thomas, Leroux, Jérôme et Sutre, Grégoire, 2017b. Backward coverability with pruning for lossy channel systems. Dans Erdogmus et Havelund [2017], pages 132–141. doi :10.1145/3092282.3092292.
URL http://doi.acm.org/10.1145/3092282.3092292

Geffroy, Thomas, Leroux, Jérôme et Sutre, Grégoire, 2016b. ICover. https://github.com/gsutre/icover/.

German, Steven M. et Sistla, A. Prasad, 1992. Reasoning about systems with many processes. *J. ACM*, 39(3) :675–735. doi :10.1145/146637.146681.
URL http://doi.acm.org/10.1145/146637.146681

Ginsburg, Seymour et Spanier, Edwin, 1966. Semigroups, presburger formulas, and languages. *Pacific journal of Mathematics*, 16(2) :285–296.

GOPALAKRISHNAN, Ganesh et QADEER, Shaz, rédacteurs, 2011. *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, tome 6806 de *Lecture Notes in Computer Science*. Springer. ISBN 978-3-642-22109-5. doi : 10.1007/978-3-642-22110-1.
URL https://doi.org/10.1007/978-3-642-22110-1

GOTZHEIN, Reinhard et BREDEREKE, Jan, rédacteurs, 1996. *Formal Description Techniques IX : Theory, application and tools, IFIP TC6 WG6.1 International Conference on Formal Description Techniques IX / Protocol Specification, Testing and Verification XVI, Kaiserslautern, Germany, 8-11 October 1996*, tome 69 de *IFIP Conference Proceedings*. Chapman & Hall. ISBN 0-412-79490-X.

HERMENEGILDO, Manuel V. et PALSBERG, Jens, rédacteurs, 2010. *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010*. ACM. ISBN 978-1-60558-479-9.
URL http://dl.acm.org/citation.cfm?id=1706299

HEUSSNER, Alexander, GALL, Tristan Le et SUTRE, Grégoire, 2012. Mcscm : A general framework for the verification of communicating machines. Dans Flanagan et König [2012], pages 478–484. doi :10.1007/978-3-642-28756-5_34.
URL https://doi.org/10.1007/978-3-642-28756-5_34

HIGMAN, G., 1952. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc.*, s3-2(1) :326–336.

HU, Alan J. et VARDI, Moshe Y., rédacteurs, 1998. *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, tome 1427 de *Lecture Notes in Computer Science*. Springer. ISBN 3-540-64608-6. doi :10.1007/BFb0028725.
URL https://doi.org/10.1007/BFb0028725

JHALA, Ranjit et SCHMIDT, David A., rédacteurs, 2011. *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings*, tome 6538 de *Lecture Notes in Computer Science*. Springer. ISBN 978-3-642-18274-7. doi :10.1007/978-3-642-18275-4.
URL https://doi.org/10.1007/978-3-642-18275-4

JOHNSON, Michael et VENE, Varmo, rédacteurs, 2006. *Algebraic Methodology and Software Technology, 11th International Conference, AMAST 2006, Kuressaare, Estonia, July 5-8, 2006, Proceedings*, tome 4019 de *Lecture Notes*

in Computer Science. Springer. ISBN 3-540-35633-9. doi :10.1007/11784180. URL https://doi.org/10.1007/11784180

Kaiser, Alexander, Kroening, Daniel et Wahl, Thomas, 2014. A widening approach to multithreaded program verification. *ACM Trans. Program. Lang. Syst.*, 36(4) :14 :1–14 :29. doi :10.1145/2629608. URL http://doi.acm.org/10.1145/2629608

Karp, Richard M. et Miller, Raymond E., 1969a. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2) :147–195. doi :10.1016/S0022-0000(69)80011-5. URL https://doi.org/10.1016/S0022-0000(69)80011-5

Karp, Richard M. et Miller, Raymond E., 1969b. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2) :147–195.

Kloos, Johannes, Majumdar, Rupak, Niksic, Filip et Piskac, Ruzica, 2013. Incremental, inductive coverability. Dans Sharygina et Veith [2013], pages 158–173. doi :10.1007/978-3-642-39799-8_10. URL https://doi.org/10.1007/978-3-642-39799-8_10

Lamport, Leslie, 1986. The mutual exclusion problem : partii - statement and solutions. *J. ACM*, 33(2) :327–348. doi :10.1145/5383.5385. URL http://doi.acm.org/10.1145/5383.5385

Larsen, Kim Guldstrand, Potapov, Igor et Srba, Jirí, rédacteurs, 2016. *Reachability Problems - 10th International Workshop, RP 2016, Aalborg, Denmark, September 19-21, 2016, Proceedings*, tome 9899 de *Lecture Notes in Computer Science*. Springer. ISBN 978-3-319-45993-6. doi :10.1007/978-3-319-45994-3. URL https://doi.org/10.1007/978-3-319-45994-3

Larsen, Kim Guldstrand, Skyum, Sven et Winskel, Glynn, rédacteurs, 1998. *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings*, tome 1443 de *Lecture Notes in Computer Science*. Springer. ISBN 3-540-64781-3. doi :10.1007/BFb0055035. URL https://doi.org/10.1007/BFb0055035

Lipton, R. J., 1976. The reachability problem requires exponential space. Rapport technique 62, Yale University.

Logozzo, Francesco et Fähndrich, Manuel, rédacteurs, 2013. *Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings*, tome 7935 de *Lecture Notes in Computer Science*. Springer. ISBN 978-3-642-38855-2. doi :10.1007/978-3-642-38856-9. URL https://doi.org/10.1007/978-3-642-38856-9

MAJUMDAR, Rupak et WANG, Zilong, 2013. Expand, enlarge, and check for branching vector addition systems. Dans D'Argenio et Melgratti [2013], pages 152–166. doi :10.1007/978-3-642-40184-8_12.
URL https://doi.org/10.1007/978-3-642-40184-8_12

MAYR, Richard, 2003. Undecidable problems in unreliable computations. *Theor. Comput. Sci.*, 297(1-3) :337–354. doi :10.1016/S0304-3975(02)00646-1.
URL https://doi.org/10.1016/S0304-3975(02)00646-1

OTTMANN, Thomas, rédacteur, 1987. *Automata, Languages and Programming, 14th International Colloquium, ICALP87, Karlsruhe, Germany, July 13-17, 1987, Proceedings*, tome 267 de *Lecture Notes in Computer Science*. Springer. ISBN 3-540-18088-5. doi :10.1007/3-540-18088-5.
URL https://doi.org/10.1007/3-540-18088-5

PIIPPONEN, Artturi et VALMARI, Antti, 2016. Constructing minimal coverability sets. *Fundam. Inform.*, 143(3-4) :393–414. doi :10.3233/FI-2016-1319.
URL https://doi.org/10.3233/FI-2016-1319

PRESBURGER, Mojzesz, 1929. Uber die vollstandigkeit eines gewissen systems der arithmetik ganzer zahlen. *welchen die Addition als einzige Operation hervortritt*.

RACKOFF, C., 1978. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2) :223–231.

RAVN, Anders P., SRBA, Jirí et VIGHIO, Saleem, 2011. Modelling and verification of web services business activity protocol. Dans Abdulla et Leino [2011], pages 357–371. doi :10.1007/978-3-642-19835-9_32.
URL https://doi.org/10.1007/978-3-642-19835-9_32

RECALDE, Laura, TERUEL, Enrique et SUÁREZ, Manuel Silva, 1999. Autonomous continuous P/T systems. Dans Donatelli et Kleijn [1999], pages 107–126. doi :10.1007/3-540-48745-X_8.
URL https://doi.org/10.1007/3-540-48745-X_8

REYNIER, Pierre-Alain et SERVAIS, Frédéric, 2013. Minimal coverability set for petri nets : Karp and miller algorithm with pruning. *Fundam. Inform.*, 122(1-2) :1–30. doi :10.3233/FI-2013-781.
URL https://doi.org/10.3233/FI-2013-781

ROZENBERG, Grzegorz, rédacteur, 1993. *Advances in Petri Nets 1993, Papers from the 12th International Conference on Applications and Theory of Petri Nets, Gjern, Denmark, June 1991*, tome 674 de *Lecture Notes in Computer Science*. Springer. ISBN 3-540-56689-9. doi :10.1007/3-540-56689-9.
URL https://doi.org/10.1007/3-540-56689-9

SCADE, 1988. SCADE Suite. http://www.ansys.com/products/embedded-software/ansys-scade-suite.

Sharygina, Natasha et Veith, Helmut, rédacteurs, 2013. *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, tome 8044 de *Lecture Notes in Computer Science*. Springer. ISBN 978-3-642-39798-1. doi : 10.1007/978-3-642-39799-8.
URL https://doi.org/10.1007/978-3-642-39799-8

slam, 2001. SLAM. http://www.microsoft.com/en-us/research/project/slam.

Souyris, Jean et Delmas, David, 2007. Experimental assessment of astrée on safety-critical avionics software. *Computer Safety, Reliability, and Security*, pages 479–490.