THÈSE DE DOCTORAT DE
L'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité Informatique

(École Doctorale Informatique, Télécommunications et Électronique de Paris – ED130)

Présentée par

M. Mathieu SASSOLAS

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse

Méthodes qualitatives et quantitatives pour la détection d'information cachée

Soutenue le 28 novembre 2011

devant le jury composé de

Catuscia PALAMIDESSI	*Rapporteur*
Jean-François RASKIN	*Rapporteur*
Philippe DARONDEAU	*Examinateur*
Serge HADDAD	*Examinateur*
Fabrice KORDON	*Examinateur*
Béatrice BÉRARD	*Directrice de thèse*

REMERCIEMENTS

Je souhaite remercier Catuscia PALAMIDESSI et Jean-François RASKIN d'avoir été rapporteurs de cette thèse. Je souhaite également remercier Philippe DARONDEAU, Serge HADDAD et Fabrice KORDON d'avoir accepté de faire partie de mon jury. Collaborer avec Serge a toujours été fort enrichissant, même lorsqu'il s'agissait de sauter un repas afin de terminer une preuve. Les conseils de Fabrice ont toujours été très avisés – et bien moins coûteux en chocolat qu'on ne pouvait le penser. Je voudrais remercier tout particulièrement Béatrice BÉRARD, dont l'encadrement patient et minutieux est pour beaucoup dans cette thèse.

J'ai aussi eu beaucoup de plaisir à travailler, tout au long de cette thèse, avec Gilles BENATTAR, Didier LIME, Olivier H. ROUX, John MULLINS (avec qui la recherche est mêlée aux bagels, au sirop d'érable et à la viande fumée), Adrian EFTENIE, Marc ZEITOUN, Claudine PICARONNY et Nathalie SZNAJDER.

Travailler au sein de l'équipe MoVe est synonyme d'ambiance chaleureuse, de débats passionés dans la salle café, mais aussi de solidarité lorsqu'il s'agit de relire un papier (refusé pour la n-ième fois) ou une introduction de thèse (merci Denis POITRENAUD !). Je remercie en particulier Véronique VARENNE et Jean-Luc MOUNIER qui m'ont aidés dans deux combats quotidiens : à force de détermination, Véronique a pu démêler le sac de nœuds qu'était mon dossier administratif ; Jean-Luc a quant à lui toujours été un grand atout contre l'inconstance des imprimantes.

Au quotidien, j'ai grandement aprécié l'ambiance du BUREAU 818, faite à la fois de saucisson, de BZflag, d'alligators (jamais découpés), de débats stériles, de totems et de gongs, d'horoscopes de thèse, de batailles pour le contrôle de la musique, de dessins sur la porte... C'est pourquoi je tiens à remercier les membres (et membres associés) du Bureau 818, passés et présents : Dr Lom HILLAH (dont l'optimisme permanent est toujours une source d'encouragements), Dr Alban LINARD (avec qui je partageais ma passion pour l'Ardèche et de ses spécialités charcutières), Dr Alexandre HAMEZ (fan comme moi de l'*effet flamme*), Dr Jean-Baptiste VORON (toujours à jour pour les gadgets Apple), Dr Xavier RENAULT, Jun ZHU, Silien HONG (dont le sérieux a pu finalement être corrompu par BZflag), (ne veut pas être Dr) Clément DÉMOULINS (à qui je pardonne son client modifié), Nicolas GIBELIN (*admin* geek bien trop sportif), Yan ZHANG (qui supporte nos jeux puérils sans broncher), Thomas PREUD'HOMME (MoVe par alliance), Ala Eddine BEN SALEM (qui m'impressionne toujours avec ses journées de 14h), Yan BEN MAÏSSA (trafiquant en cornes de gazelles), Maximilien COLANGE (éternel stagiaire que je suis fier d'avoir « recruté » dans le bureau), Etienne RENAULT (qui apporte de la joie dans le bureau en mettant du JJG), ainsi que les nombreux stagiaires (Pierre PARUTTO, Élodie BANEL...) qui ont eu le (bon ?)heur de passer par le bureau.

Hors des murs de Jussieu – ou de Kennedy – j'ai eu la chance de compter sur le soutien moral de ma famille et de mes amis, que je remercie chaleureusement. Mes parents, Nathalie et Bruno, ont toujours été disponibles pour que je me repose

avec eux lors de sympathiques week-ends. Pouvoir retrouver Paul et Alban lors de ces virées au bord de la mer n'était évidemment pas étranger au bonheur qu'elles me procuraient.

Mes camarades de la Syndic@list ont su me faire oublier l'informatique à la fois par des interrogations fondamentales sur le végétalisme mais aussi en me permettant de détendre mes doigts engourdis par la rédaction en coulant du béton. J'espère pouvoir vous apporter le même soutien moral lors de vos rédactions respectives, à l'aide de force bœuf bourguignon et d'époisse à l'occasion de futurs séjours à Vermot.

J'ai une pensée particulière pour mes colocataires Tanguy et Malo, qui ont supporté mes horaires aléatoires pendant ces trois ans de thèse. Ils m'ont toujours accueillis avec le sourire et une bière bien fraîche.

Enfin, je souhaite remercier de tout mon cœur Juliette, qui a pu suivre de (trop) loin puis de près l'avancement de ma rédaction.

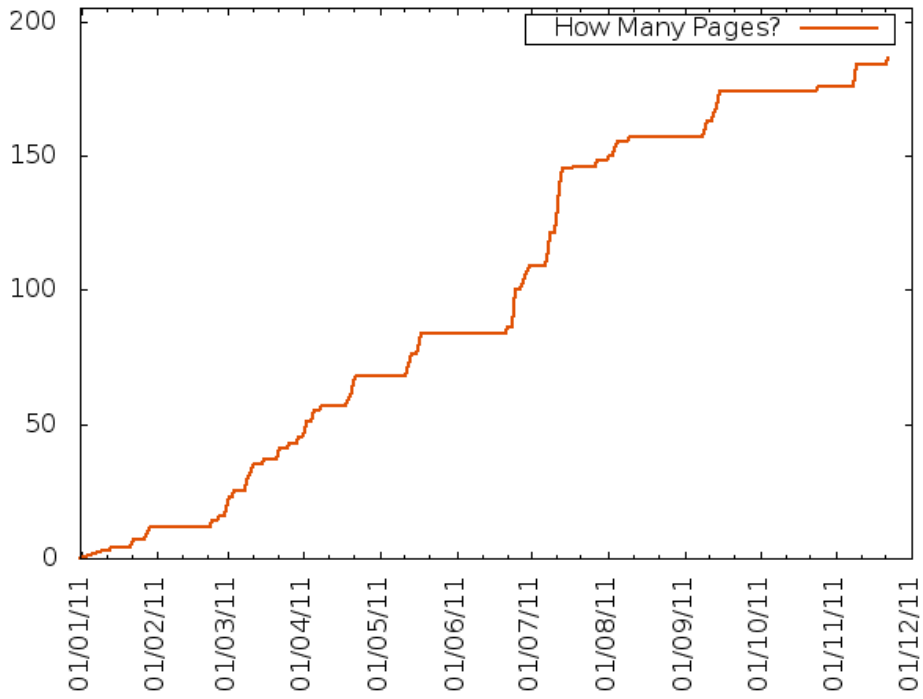


FIGURE 1: Évolution du nombre de pages de ce manuscrit en fonction du temps.

RÉSUMÉ

Les systèmes informatiques sont devenus omniprésents et sont utilisés au quotidien pour gérer toujours plus d'information. Ces informations sont de plus en plus souvent confidentielles : il peut s'agir d'informations stratégiques militaires ou financières, ou bien d'informations personnelles. La fuite de ces informations peut ainsi avoir des conséquences graves telles que des pertes humaines, financières, des violations de la vie privée ou de l'usurpation d'identité.

Les méthodes formelles de vérification sont nécessaires pour garantir la sûreté et la sécurité du système. Ces techniques s'appliquent sur des modèles du système où certains paramètres – en particulier des paramètres quantitatifs – peuvent avoir été abstraits. Ces paramètres ont pu être utilisés dans des attaques de systèmes réputés sûrs. Les modèles actuels intègrent donc ces informations, qui se rapportent le plus souvent à l'écoulement (continu) du temps ou aux lois de probabilités qui régissent le fonctionnement du système.

Les modèles intègrent d'autre part le découpage de l'application en divers processus, mûs par des buts différents, voire contraires : l'un peut chercher à dissimuler de l'information tandis que l'autre cherche à la découvrir. Lorsque le fonctionnement de tous les processus est partiellement ou totalement inconnu, les méthodes formelles doivent considérer tous les cas possibles, ce qui est en général impossible en théorie – s'il existe un nombre infini de possibilités – ou en pratique – s'il existe trop de possibilités pour que l'on puisse les envisager exhaustivement.

Les contributions de cette thèse se découpent en trois parties. Tout d'abord, nous étudions le problème de synthèse d'un canal de communication dans un système décrit par un transducteur. Malgré les limites imposées par ce modèle, nous montrons que le problème de synthèse est indécidable en général. Cependant, lorsque le système est fonctionnel, c'est-à-dire que son fonctionnement externe est toujours le même, le problème devient décidable.

Nous généralisons ensuite le concept d'opacité aux systèmes probabilistes, en donnant des mesures groupées en deux familles. Lorsque le système est opaque, nous évaluons la robustesse de cette opacité vis-à-vis des informations données par les lois de probabilités du système. Lorsque le système n'est pas opaque, nous évaluons la taille de la faille de sécurité induite par cette non opacité.

Enfin, nous étudions le modèle des automates temporisés à interruptions (ITA) où les informations sur l'écoulement du temps sont organisées en niveaux comparables à des niveaux d'accréditation. Nous étudions les propriétés de régularité et de clôture des langages temporisés générés par ces automates et proposons des algorithmes de *model-checking* pour des fragments de logiques temporelles temporisées. Ces résultats permettent la vérification de propriétés telles que « le système est (dans un état) *prêt* avant les 7 premières unités de temps de fonctionnement du système ».

Abstract

Information systems have become ubiquitous and are used to handle each day more and more data. This data is increasingly confidential: it may be strategic military or financial information, or private information. Any leakage of this data can be harmful in many different ways, such that human casualties, money loss, privacy breaching or identity theft.

The use of formal methods and especially formal verification of such systems has become necessary to ensure both the safety of the behavior of the system and the security of the information it handles. These techniques are applied on models of the system where some parameters – especially quantitative parameters – may have been abstracted. These omitted parameters have sometimes been used in order to breach the security of supposedly secure systems. Nowadays, models tend to include more of this information, which is often relative to the (continuous) elapsing of time or to the probability distributions that rule the system's behavior.

Models also include the architecture of the systems as several processes, who may have different, or even contradictory, goals: one may want to keep a secret while the other tries to discover it. When the behavior of all processes is partially or completely unknown, formal methods have to consider all possible cases, which is in general impossible (if there exist an infinite number of choice) or intractable (if there exist too many possibilities to consider them all).

The contributions of this thesis are threefold. First, we study the problem of synthesis of a communication channel inside a system given as a transducer. Even though the model of transducers is syntactically limiting, we show that this synthesis problem is undecidable in general. However, when the system is functional, meaning that its behavior from an external point of view is always the same, the problem becomes decidable.

We then generalize the concept of opacity to probabilistic systems, by giving measures separated in two groups. When the system is opaque, we evaluate the robustness of this opacity with respect to the bias induced by the probability distributions in the system. When the system is not opaque, we evaluate the size of the security hole opened by this non-opacity.

Finally, we study the model of Interrupt Timed Automata (ITA) where information about time elapsing is organized along levels, which therefore resemble accreditation levels. We study properties of regularity and closure of the time languages accepted by these automata and give some model-checking algorithms for fragments of timed temporal logics. These results allow to verify formulas such as "the system is (in state) *ready* before 7 time units have elapsed since the beginning of the system's execution".

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION

CHAPITRE 1

Sometimes, attackers try to slip information out slowly, hidden in ordinary internet traffic.

The Economist, 3-9 juillet 2010 [eco10].

1.1 Sûreté et sécurité des systèmes informatiques

Les années 1970 ont vu la conjonction de deux facteurs. L'un est la confrontation de deux grandes puissances militaires dans la Guerre Froide, accompagnée d'une escalade technologique pour garantir des communications sécurisées et maintenir secrètes certaines informations (stratégiques, techniques *etc*). L'autre est le début de la prolifération des systèmes d'information.

La crainte d'une attaque des systèmes militaires des années de la guerre froide a fait place à celle d'attaques sur des millions d'ordinateurs personnels à l'époque d'internet. Ce ne sont plus des secrets militaires qu'un attaquant va chercher à découvrir, mais des numéros de carte bancaire et d'autres informations personnelles. Cependant, le cas récent de Stuxnet [FMC10] montre que ces deux cibles d'attaque peuvent être combinées : Stuxnet se propageait via des millions d'ordinateurs privés, mais ne causait des dommages qu'à des cibles à applications militaires.

Ainsi, un important effort de recherche fut – et est encore – consacré à sécuriser les systèmes que ce soit de manière cryptographique [DH76, RSA78] ou en mettant en place des politiques de sécurité, qui garantissent, par exemple, qu'un document ne peut être lu qu'avec l'accréditation nécessaire.

Orthogonalement à cet effort de sécurisation de l'information, la fiabilité, et donc la sûreté de fonctionnement des systèmes, est un besoin de plus en plus critique dans les applications informatiques. L'apport des méthodes formelles dans ce domaine que ce soit par la modélisation de systèmes complexes ou par la vérification d'iceux a permis d'atteindre une grande confiance dans la sûreté d'applications critiques. Les techniques ainsi développées, en particulier le *model-checking* [QS82, EH82], sont désormais au cœur de tout processus de modélisation formelle et associées au développement de tous nouveaux formalismes.

1.1.1 Canaux et interférences

Des méthodes formelles ont été développées afin de vérifier que ces politiques de sécurité sont bien satisfaites. La première fut le critère de Bell et LaPadula [BL73].

Ce critère stipule qu'un utilisateur ne peut lire un document d'un niveau d'accréditation supérieur au sien (*no read up*), et ne peut écrire dans un document d'un niveau d'accréditation inférieur (*no write down*). Les niveaux d'accréditation des utilisateurs et documents sont donnés par des matrices, qui déterminent ensuite les actions autorisées. D'autres méthodes de calcul des autorisations reposent sur des techniques de typage [VSI96, SM03] : les variables sont typées selon le niveau de sécurité des informations qu'elles peuvent contenir suite à l'exécution du programme. Un terme est donc correctement typé si les contraintes de sécurité sont satisfaites.

À la même époque, Lampson [Lam73] invente le terme de *canal caché* (*covert channel*), l'opposant aux canaux de communication légitimes. Un canal de communication – légitime ou illégitime – est un système qui peut transmettre à travers ses actions de l'information entre un émetteur et un récepteur. Typiquement, un système qui lit un octet dans un disque dur puis écrit successivement les huit bits de l'octet lu sur la carte réseau est un canal de communication. Ces canaux sont plus ou moins fiables. L'exemple ci-dessus est par exemple un canal parfait, car toute l'information qui lui est fournie par le disque est transmise sur le réseau. Un système similaire qui ne transmettrait que les trois premiers bits serait lui imparfait. Si le système comporte d'autres fonctionnalités que celui de transmettre de l'information, on dit qu'il contient un canal potentiel : la transmission d'information dépend de la manière dont son environnement (en particulier le récepteur et l'émetteur) se comporte.

Il est cependant souvent difficile de différencier entre les canaux légitimes et les canaux cachés [Mil99], car la légitimité d'un canal dépend avant tout de l'intention du créateur du système et de la politique de sécurité, souvent implicite. Dans la suite, nous ne parlons donc plus que de canaux, sans préciser s'ils sont légitimes ou non. Les premiers exemples de canaux contournent les systèmes de sécurité par divers moyens, par exemple en utilisant des informations bien souvent abstraites dans les modèles de sécurité : l'utilisation des disques, l'activité de certains processus, une autorisation d'accès à une ressource, *etc*.

Cherchant à détecter ce genre de faiblesse dans les systèmes, la notion de *non interférence* a été proposée dans les années 1980 [GM82]. Ce critère sépare les actions du système en actions *secrètes* et en actions *publiques*. La non interférence est atteinte lorsque l'observation d'une exécution du système à travers ses actions publiques ne permet pas de déduire la présence d'une action secrète. C'est une notion assez contraignante, car une seule action secrète détectée rend le système non sûr selon ce critère.

Néanmoins, la manière dont le système est observé peut varier [FG01]. En effet, l'attaquant peut, selon le pouvoir qui lui est laissé dans le modèle de sécurité, ne disposer que d'une information très partielle sur l'exécution du système. Il peut donc confondre deux exécutions pour peu que celles-ci aient par exemple la même trace, ou soient bisimilaires, ou encore soient équivalentes vis-à-vis de l'équivalence de test.

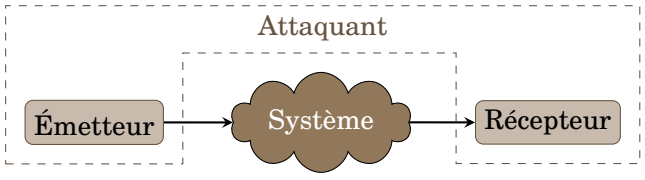
La forte sécurité induite par la propriété de non interférence [RMMG01] en fait une propriété souvent difficile à satisfaire dans le cas d'un système ne fonctionnant pas totalement séparément de son environnement. En particulier, le fait que la propriété de non interférence soit violée n'implique pas nécessairement l'existence d'un canal de communication ou d'une réelle menace sur la sécurité du système [HR10].

1.1.2 Vérification des propriétés de sécurité

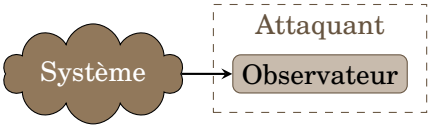
Des généralisations de propriétés de sécurité cherchent à englober divers critères [Maz04, BKMR08], voire à les combiner [Man00]. Ces approches sont bien adaptées à la modélisation de systèmes sûrs et de leur propriétés. Le niveau de détail qu'il est permis d'exprimer rend cependant la vérification impossible en général.

Afin de vérifier la sécurité des systèmes selon les divers critères, de nombreuses techniques issues d'autres domaines furent utilisées. Les critères proches de la non interférence nécessitent souvent la vérification de propriétés de bisimulation, et donc de jeux [Mar75]. Ces jeux sont potentiellement à connaissance incomplète [Rei79] afin de modéliser par le manque d'information l'imprécision dûe à une observation partielle. Plus généralement, le problème de recherche d'un canal peut être vu comme une recherche d'une *stratégie* de communication au travers du système, comme il est représenté sur la figure 1.1(a). Dans ce cas, le problème s'apparente à celui de la synthèse de processus dans le cas restreint de processus en chaîne [PR89a, KV01, FS05]. Ce cadre restreint est cependant rendu plus difficile ici par le fait que le système agit *contre* l'attaquant et non avec lui. Le système n'est donc pas un processus comme les autres, et toutes ses stratégies doivent être considérées. D'autre part, une communication asynchrone, bien que plus réaliste du point de vue de la modélisation, accroît la difficulté théorique du problème [PR89b, FS06].

Un attaquant ne dispose cependant pas toujours d'un émetteur, et doit se contenter d'observer le système pour en déduire de l'information, comme sur la figure 1.1(b). Dans ce cas, seule une stratégie de décodage est nécessaire. C'est aussi le cadre choisi par la théorie de l'opacité [BKMR08] : l'attaquant doit choisir, à partir de ses observations, si un prédicat donné est vrai ou faux sur l'exécution qui s'est réellement produite. S'il est possible pour l'attaquant d'avoir une réponse sûre à partir d'*une certaine* observation, le système n'est pas opaque. Notons que la théorie de la *diagnosticabilité* se place dans un cadre similaire, hormis que l'attaquant (qui est dans ce cas du « bon » coté) doit cette fois être *toujours* capable de déterminer la valeur de vérité du prédicat, afin de détecter des erreurs du système.



(a) Canal actif.



(b) Canal passif.

FIGURE 1.1: Principe d'un canal.

1.1.3 Mesure des flux d'information

Les approches citées plus haut cherchent à détecter la présence d'un canal, et non d'évaluer sa performance. La mesure de *capacité* [1] d'un canal permet d'évaluer son pouvoir de nuisance. Ainsi un flux d'information faible (une bande passante suffisamment faible) pourra être toléré [MK94]. Des implémentations de tels canaux [CBS04] ont permis de mesurer expérimentalement leur bande passante : de l'ordre de 16 octets par secondes. Cependant, comme le souligne Millen [Mil99], il est difficile d'évaluer le risque qu'un canal présente en terme de sécurité seulement à partir de sa bande passante. Si une bande passante de 16 octets par seconde est négligeable comparativement aux vitesses de transfert atteintes actuellement, cela devient un risque important si les octets ainsi transférés sont la valeur d'un clef privée, comme dans les attaques de Kocher [Koc96]. Cela reste néanmoins un critère quantitatif utile pour comparer divers systèmes.

La mesure de cette bande passante peut se faire de diverses façons. La théorie de l'information [Sha48, CT06] offre un cadre large et bien connu pour l'étude de ces propriétés. Il faut alors que le modèle intègre des informations de probabilités [Seg95]. Ces informations peuvent être une abstraction partielle du non déterminisme du modèle ; en effet, certains modèles autorisent à la fois le non déterminisme et les actions probabilistes, comme c'est le cas pour l'automate de la figure 1.2 [Seg95] : l'ordonnanceur peut choisir une pièce biaisée ou non avant que celle-ci ne soit tirée. Le non déterminisme restant est alors résolu à l'exécution du système à travers un ordonnanceur qui applique sur ce choix une distribution de probabilités. Ainsi, dans le cas du lancer de pièce précédent, l'ordonannceur peut, en pondérant son choix de pièce par une distribution de son choix, obtenir n'importe quelle distribution où la probabilité de tirer face (action h) est comprise entre $\frac{1}{2}$ et $\frac{3}{4}$. Toute évaluation quantitative, que ce soit dans le model-checking de logiques probabilistes ou dans l'évaluation de la sécurité, doit donc prendre en compte l'ensemble de toutes les distributions que l'ordonnanceur peut choisir, ce qui est en général impossible.

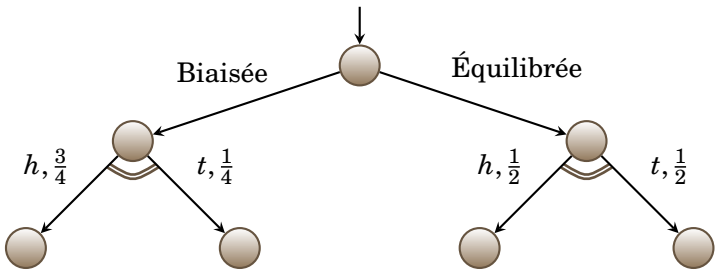


FIGURE 1.2: Automate probabiliste non déterministe de lancer d'une pièce.

On peut néanmoins noter que pour certaines propriétés, notamment les propriétés relatives aux traces d'exécutions, les stratégies optimales de l'attaquant sont obtenues comme solutions d'un problème linéaire [BdA95]. La fuite d'information du système ordonnancé peut alors être évaluée.

1. Ce terme est ici utilisé informellement, et non nécessairement dans le cadre de la théorie de l'information.

L'analyse de propriétés concernant des ensembles de traces du système (ou *hyperproperties* [CS10]) n'obéit malheureusement pas aux mêmes règles [GD09, AAP10]. Ces propriétés sont cependant les plus utilisées en sécurité : elle comprennent entre autres la non interférence [GM82], les propriétés d'opacité [Maz04, BKMR08], ainsi que les propriétés définies par Mantell [Man00]. Généraliser ces propriétés dans le cadre d'une étude quantitative demande donc des méthodes *ad hoc*.

Des calculs exacts ou des approximations de capacité [2] sont aussi possibles sous certaines hypothèses [GV96]. On rencontre cependant des systèmes très simples, tels que le *trapdoor channel* [Bla61] pour lesquelles aucune expression générale de la capacité n'est encore connue et elle n'est évaluée que par simulation [PCRW08].

1.2 Analyse des systèmes temporisés

Divers moyens s'offrent aux attaquants pour contourner les systèmes de sécurité. En particulier, il est possible d'utiliser des éléments qui ont été abstraits lors de la modélisation du système. Ainsi, une fuite d'information est possible malgré une étape de vérification formelle effectuée sur le modèle. Les attaquants peuvent donc se servir de données souvent peu modélisées car « de bas niveau » : la température du processeur, les interférences électriques [KA98], et le temps de réponse. Ce dernier est par exemple utilisé par Kocher ou Bernstein [Koc96, Ber04] pour récupérer les valeurs de clefs privées utilisées dans des protocoles tels que Diffie-Hellman [DH76] ou l'*Advanced Encryption Standard*. Il est aussi possible de transmettre de l'information non pas au travers du contenu de paquets TCP/IP, mais à travers leurs dates d'émission et de réception [CBS04]. Le temps, et plus particulièrement le temps continu, est donc devenu une dimension essentielle des systèmes et doit être pris en compte du point de vue de la sécurité de ceux-ci.

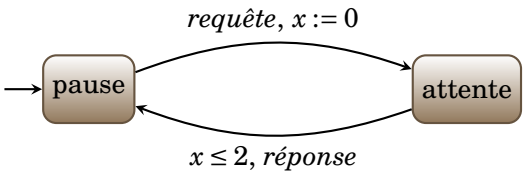


FIGURE 1.3: Automate temporisé $\mathscr{A}_1$ avec réponse bornée.

D'autre part, le temps est de plus en plus présent dans les modèles, depuis l'introduction des automates temporisés par Alur et Dill [AD94]. Un automate temporisé est un automate fini enrichi de variables évoluant continûment, toutes à la même vitesse (des *horloges*). Par exemple, l'automate temporisé $\mathscr{A}_1$ représenté figure 1.3 dispose d'une seule horloge x. Ces horloges peuvent être testées dans des *gardes* sur les transitions, telles que $x \leq 2$, et remises à zéro, ce qui est noté $x := 0$. Un tel automate accepte donc des mots temporisés où à chaque lettre est associée la date à laquelle elle s'est produite, par exemple, l'automate $\mathscr{A}_1$ accepte le mot $(\textit{requête}, 1.32)(\textit{réponse}, 2.76)(\textit{requête}, 5.29)(\textit{réponse}, 7.29)\ldots$. Plus globalement, cet au-

2. Au sens de la théorie de l'information.

tomate accepte tous les mots où chaque requête est suivie d'une réponse en moins de 2 unités de temps.

Bien que ce modèle dispose de bonnes propriétés de régularité, il est impossible de tester l'inclusion des langages d'un automate temporisé dans un autre. Ceci empêche de tester par exemple l'inclusion de certains motifs de canaux connus dans un système modélisé par un automate temporisé.

Que ce soit pour vérifier la sécurité de l'information manipulée par le système ou la sûreté de fonctionnement de celui-ci, on utilise des formules de logiques temporelles afin de spécifier le comportement correct du système. La modalité centrale de ces logiques est U (*jusqu'à*). La formule $p \,\mathsf{U}\, q$ signifie que p est vérifiée tout le long du chemin jusqu'à un instant où q l'est. Par exemple, la formule $\mathbf{true}\,\mathsf{U}\,ok$, que l'on abrège en $\mathsf{F}ok$, stipule que l'on atteindra un jour un état ok. Dualement, la formule $\neg(\mathbf{true}\,\mathsf{U}\,\neg ok)$, que l'on abrège en $\mathsf{G}ok$, stipule que l'on n'atteindra jamais un état qui n'est pas ok, ou encore que l'on est toujours dans un état ok.

Les formules de LTL [Pnu77, SC85] expriment des propriétés qui doivent être vraies sur toutes les exécutions du système, telles que $\mathsf{G}(\mathit{requête} \Rightarrow \mathsf{F}\mathit{réponse})$ qui exprime qu'une requête est toujours suivie d'une réponse [3]. Les formules de CTL [QS82, EH82, CES86] expriment des propriétés sur l'arbre des exécutions du système, c'est-à-dire qu'elles permettent de différencier les états où une erreur est possible de ceux ou l'erreur est inévitable. Un état où une erreur est possible, c'est-à-dire tel qu'*il existe une exécution* depuis cet état qui atteint un état d'erreur, satisfait la formule $\mathsf{EF}\mathit{erreur}$. Un état où l'erreur est inévitable satisfait la formule $\mathsf{AF}\mathit{erreur}$, qui signifie que *toute exécution* depuis cet état atteint un état d'erreur.

Ces logiques ne permettent pas de spécifier sur le temps continu, exprimant par exemple qu'une réponse suit une requête en moins de deux unités de temps. Plusieurs extensions temporisées furent donc considérées pour LTL et CTL, dessinant les limites de l'équilibre entre le pouvoir d'expression et la possibilité de décider la satisfaction d'une formule par un modèle (*model-checking*).

Dans le cas de LTL, on peut citer l'extension « directe » MTL [Koy90] qui ajoute des contraintes de temps sur la modalité U et ses dérivées par des intervalles. La propriété de réponse bornée s'exprime alors par la formule $\mathsf{G}(\mathit{requête} \Rightarrow \mathsf{F}_{[0,2]}\mathit{réponse})$, qui spécifie qu'à chaque instant, si une requête se produit alors le système produira une réponse à un moment entre maintenant (car 0 est la borne inférieure de l'intervalle) et dans 2 unités de temps. Le model-checking de MTL est indécidable, alors que sa restriction MITL [AFH96], qui interdit de spécifier des durées exactes (c'est-à-dire des intervalles comme $[2,2]$), peut être vérifiée sur les automates temporisés.

Pour CTL, deux variantes de CTL temporisée (TCTL) ont été proposées. La première imposant des contraintes de temps sur les modalités [ACD93] : la propriété de réponse bornée dans ce cas s'écrit $\mathsf{AG}(\mathit{requête} \Rightarrow \mathsf{AF}_{\leq 2}\mathit{réponse})$. L'autre variante donne explicitement des contraintes sur des nouvelles horloges de la formule en tant que propositions atomiques [HNSY94] ; la propriété de réponse bornée devient alors $\mathsf{AG}(\mathit{requête} \Rightarrow y.(\mathsf{AF}(\mathit{réponse} \wedge y \leq 2))$, où « $y.$ » correspond à la remise à zéro de l'horloge de formule y.

Dans toutes ces extensions, il est nécessaire de bien définir la sémantique des

3. Remarquons qu'une réponse n'est pas associée à une requête en particulier et une réponse peut donc « servir » toutes les requêtes qui la précèdent.

logiques de manière à déterminer ce que signifie « être vraie à un instant donné ». Par exemple, dans TCTL, la formule $y.(\mathsf{A}\,\mathsf{G}\,y \leq 2\,\mathsf{U}\,y > 2)$ est toujours vraie, alors qu'il n'y a pas *un instant* auquel la formule $y > 2$ devient vraie.

Complémentairement, divers formalismes, comparables ou non à celui des automates temporisés, ont été développés dans le but de modéliser des systèmes temporisés. Ici encore, il y a un équilibre entre l'expressivité du modèle et les possibilités de vérification. Le problème le plus simple, car exprimable dans toutes les logiques temporelles, que l'on cherche à résoudre sur ces automates est celui de l'accessibilité d'un état de contrôle. Par exemple, vérifier qu'un état d'erreur n'est pas accessible permet de montrer la sûreté du système.

Le modèle le plus général est celui des automates hybrides [Hen96], car il permet de modéliser non seulement le temps, mais n'importe quelle variable physique dont la variation est régie par des équations différentielles. Mais le problème d'accessibilité est indécidable pour les automates hybrides (même pour les automates hybrides linéaires dont les dérivées sont des constantes), rendant leur utilité limitée pour la vérification. Malgré leur syntaxe plus limitée, le modèle des automates à chronomètres [CL00], autorisant les horloges à cesser d'évoluer puis à reprendre leur cours, est aussi indécidable pour l'accessibilité (en réalité ce modèle est équivalent à celui des automates hybrides linéaires). De nombreuses restrictions ont alors été définies de sorte à rendre l'accessibilité décidable. Le prix de ces restrictions est une expressivité moindre, bien que chaque formalisme soit défini pour permettre la modélisation de certains aspects du système.

Certains formalismes restreignent syntaxiquement les variations du taux d'évolution des variables pour permettre la décidabilité de l'accessibilité. Parmi ces modèles on trouve les systèmes à dérivées constantes par morceaux [AMP95], les automates temps réel contrôlés [DZ98]. Les gardes peuvent elles aussi être limitées, comme dans le cas des automates rectangulaires multi-vitesses [ACH+95], certains graphes d'intégration [KPSY99], ou les sytèmes hybrides polygonaux [ASY07]. D'autre part, restreindre les remises à zéro peut aussi rendre le modèle décidable, par exemple dans les automates hybrides à remises à zéro fortes [BBJ+08] et les automates à chronomètres initialisés [HKPV98]. Les systèmes hybrides O-minimaux [LPY99, LPY01] sont une caractérisation algébrique de contraintes placées sur un modèle d'automates hybrides afin de rendre le problème d'accessibilité décidable. Adoptant le point de vue de l'extension du modèle des automates temporisés qui gardent la décidabilité du problème d'accessibilité, plutôt que de la restriction des automates hybrides, les automates temporisés avec mises à jour [Bou04, BCFL04] permettent des affectations plutôt que des remises à zéro. Dans le but d'étudier des problèmes d'ordonnancement, des modèles *ad hoc* tels que les automates de tâches [FKPY07] et les automates à suspensions [MV94] furent définis, ainsi que les automates temporisés à interruptions [BH09]. Ces derniers modélisent explicitement les interruptions pouvant avoir lieu dans les systèmes d'eploitation [SGG08]. De plus, ils contiennent syntaxiquement une hiérarchisation des horloges analogue à la hiérarchisation des niveaux d'accréditation.

1.3 Contributions et organisation de la thèse

Le but de cette thèse est de définir et d'étudier des modèles :
- suffisamment expressifs pour représenter le comportement d'un système complexe ;
- suffisamment simples d'un point de vue théorique de sorte à pouvoir décider si une fuite d'information est possible ;
- prenant en compte non seulement les aspects qualitatifs mais aussi quantitatifs du système.

Le chapitre 2 donne les notations et rappelle les définitions des modèles utilisés dans la suite. En particulier, tous les modèles de systèmes sont *in fine* des systèmes de transitions, pas nécessairement finis. Les exécutions du système sont alors des chemins finis ou infinis dans ce système de transitions. Le cas le plus simple est celui des automates finis, où le système est fini et des lettres étiquettent les transitions. Ces automates sont étendus de plusieurs manières, de sorte qu'ils puissent exprimer, respectivement, les mécanismes d'entrée/sortie du système, les probabilités régissant le fonctionnement de celui-ci, l'écoulement du temps dans une exécution du système.

Les transducteurs [Ber79] sont des automates finis étiquetées par des mots d'entrée et de sortie. Ainsi ce ne sont plus des mots qui sont acceptés par l'automate mais des paires de mots, le langage formé par ces paires est donc une relation binaire (dite rationnelle).

L'extension probabiliste consiste à attribuer des probabilités aux transitions d'un automate fini afin d'en abstraire le non déterminisme. Ceci produit le modèle des automates purement probabilistes (un cas particulier des automates probabilistes de [Seg95]).

Enfin, l'extension temporisée ajoute au système des horloges à valeurs réelles qui peuvent être testées et remises à zéro. Ce modèle des automates temporisés [AD90, AD94] a pour sémantique un système de transitions infini (non dénombrable) car chaque configuration du système dépend de la valeur des horloges. Les exécutions d'un tel automate acceptent donc des mots temporisés où à chaque lettre est associée la date où elle a été lue.

Outre les différent modèles, on rappelle les définitions de propriétés de sécurité de la littérature, telles que la non interférence [GM82] et l'opacité [Maz04].

Nous définissons dans le chapitre 3 un modèle de canaux pour des systèmes décrits par des transducteurs. Plus précisément, un canal existe au sein d'un système modélisé par un transducteur s'il existe une paire de transducteurs (l'encodeur et le décodeur) qui permettent de transmettre parfaitement une information, sous la forme d'un message binaire fini : le message de sortie doit être identique au message d'entrée. Nous montrons en effet qu'il est possible de pallier un certain nombre d'erreurs de transmission par un envoi redondant de données, et nous concentrons ainsi sur le cas théoriquement plus simple de la recherche d'une transmission parfaite d'information.

Ce modèle allie la puissance algébrique des transducteurs à la facilité qu'ont les transducteurs à modéliser des processus d'entrée/sortie, de manière asynchrone. La détection de canal dans ce cadre s'apparente donc à un cas particulier de synthèse asynchrone [FS06] : les processus à synthétiser sont des transducteurs dans une

chaîne et la spécification est la transmission parfaite d'information. Nous montrons cependant que le problème général de détection de canaux dans ce formalisme est indécidable, malgré la simplicité à la fois de l'architecture de communication et de la spécification.

Cependant lorsque le modèle est fonctionnel (chaque entrée produit au plus une sortie, ce qui est une forme faible de déterminisme), l'existence de canaux de communication au sein du système devient une propriété structurelle. Il suffit alors de détecter dans le système un état dit *encodant*, dans un sens proche de la définition de [HZD05], à partir duquel deux séquences formant un code en entrée et en sortie peuvent être répétées. Notons que la propriété d'état encodant, qui est nécessaire et suffisante dans le cas où le système est fonctionnel, est dans le cas général une propriété nécessaire à l'existence d'un canal.

Nous comparons ce modèle de sécurité à celui de la non interférence, en montrant que ce sont deux propriétés incomparables.

Dans le cas des transducteurs, la détection du canal est purement qualitative. On ne cherche en particulier pas à calculer ni à optimiser la bande passante du canal ainsi détecté. Il faut noter que le problème de mesure de la bande passante [AD10] est assez orthogonal à celui de la détection de l'existence d'un canal parfait. En effet, un codage plus efficace mais moins précis peut au final produire un canal (imparfait) avec une meilleure bande passante.

Ces résultats ont été partiellement publiés dans [BBL+09] et [BBL+11].

Afin de donner une mesure plus quantitative de la fuite d'information induite par un canal, nous cherchons, dans le chapitre 4, à définir une mesure de l'opacité. Un prédicat φ sur les exécutions d'un système est opaque vis-à-vis d'une fonction d'observation $\mathcal{O}$ si un attaquant ne voyant une exécution ρ du système qu'à travers $\mathcal{O}$ ne peut jamais être sûr que l'exécution ρ satisfait φ. L'opacité est un cadre général pour définir des propriétés de sécurité [BKMR08] : il suffit d'instancier le prédicat et la fonction d'observation.

La mesure de l'opacité est ici effectuée à travers des probabilités sur le système, qui abstraient le non déterminisme de celui-ci. La quantification de l'opacité permet donc d'évaluer, après instanciation, la sécurité du système vis-à-vis, par exemple, de la non interférence ou de l'anonymat, comme il est fait dans une étude de cas sur le protocole *Crowds* [RR98].

Lorsque l'opacité n'est pas vérifiée, on cherche à évaluer le risque que cela induit en terme de sécurité. La fuite de l'information est d'autant plus grande qu'une observation du système qui trahit la propriété est probable : c'est donc ce qui est retenu comme critère pour la mesure d'opacité dans ce cas.

Outre l'efficacité avec laquelle un attaquant peut obtenir de l'information sur le système, on veut pouvoir mesurer la robustesse de la sécurité du système. On cherche donc à mesurer, pour chaque observation possible de la part de l'attaquant, la connaissance que lui fournit cette observation. Par exemple, si un son *bip* est le résultat dans 99% des cas de l'arrivée d'un message, tout observation du *bip* trahit presque sûrement l'arrivée d'un message.

Si l'on regarde cette connaissance dans l'absolu, on obtient une mesure de l'opacité proche de la définition classique. Cependant, si l'on compare la connaissance de l'attaquant avant et après l'observation, on obtient une mesure qui se rapproche de celle

l'information mutuelle en théorie de l'information [Smi09]. Dans ce cas la mesure est plus éloignée de la notion d'opacité mais peut devenir intéressante complémentairement aux mesures d'opacité dans l'évaluation de la sécurité d'un système.

Nous définissons donc plusieurs mesures de l'opacité, applicables dans des cas différents, selon que l'on cherche à quantifier la fuite d'information due à la non-opacité ou la robustesse de cette même opacité. Ces mesures sont intanciées dans des exemples parfois paramétrés, afin d'obtenir une mesure d'opacité en fonction de ces paramètres.

On montre de plus qu'il est possible de calculer automatiquement leur valeur lorsque les classes d'observation (en nombre fini) et le prédicat sont des ensembles réguliers.

Une partie des résultats présentés dans ce chapitre a été publiée dans [BMS10].

Nous cherchons à prendre en compte le temps comme vecteur d'information avec l'étude, dans le chapitre 5, du modèle des automates temporisés à interruptions (ITA). Dans ce modèle, le temps est organisé en niveaux de sorte qu'une horloge d'un certain niveau ne peut lire les valeurs que des horloges de niveau plus bas, et ne peut y écrire que des valeurs de niveau inférieur au niveau écrit. Cette implémentation du critère de Bell et LaPadula (*no read up* et *no write down*) dans la syntaxe fournit au modèle de bonnes propriétés de régularité. De plus, cette organisation en niveaux imite celle qui est présente dans les systèmes d'exploitation : les ITA peuvent par exemple être utilisés lors de la modélisation de noyaux temps-réels.

Le critère n'est pas interprété mot à mot : on a en effet le droit d'écrire dans un niveau inférieur, pourvu que ce que l'on écrit eût aussi pu être lu directement depuis ce niveau. Les écritures au niveau supérieur (*no write up*) sont quant à elles interdites syntaxiquement, mais peuvent être implémentées par une lecture depuis le niveau supérieur. Nous étudions par ailleurs ce modèle pour lui même, du point de vue de l'expressivité et du *model-checking*.

Nous définissons aussi les ITA$_-$, une classe restreinte des ITA qui interdit l'écriture à des niveaux inférieurs, interprétant alors plus strictement le critère de Bell et LaPadulla. Nous montrons qu'elle est aussi expressive que les ITA sans cette restriction, bien que moins succincte. De plus, cette classe permet d'atteindre une meilleure complexité sur le problème d'accessibilité d'un état de contrôle dans les ITA.

Nous montrons que les ITA ont, comme les automates temporisés, un langage non temporisé régulier, à travers la construction d'un graphe des classes généralisant les régions [AD94]. Cela permet d'adapter directement des algorithmes de model-checking de propriétés non temporisées. Il faut cependant des procédures *ad hoc* pour le cas des extensions temporisées des logiques temporelles. En fait, le model-checking sur les ITA est en général indécidable pour MITL et nous conjecturons que c'est aussi le cas pour TCTL. Nous identifions deux fragments décidables de TCTL : l'un ne portant que sur les horloges internes du système, l'autre restreignant l'imbrication de la modalité U (*jusqu'à*). Dans le premier fragment, l'algorithme repose sur une version plus fine du graphe des classes, dépendant de la formule à vérifier. Le second fragment permet d'exprimer des propriétés d'urgence et de délai sur l'exécution du système. L'algorithme de model-checking se base pour partie sur ce graphe des classes et pour partie sur des arguments de comptage sur l'ITA$_-$ équivalent.

Du point de vue de l'expressivité, nous montrons que la classe des langages définis par un ITA est incomparable avec celle des langages définis par un automate temporisé. Nous donnons une extension qui englobe syntaxiquement ces deux classes et qui garde néanmoins de bonnes propriétés.

Une partie de ces résultats a été publiée dans [BHS10].

Enfin, le chapitre 6 résume la thèse et discute des perspectives d'extension des résultats présentés.

Notations

Chapitre 2

No one definition has as yet satisfied all naturalists.

Charles Darwin, *On the Origin of Species*, 1859.

Ce chapitre rappelle les définitions et notations usuelles utilisées dans la suite de la thèse.

2.1 Ensembles

On note $\mathbb{B}$ l'ensemble des booléens ($\{0,1\}$ ou $\{\bot,\top\}$ selon le contexte), $\mathbb{N}$ les entiers naturels, $\mathbb{Z}$ les entiers, $\mathbb{Q}$ les nombres rationnels, $\mathbb{R}$ les nombres réels, munis de leur structure et ordre usuels. Les nombres rationnels (respectivement réels) positifs sont notés $\mathbb{Q}_{\geq 0}$ (respectivement $\mathbb{R}_{\geq 0}$).

Pour deux ensembles A et B disjoints, on note $A \uplus B$ leur union disjointe. Le complémentaire de A est noté $\overline{A}$. L'ensemble des sous-ensembles de A est $\mathscr{P}(A)$.

2.2 Probabilités et théorie de l'information

Pour un ensemble dénombrable Ω, une *distribution discrète* aussi appelée *loi de probabilité* [CT06] (resp. une *sous-distribution* ou *loi de sous-probabilité*) est une fonction $\mu : \Omega \to [0,1]$ telle que $\sum_{\omega\in\Omega} \mu(\omega) = 1$ (resp. telle que $\sum_{\omega\in\Omega} \mu(\omega) \leq 1$). L'ensemble des distributions (resp. des sous-distributions) sur Ω est noté $\mathbb{D}(\Omega)$ (resp. $\underline{\mathbb{D}}(\Omega)$). Un sous-ensemble E de Ω est appelé un *évènement* et $\mu(E) = \sum_{\omega\in E} \mu(\omega)$. Pour deux évènements E et E', leur intersection sera notée $E \wedge E'$ ou E,E'. La loi de Bayes définit les *probabilités conditionnelles* par

$$\mu\big(E,E'\big) = \mu\big(E|E'\big)\cdot\mu\big(E'\big) = \mu\big(E'|E\big)\cdot\mu(E)\,.$$

Une variable aléatoire à valeurs dans Γ est une fonction $Z : \Omega \to \Gamma$ où $[Z = z]$ dénote l'évènement $\{\omega \in \Omega \mid Z(\omega) = z\}$.

L'*entropie* (*de Shannon*) d'une variable aléatoire Z donne une mesure de l'incertitude sur cette variable [Sha48]. Elle est définie par :

$$H(Z) = -\sum_{z\in\Gamma} \mu(Z=z)\cdot\log\big(\mu(Z=z)\big)\,.$$

La fonction log désigne le logarithme de base 2.

Pour deux variables aléatoires $Z : \Omega \to \Gamma$ et $Z' : \Omega \to \Gamma'$, l'entropie conditionnelle de Z sachant $Z' = z'$ (tel que $\mu(Z' = z') > 0$) est donnée par :

$$H\left(Z|Z' = z\right) = -\sum_{z \in \Gamma} \mu\left(Z = z|Z' = z'\right) \cdot \log\left(\mu\left(Z = z|Z' = z'\right)\right)$$

et l'*entropie conditionnelle* de Z sachant Z' est

$$\begin{aligned} H\left(Z|Z'\right) &= \sum_{z' \in \Gamma'} \mu\left(Z' = z'\right) \cdot H\left(Z|Z' = z'\right) \\ &= -\sum_{z \in \Gamma} \sum_{z' \in \Gamma'} \mu\left(Z = z, Z' = z'\right) \cdot \log\left(\mu\left(Z = z|Z' = z'\right)\right) \end{aligned}$$

L'entropie conditionnelle peut être vue comme une mesure moyenne de l'incertitude sur Z après avoir observé Z'. Ainsi l'information mutuelle entre Z et Z' est ce que l'on peut apprendre de l'une en n'observant que l'autre :

$$I\left(Z;Z'\right) = H(Z) - H\left(Z|Z'\right) = H\left(Z'\right) - H\left(Z'|Z\right) = I\left(Z';Z\right).$$

Une autre façon d'envisager la connaissance possible sur une variable aléatoire est sa *vulnérabilité* qui évalue la probabilité de deviner correctement en un seul coup la valeur prise par la variable aléatoire [Smi09]. Elle est donc définie comme la probabilité de l'évènement le plus probable

$$V(Z) = \max_{z \in \Gamma} \mu(Z = z).$$

La vulnérabilité conditionnelle de Z sachant Z' est la moyenne (pondérée) des vulnérabilités pour chaque possibilité pour Z' et est donnée par

$$V\left(Z|Z'\right) = \sum_{z' \in \Gamma'} \mu\left(Z' = z'\right) \cdot V\left(Z|Z' = z'\right) = \sum_{z' \in \Gamma'} \mu\left(Z' = z'\right) \cdot \max_{z \in \Gamma} \mu\left(Z = z|Z' = z'\right).$$

De manière similaire à l'entropie de Shannon, une notion d'entropie et d'entropie conditionnelle sont définies relativement à la vulnérabilité. Ainsi la *min-entropie* et la *min-entropie conditionnelle* sont définies respectivement par

$$H_\infty(Z) = -\log(V(Z)) \qquad \text{et} \qquad H_\infty\left(Z|Z'\right) = -\log\left(V\left(Z|Z'\right)\right).$$

Ces notions correspondent à la limite de l'entropie de Rényi [Rén60] quand le paramètre tend vers l'infini, d'où la notation H_∞ (l'entropie de Shannon étant l'entropie de paramètre 1). La min-information mutuelle correspond donc à la façon dont l'observation d'une variable aléatoire affecte la vulnérabilité de l'autre :

$$I_\infty\left(Z;Z'\right) = H_\infty(Z) - H_\infty\left(Z|Z'\right).$$

Contrairement à l'information mutuelle mesurée à l'aide de l'entropie de Shannon, cette notion n'est pas symétrique.

2.3 Mots, monoïdes, langages

Pour un alphabet fini A, l'ensemble des mots sur A est noté A^*, et le mot vide ε. L'opérateur de concaténation est noté $\cdot$ (ou omis) et $\langle A^*, \cdot, \varepsilon \rangle$ est un monoïde libre (engendré par A). Un langage sur A est un sous ensemble de A^*.

La *longueur* d'un mot w est notée $|w|$, et la i-ième lettre d'un mot, pour $1 \leq i \leq |w|$ est notée $w[i]$. Pour un sous-ensemble B de A, on note π_B la *projection* sur B, définie comme le morphisme de A^* dans B^* tel que, pour $a \in A \setminus B$, $\pi_B(a) = \varepsilon$ et pour $a \in B$, $\pi_B(a) = a$.

Pour deux mots $v, w \in A^*$, v est un *préfixe* de w, noté $v \preceq w$, s'il existe un mot $u \in A^*$ tel que $w = v \cdot u$. Si $u \neq \varepsilon$, v est un *préfixe strict* de w.

Pour deux alphabets A et B, le produit $A^* \times B^*$ est un monoïde (non libre [1]). Il reste cependant, tout comme A^*, un monoïde *simplifiable*, c'est-à-dire que pour $x, y, z \in A^* \times B^*$, $x \cdot y = x \cdot z \Rightarrow y = z$ et $y \cdot x = z \cdot x \Rightarrow y = z$.

2.4 Automates

Un système de transition est un graphe étiqueté $\mathscr{T} = \langle S, Lab, \Delta, I \rangle$ avec :
- S un ensemble d'états ;
- Lab un ensemble d'étiquettes ;
- $\Delta \subseteq S \times Lab \times S$ est la relation de transition, une transition $(s, a, s') \in \Delta$ est notée $s \xrightarrow{a} s'$;
- $I \subseteq S$ est l'ensemble des état initiaux.

Lab n'est pas nécessairement un alphabet fini ; il peut être, entre autres, un monoïde, un ensemble continu ($\mathbb{R}$ par exemple)...

Exécutions. Une *exécution* dans le système de transition depuis un état s_0 est un chemin étiqueté dans $\mathscr{T}$. La *trace* d'une exécution est le mot formé par la suite des étiquettes de ce chemin :

$$\rho = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s_n \qquad tr(\rho) = a_1 \cdots a_n$$

On note $\mathrm{fst}(\rho) = s_0$ le premier état de ρ et $\mathrm{lst}(\rho) = s_n$ son dernier état. Deux exécutions ρ et ρ' peuvent être concaténées en $\rho \cdot \rho'$ si $\mathrm{lst}(\rho) = \mathrm{fst}(\rho')$.

L'ensemble des exécutions de $\mathscr{T}$ débutant dans un état s est noté $Exec_s(\mathscr{A})$. L'ensemble des exécutions de $\mathscr{T}$ dont le premier état est initial est noté simplement $Exec(\mathscr{A})$. La clôture réflexive transitive de Δ est notée $\Rightarrow$: $s \overset{w}{\Rightarrow} s'$ s'il existe une exécution de s à s' de trace w.

Automates finis. Un *automate fini* (ou simplement un automate) est un système de transition fini muni d'un ensemble d'états finaux. À ce titre il hérite des notations définies pour les systèmes de transition. Formellement, $\mathscr{A} = \langle S, Lab, \Delta, I, F \rangle$ où $\langle S, Lab, \Delta, I \rangle$ est un système de transition, S et Δ sont finis et $F \subseteq S$ est l'ensemble des états finaux.

Une exécution est *acceptée* par $\mathscr{A}$ si elle débute dans un état initial et termine dans un état final : $s_0 \in I$ et $s_n \in F$. Un état $s \in S$ est *accessible* (resp. *co-accessible*) s'il existe un état $s' \in I$ (resp. $s' \in F$) et un mot w tel que $s' \overset{w}{\Rightarrow} s$ (resp. $s \overset{w}{\Rightarrow} s'$).

Le *langage* de $\mathscr{A}$, noté $\mathscr{L}(\mathscr{A})$ est l'ensemble des traces des exécutions acceptées par $\mathscr{A}$. Un langage régulier sur un alphabet fini A est un langage accepté par un automate avec $Lab = A$. On peut considérer que $\mathscr{A}$ est *émondé*, c'est-à-dire que tout état est à la fois accessible et co-accessible, sans que cela ne change le langage accepté. Rappelons que tout langage régulier peut être accepté par un automate déterministe et complet : dans chaque état q et pour chaque lettre $a \in Lab$ il existe un unique état s' tel que $s \xrightarrow{a} s' \in \Delta$.

1. Si A et B ne sont pas vides

Transducteurs et relations rationnelles. Pour deux alphabets finis A et B, un *transducteur* [Ber79, Sak03] de A^* vers B^* est un automate fini $\mathscr{A}$ avec $Lab = A^* \times B^*$. On peut voir un transducteur comme une machine qui transforme ses *entrées* (sur A) en *sorties* (sur B). L'étiquette (v,w) est parfois écrite $(v|w)$. On note donc $s \xrightarrow{v|w} s'$ lorsque $(s,(v,w),s') \in \Delta$. On note également $s \overset{v|w}{\Longrightarrow} s'$ lorsqu'un chemin étiqueté par $v|w$ relie s à s'. Il faut remarquer que, comme évoqué précédemment, la concaténation de $A^* \times B^*$ est celle utilisée dans le transducteur.

Le langage de $\mathscr{A}$ est alors une *relation rationnelle* $\mathscr{M} \subseteq A^* \times B^*$. Ainsi, pour un mot $v \in A^*$, l'*image* de v par $\mathscr{M}$ est $\mathscr{M}(v) = \{w \in B^* \mid (v,w) \in \mathscr{M}\}$. De même, $\mathscr{M}^{-1}(w) = \{v \in A^* \mid (v,w) \in \mathscr{M}\}$ est l'*image inverse* du mot $w \in B^*$ par $\mathscr{M}$. La relation $\mathscr{M}$ (resp. $\mathscr{M}^{-1}$) est étendue aux langages sur A (resp. B). Ainsi, $Im(\mathscr{M}) = \mathscr{M}(A^*)$ est l'*image* de $\mathscr{M}$ et $Dom(\mathscr{M}) = \mathscr{M}^{-1}(B^*)$ est son *domaine*. Si $Dom(\mathscr{M}) = A^*$, on dit que $\mathscr{M}$ est *complet*.

L'image et le domaine d'une relation rationnelle sont des langages réguliers. La composition de deux relations $\mathscr{M} \subseteq A^* \times B^*$ et $\mathscr{M}' \subseteq B^* \times C^*$ est notée $\mathscr{M} \cdot \mathscr{M}' \subseteq A^* \times C^*$. La composition de deux relations rationnelles, l'inverse d'une relation rationnelle, sont des relations rationnelles.

Lorsque $\mathscr{M}(u)$ est un singleton, il est identifié, abusivement, avec son unique élément. Si $\mathscr{M}(u)$ est soit vide, soit un singleton, la relation $\mathscr{M}$ est *fonctionnelle* (on dit aussi que $\mathscr{M}$ est une *fonction rationnelle*) ; le transducteur $\mathscr{A}$ est alors *fonctionnel*.

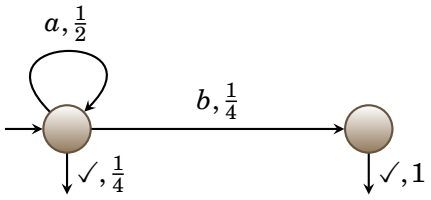
L'identité sur A est la relation $Id(A) = \{(w,w) \mid w \in A^*\}$.

Automates avec probabilités. Un automate sous-stochastique est un automate fini où la relation de transition est munie d'une loi de sous-probabilité. Dans un état, la transition suivante est choisie aléatoirement selon cette sous-distribution. Ce modèle se rapproche de celui de Segala [Seg95], bien que le formalisme utilisé ici ne contienne pas de non déterminisme. La transition comportant à la fois l'étiquette et l'état atteint, ce modèle s'apparente plus à une chaîne de Markov [Mar06] étiquetée qu'à un processus de décision Markovien [Bel57]. De même, la condition d'acceptation est définie de manière stochastique : dans un état, l'exécution peut s'arrêter on non. On introduit pour modéliser cette terminaison un nouveau symbole $\checkmark$.

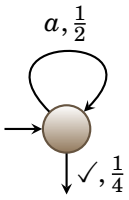
Un *automate sous-stochastique* (SA) est un quadruplet $\mathscr{A} = \langle S, Lab, \Delta, s_I \rangle$ où

- S est un ensemble fini d'états ;
- Lab est un ensemble fini d'étiquettes ;
- $\Delta : S \to \underline{\mathbb{D}}((Lab \times S) \uplus \{\checkmark\})$ est une fonction telle que pour chaque état $s \in S$, $\Delta(s)$ est une sous-distribution sur les paires de lettres et d'état $(Lab \times Q) \uplus \{\checkmark\}$:
$$\sum_{x \in (Lab \times Q) \uplus \{\checkmark\}} \Delta(s)(x) \leq 1;$$
- $s_I \in S$ est l'état initial.

Dans chaque état, la fonction Δ définit le poids de chaque transition sortante. Un *automate probabiliste* (FPA) est un automate sous-stochastique pour lequel, pour chaque état $s \in S$, Δ est une distribution :
$$\sum_{x \in (Lab \times Q) \uplus \{\checkmark\}} \Delta(s)(x) = 1$$
et de chaque état il existe un chemin de probabilité non-nulle vers l'action $\checkmark$. Par exemple sur la figure 2.1, l'automate $\mathscr{A}_2$ est un automate probabiliste, tandis que l'automate $\mathscr{A}_3$ est un automate sous-stochastique.



(a) L'automate probabiliste $\mathscr{A}_2$.



(b) L'automate sous-stochastique $\mathscr{A}_3$.

FIGURE 2.1: $\mathscr{A}_3$ est la restriction de $\mathscr{A}_2$ à a^*.

Il y a un seul état initial dans ces automates car tout le non déterminisme est abstrait dans les probabilités.

On note $s \to \mu$ si $\Delta(s) = \mu$ et $s \xrightarrow{a} s'$ si $\mu(a,s') > 0$. De plus, on note $s \cdot \checkmark$ si $\mu(\checkmark) > 0$; dans ce cas s est dit *final*.

Avec ces notations, on obtient un automate fini $Unprob(\mathscr{A})$. On définit donc les exécutions et les notations associées de la même manière que dans les systèmes de transition.

On appelle *exécution totale* une exécution qui se termine par $\checkmark$. On note $TExec(\mathscr{A})$ (resp. $TExec_s(\mathscr{A})$) les exécutions totales de $\mathscr{A}$ (resp. qui commencent en s). Le langage de $\mathscr{A}$ est donc $\mathscr{L}(\mathscr{A}) = tr\big(TExec_{s_I}(\mathscr{A})\big)$.

L'ensemble des exécutions totales commençant dans s_I est muni d'une loi de sous-probabilités définie inductivement par

$$\begin{aligned} \mathbf{P}_{\mathscr{A}}(s \cdot \checkmark) &= \mu(\checkmark) \\ \mathbf{P}_{\mathscr{A}}(s \xrightarrow{a} \rho) &= \mu(a,s') \cdot \mathbf{P}_{\mathscr{A}}(\rho) \end{aligned}$$

où $s \to \mu$ et $\mathrm{fst}(\rho) = s'$. Cette loi devient une loi de probabilités lorsque $\mathscr{A}$ est un automate probabiliste. En effet, dans ce cas tout chemin dans l'automate peut être prolongé en une exécution totale, c'est-à-dire atteindre le symbole $\checkmark$. On a donc bien $\mathbf{P}(TExec(\mathscr{A})) = 1$.

Automates temporisés. Un automate temporisé est un automate fini muni d'horloges évoluant toutes à la même vitesse et interprétées sur $\mathbb{R}_{\geq 0}$ [AD90, AD94]. Les horloges mesurent le temps qui s'écoule, et peuvent être testées et remises à zéro.

Les horloges de X peuvent être remises à zéro par des mises à jour de la forme $x := 0$. L'ensemble $\mathscr{U}_0(X)$ est formé de conjonctions de remises à zéro ou d'absence d'affectation ($x := x$, souvent omises). Un élément $u \in \mathscr{U}_0(X)$ peut donc être écrit $\bigwedge_{x \in X} x := C_x$ où $C_x \in \{0, x\}$. On note $\mathscr{C}_0(X)$ est l'ensemble des conditions formées de conjonctions de contraintes de la forme $x + b \bowtie 0$ où b est une constante rationelle et $\bowtie \in \{<, \leq, =, \geq, >\}$ est un opérateur de comparaison. Les mises à jour et les conditions sont généralisées avec des expressions linéaires (voir chapitre 5).

Un *automate temporisé* (TA) est un sextuplet $\mathscr{A} = \langle S, Lab, X, \Delta, s_I, F \rangle$ où

- S est un ensemble fini d'états ;
- Lab est un ensemble fini d'étiquettes ;
- X est un ensemble fini d'horloges ;
- $\Delta \subseteq S \times \mathscr{C}_0(X) \times (Lab \uplus \{\varepsilon\}) \times \mathscr{U}_0(X) \times S$ est la relation de transition : chaque transition est composée d'un état source, d'une *garde* (une condition sur X), d'une

étiquette, de *remises à zéro*, et d'un état cible ; une transition $(s,g,a,u,s') \in \Delta$ est aussi notée $s \xrightarrow{g,a,u} s'$.
- $s_I \in S$ est l'état initial ;
- $F \subseteq S$ est l'ensemble des états finaux.

Une *valuation* des horloges est une fonction $v : X \to \mathbb{R}_{\geq 0}$ qui associe à chaque horloge sa valeur. La valuation $\mathbf{0}$ désigne la valuation où toutes les horloges valent 0.

Lorsque $v(x) \bowtie 0$, on écrit $v \vDash x \bowtie 0$; cette notation s'étend aux conditions (en tant que conjonctions de telles contraintes). Pour une mise à jour $u = \bigwedge_{x \in X} x := C_x$, pour toute horloge $x \in X$, la valuation en x, $v[u](x)$ est donnée par $v[u](x) = v(x)$ si $C_x = x$ et $v[u](x) = 0$ si $C_x = 0$. Pour $d \in \mathbb{R}_{\geq 0}$, la valuation $v + d$ est définie, pour $x \in X$, par $(v+d)(x) = v(x) + d$.

Les transitions ne peuvent être franchies que lorsque la garde est satisfaite. Entre deux transitions, le temps s'écoule à la même vitesse dans toutes les horloges. Ainsi, la sémantique d'un automate temporisé est un système de transition temporisé, c'est-à-dire dont les transitions peuvent être étiquetées par une lettre (ou ε) ou par un délai de temps à valeur réelle. Les états de ce système gardent non seulement l'état dans lequel se trouve l'automate mais aussi la valeur des horloges. La sémantique de $\mathscr{A}$ est donc le système infini (non dénombrable) $\mathscr{T}_{\mathscr{A}} = \langle S_{\mathscr{T}}, Lab \uplus \{\varepsilon\} \uplus \mathbb{R}_{\geq 0}, \Delta_{\mathscr{T}}, I_{\mathscr{T}} \rangle$ défini de la façon suivante. Les états de $\mathscr{T}_{\mathscr{A}}$ sont les *configurations* de $\mathscr{A}$: une paire (s,v) où $s \in S$ est un état de l'automate et v une valuation sur X ; donc $S_{\mathscr{T}} = S \times \mathbb{R}_{\geq 0}^X$. La configuration initiale d'un automate temporisé est $(s_I, \mathbf{0})$, c'est-à-dire $I_{\mathscr{T}} = \{(s_I, \mathbf{0})\}$. Les transitions de $\Delta_{\mathscr{T}}$ se divisent en deux types : les *pas de temps* qui correspondent à des délais, et les *pas discrets* qui correspondent au franchissement d'une transition.

Soit (s,v) une configuration de $\mathscr{A}$. Un *pas de temps* de durée d pour $d \in \mathbb{R}_{\geq 0}$ est une transition $(s,v) \xrightarrow{d} (s, v+d) \in \Delta_{\mathscr{T}}$. Si $s \xrightarrow{g,a,u} s' \in \Delta$ et que $v \vDash g$, le *pas discret* correspondant dans $\mathscr{T}_{\mathscr{A}}$ est la transition $(s,v) \xrightarrow{a} (s', v[u]) \in \Delta_{\mathscr{T}}$.

Une exécution de l'automate temporisé $\mathscr{A}$ est un chemin sur $\mathscr{T}_{\mathscr{A}}$ qui alterne pas temporisés (potentiellement de durée nulle) et pas discrets. Comme dans le cas non-temporisé, $Exec(\mathscr{A})$ désigne l'ensemble des exécutions de $\mathscr{A}$ (elles démarrent toutes en $(s_I, \mathbf{0})$). Une exécution est acceptée si elle se termine dans une configuration (s,v) avec $s \in F$. Pour une exécution

$$\rho = (s_0, v_0) \xrightarrow{d_0} (s_0, v_0 + d_0) \xrightarrow{a_0} (s_1, v_1) \cdots (s_n, v_n) \xrightarrow{d_n} (s_n, v_n + d_n) \xrightarrow{a_n} (s_{n+1}, v_{n+1}),$$

le mot temporisé associé est $(a_0, \tau_0) \cdots (a_n, \tau_n)$ où les transitions étiquetées par ε sont enlevées et pour $i \in \{0, \ldots, n\}$, $\tau_i = \sum_{j=0}^{i} d_j$. Ainsi, dans un mot temporisé, à une étiquette de Lab est associée la date à laquelle elle apparaît. Le langage temporisé de $\mathscr{A}$, aussi noté $\mathscr{L}(\mathscr{A})$, est l'ensemble des mots temporisés générés par les exécutions acceptantes de $\mathscr{A}$. L'ensemble des langages acceptés par un automate temporisé est noté TL. On note $Untimed(\mathscr{L}(\mathscr{A}))$ le langage non temporisé de $\mathscr{A}$, qui est la projection sur Lab^* de $\mathscr{L}(\mathscr{A})$. Pour une exécution finie ρ, la *durée* de ρ, notée $Dur(\rho)$, est la somme des durées de tous les pas de temps le long de ρ. La longueur de ρ, notée $|\rho|$, est le nombre de pas discrets franchis le long de ρ.

Par exemple, l'automate temporisé $\mathscr{A}_4$ de la figure 2.2, introduit par [AD94] accepte le mot $(a,1)(b,1.875)(a,2)(b,2.7)(a,3)(b,3.375)(a,4)(b,4.225)$ par l'exécution sui-

vante, où chaque tuple représente $(s, v(z), v(y))$:

$$(q_0,0,0) \xrightarrow{1} (q_0,1,1) \xrightarrow{a} (q_1,0,1) \xrightarrow{0.875} (q_1,0.875,1.875) \xrightarrow{b} (q_2,0.875,0) \xrightarrow{0.125}$$

$$(q_2,1,0.125) \xrightarrow{a} (q_3,0,0.125) \xrightarrow{0.7} (q_3,0.7,0.825) \xrightarrow{b} (q_2,0.7,0) \xrightarrow{0.3} (q_2,1,0.3) \xrightarrow{a}$$

$$(q_3,0,0.3) \xrightarrow{0.375} (q_3,0.375,0.675) \xrightarrow{b} (q_2,0.375,0) \xrightarrow{0.625} (q_2,1,0.625) \xrightarrow{a}$$

$$(q_3,0,0.625) \xrightarrow{0.225} (q_3,0.225,0.85) \xrightarrow{b} (q_2,0.225,0).$$

Cette exécution peut aussi être représentée dans le plan comme sur la figure 2.3

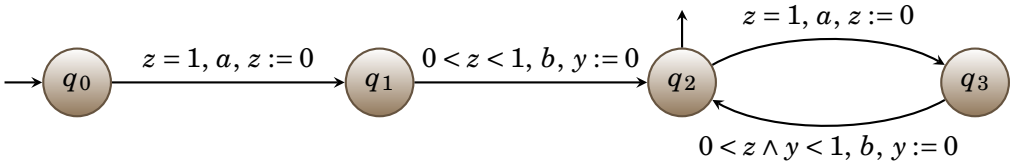


FIGURE 2.2: Un automate temporisé $\mathscr{A}_4$.

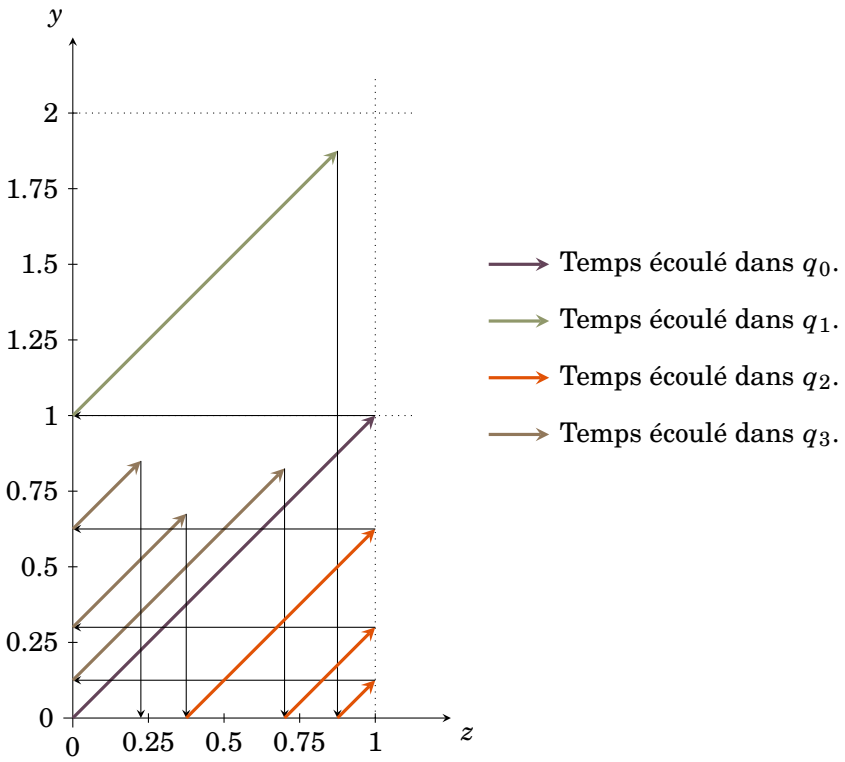


FIGURE 2.3: Valeurs des horloges dans une exécution de $\mathscr{A}_4$.

L'une des principales caractéristiques d'un automate temporisé est qu'il peut être transformé en un automate fini (non temporisé) équivalent du point de vue des langages non temporisés [AD94]. Cette abstraction dite des régions se base sur le fait

qu'il est suffisant de connaître non pas la valeur exacte des horloges, mais leur valeur relativement aux constantes auxquelles elles sont comparées dans les gardes ainsi que les positions relatives de leurs parties fractionnaires.

2.5 Propriétés de sécurité

Non interférence. La non interférence [GM82] est une propriété des systèmes de transitions, étiquetés par un alphabet partitionné $A = H \uplus L$. Le sous alphabet H comporte des actions privées dites de *haut niveau*, qui ne sont observées que par des usagers jouissant de certains privilèges. Le sous alphabet L comporte lui des actions publiques dites de *bas niveau*, qui sont observées par tout le monde.

Un système est non interférent si la présence d'un action de haut niveau ne peut pas être détectée par l'observation des actions de bas niveau. Cette détection revient à comparer le système où les actions de haut niveau sont interdites avec le système où ces mêmes actions sont simplement masquées. Plusieurs notions de non interférence découlent alors de la manière dont ces systèmes sont comparés [FG01] : équivalence de langage, bisimulation faible ou forte, *etc*. Ici c'est l'équivalence de langages qui est utilisée.

Plus formellement, un automate fini $\mathscr{A}$ sur un alphabet $H \uplus L$ est non interférent si pour tout mot $v \in \mathscr{L}(\mathscr{A})$, si v contient (au moins) une lettre de H, alors il existe un mot $w \in \mathscr{L}(\mathscr{A}) \cap L^*$ tel que $\pi_L(v) = w$. Ainsi, l'observation des lettres de L ne permettent pas de détecter la présence d'une lettre de H. Souvent critiquée pour la force du critère qui est violé dès qu'une seule action est détectée [RMMG01], la notion de non interférence peut être élargie pour autoriser un nombre fini de détections de lettres de H.

Soit $\mathscr{A}$ un automate fini sur $A = H \uplus L$ et $k > 0$ un entier. L'automate $\mathscr{A}$ est dit *k-non interférent* si pour tout mot $v \in \mathscr{L}(\mathscr{A})$,

$$v \in \left(L^* \cdot H\right)^k \cdot A^* \Longrightarrow \exists w \in \mathscr{L}(\mathscr{A}) \text{ tel que } w \in \bigcup_{i=0}^{k-1} \left(L^* \cdot H\right)^i \cdot L^*,\ \pi_L(v) = \pi_L(w).$$

La notion classique de non interférence correspond à la 1-non interférence.

Opacité. Un prédicat sur les exécutions d'un système est dit *opaque* si l'observation externe du système ne permet pas de déduire si l'exécution vérifie ce prédicat. L'opacité est un concept général pouvant être instancié, en fixant adéquatement le prédicat et la manière dont le système est observé, en de nombreuses propriétés de sécurité [Maz04, BKMR08].

Pour un système de transition $\mathscr{T}$ et une fonction d'observation (que l'on suppose surjective) $\mathscr{O} : Exec(\mathscr{T}) \to Obs$, un prédicat $\varphi \subseteq Exec(\mathscr{T})$ est *opaque* si pour toute exécution $\rho \in \varphi$, il existe une exécution $\rho \notin \varphi$ telle que $\mathscr{O}(\rho) = \mathscr{O}(\rho')$. Cette définition est équivalente à demander que pour toute observation $o \in Obs$, la *classe d'observation* [2] $\mathscr{O}^{-1}(o)$ n'est pas totalement incluse dans φ, c'est-à-dire $\mathscr{O}^{-1}(o) \not\subseteq \varphi$. L'opacité est définie de façon similaire sur les automates, en ne considérant alors que les exécutions acceptantes.

2. Qui est la classe d'équivalence induite par $\mathscr{O}$ sur $TExec(\mathscr{T})$.

Le prédicat peut être donné sous la forme d'un ensemble φ_0 de suites alternant un état et une étiquette, mais ne correspondant pas nécessairement à des exécutions sur $\mathcal{T}$. Dans ce cas le prédicat considéré est implicitement $\varphi = \varphi_0 \cap TExec(\mathcal{T})$.

L'opacité permet, par exemple, d'exprimer la k-non interférence : un automate est k-non interférent si le prédicat $tr^{-1}\left((L^* \cdot H)^k \cdot (H \uplus L)^*\right)$ (les exécutions dont la trace contient au moins k lettres de H) est opaque pour la fonction d'observation $\pi_L \circ tr$ (qui observe seulement les lettres de L).

Dans ce cas le prédicat et la fonction d'observation portent sur les lettres lues, mais en général ils peuvent être définis par les états traversés durant l'exécution. Cette grande latitude dans la définition des fonctions d'observations et des prédicats rend le problème de décision de l'opacité indécidable.

Modélisation de canaux par transducteurs rationnels

Chapitre 3

Il n'y aura bientôt plus d'erreurs. Un secret.
Une machine perfectionnée. Vous verrez.

Albert Camus, *L'état de siège*, 1948.

Dans ce chapitre nous étudions une modélisation des systèmes par des *transducteurs rationnels*. Ainsi, un attaquant cherche à utiliser les liens entre ce que le système prend en entrée et ce qu'il produit pour établir une communication fiable, c'est-à-dire sans erreurs.

Par ailleurs, dans ce modèle, l'attaquant a la même puissance de calcul que le système lui même : il ne peut utiliser, pour coder et décoder le message, que des transducteurs rationnels. Ce modèle exclut donc de transmettre via une représentation unaire un message initialement binaire.

Dans ce cadre, nous montrons tout d'abord que le critère du canal *parfait* n'est pas aussi restrictif qu'à première vue, puisqu'il est équivalent au cas d'un nombre borné d'erreurs.

Puis nous nous intéressons au problème de synthèse d'un tel canal, dégageant une condition nécessaire pour l'obtention d'un canal : l'existence d'un *état encodant*. L'existence d'un tel état est une propriété structurelle du transducteur qui peut être décidée en temps polynomial. Cette condition n'est pas suffisante en général (nous montrons d'ailleurs que le problème de synthèse est indécidable dans le cas général) mais le devient dans le cas des transducteurs fonctionnels.

Nous discutons ensuite des liens entre ce modèle de transmission d'information et la propriété de non interférence.

3.1 Notations

On donne ici quelques notations utiles à ce chapitre

Préfixes et substitutions. Pour un entier naturel k, l'ensemble des *préfixes k-bornés* de w est $Pref_k(w) = \{v \in A^* \mid v \preceq w \wedge |w| - |v| \leq k\}$ qui contient les préfixes de w dont la longueur diffère de celle de w d'au plus k lettres. Cette notation est étendue aux ensembles de mots :

$$\text{si } L \subseteq A^*, \quad Pref_k(L) = \left\{ w \in A^* \,\middle|\, \exists v \in L, w \in Pref_k(v) \right\}.$$

La relation d'identité sur A est élargie par $Id_k(A) = \{(v,w) \mid w \in Pref_k(v)\}$ entre un mot et ses préfixes k-bornés, avec $Id_0 = Id$.

La *distance de substitution* entre deux mots v et w de même longueur n est le nombre de lettres par lesquelles ils diffèrent :

$$d_s(v,w) = |\{i \mid 1 \leq i \leq n, v[i] \neq w[i]\}|.$$

Pour deux entiers m,p avec $p \leq m \leq n$, les mots v et w sont (m,p)-proches si pour tout facteur de taille au plus m dans v, le facteur correspondant dans w est à distance moins de p (pour la distance de substitution) :

$$\forall m' < m,\ \forall 1 \leq i \leq n-m',\ d_s\big(v[i]\cdots v[i+m'], w[i]\cdots w[i+m']\big) \leq p.$$

On remarque que si v et w sont $(0,m)$-proches pour un certain $m > 0$, alors $v = w$. Pour un mot v, on note $Pro_p^m(v)$ l'ensemble des mots (m,p)-proches de v. Cette notation est étendue aux ensembles de mots :

$$\text{si } L \subseteq A^*, \quad Pro_p^m(L) = \left\{(v,w) \in L \times A^* \;\middle|\; w \in Pro_p^m(v)\right\}.$$

Codes. Du point de vue de la transmission d'information, un code est un ensemble de mots avec lequel on peut transmettre de l'information de manière non-ambiguë. C'est ainsi que nous allons les utiliser dans ce chapitre.

On dit qu'un ensemble fini $B = \{w_1, \ldots, w_n\}$ de mots de A^* forme un *code* si tout mot w de B^* peut être décomposé d'une unique façon en mots de B :

$$w = w_{i_1} \cdots w_{i_m} \quad \text{avec} \quad i_j \in \{1, \ldots, n\} \quad \text{pour} \quad 1 \leq j \leq m.$$

Les codes à deux mots ont des propriétés combinatoires qui font partie du « folklore » de la théorie des langages [Har78, Lot83] : l'ensemble $\{w_0, w_1\}$ ne forme pas un code si et seulement si $w_0 \cdot w_1 = w_1 \cdot w_0$ (on dit alors que w_0 et w_1 *commutent*) si et seulement si il existe un mot w tel que $w_0, w_1 \in w^*$.

On peut également démontrer que pour tout mot $w \neq \varepsilon$, l'ensemble $C(w)$ des mots qui commutent avec w est de la forme v^*, où v est le plus petit mot non vide qui commute avec w.

3.2 Canaux parfaits dans les transducteurs

On cherche ici à modéliser des canaux qui transmettent *parfaitement* l'information. Le cas de la communication parfaite permet de modéliser l'existence de moyens de communications très fiables, tels que ceux nécessaires, par exemple, à la transmission d'une clef secrète. On considère que l'information que l'on cherche à transmettre est sous forme binaire, ce qui est le cas de toutes les données utilisées dans les systèmes informatiques, si besoin au travers d'un codage idoine.

Dans ce modèle, un émetteur, sous la forme d'un transducteur que l'on appelle *encodeur*, cherche à transmettre au travers du système de l'information secrète (voir figure 3.1). Pour ce faire, il envoie au système des actions dites de haut niveau (sur un alphabet H). Le système réagit en produisant des actions de bas niveau (sur un alphabet L). Ces actions de bas niveau, publiques, sont ensuite lues et décodées par le récepteur, qui est pour cette raison aussi appelé *décodeur*.

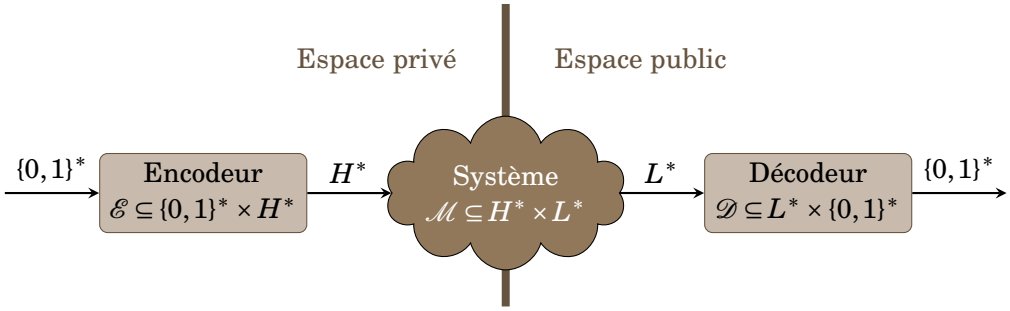


FIGURE 3.1: Canal réalisé par transducteurs.

3.2.1 Erreurs de transmission

Bien que l'on cherche à obtenir, donc à détecter, un canal parfait, on autorise certains défauts dans la communication. Ceux-ci rendent compte d'un modèle plus réaliste tout en se ramenant, comme nous allons le montrer, au cas d'un canal parfait.

Retards

Un *retard* borné est acceptable dans la communication : par exemple, pour détecter les canaux même lorsque la communication peut s'interrompre en cours de transmission. On définit alors un *canal* par l'existence d'un moyen d'encoder et de décoder un message de sorte à ce que l'on décode exactement ce qui a été encodé, ou du moins un préfixe « pas trop court ».

Définition 3.1 (Canal de retard k)**.** *Soit $k \in \mathbb{N}$. Une relation $\mathscr{M}$ de H^* vers L^* contient un* canal de retard k *s'il existe deux relations rationnelles $\mathscr{E}$ de $\{0,1\}^*$ dans H^* et $\mathscr{D}$ de L^* dans $\{0,1\}^*$ telles que*

$$Id\left(\{0,1\}^*\right) \subseteq \mathscr{E} \cdot \mathscr{M} \cdot \mathscr{D} \subseteq Id_k\left(\{0,1\}^*\right).$$

Cette notion de retard peut en réalité être éliminée des données du problème. En effet, un canal de retard $k > 0$ peut toujours être transformé en un canal de retard 0. Il suffit pour cela que l'encodeur transmette chaque bit $k+1$ fois, et que le décodeur ne garde chaque fois que le premier bit sur les $k+1$ qu'il décode.

À cet égard, on définit la famille de morphismes Rep_k de $\{0,1\}^*$ dans $\{0,1\}^*$ par $Rep_k(0) = 0^k$ et $Rep_k(1) = 1^k$. Vu comme une relation, Rep_k est rationnel : le transducteur a un seul état q (initial et final) ayant pour transitions $q \xrightarrow{0|0^k} q$ et $q \xrightarrow{1|1^k} q$ reconnaît Rep_k.

On définit de même la famille $Derep_k$ de $\{0,1\}^*$ dans $\{0,1\}^*$ qui a l'effet inverse de Rep_k :

$$\text{Pour } x \in \{0,1\}, \begin{cases} Derep_k(x^k \cdot w) = x \cdot Derep_k(w) \\ Derep_k(x^j) = x, \text{ pour } j \leq k \end{cases}$$

On prouve alors aisément :

Lemme 3.1 (Élimination des retards)**.**

Si $Id\left(\{0,1\}^*\right) \subseteq \mathscr{E} \cdot \mathscr{M} \cdot \mathscr{D} \subseteq Id_k\left(\{0,1\}^*\right)$, alors $Rep_{k+1} \cdot \mathscr{E} \cdot \mathscr{M} \cdot \mathscr{D} \cdot Derep_{k+1} = Id\left(\{0,1\}^*\right)$.

Substitutions

De même, on peut admettre qu'un « petit » nombre de substitutions se produise durant la transmission. En effet, si l'on considère qu'au moins la moitié des bits est transmise, et que les erreurs ne sont pas trop agglutinées, la communication va quand même pouvoir s'effectuer. Cette définition correspond au modèle de [KS10] dans le cas où seules les substitutions sont autorisées.

Définition 3.2 (Canal à (m,p)-substitutions)**.** *Soit $m,p \in \mathbb{N}$. Une relation $\mathscr{M}$ de H^* vers L^* contient un* canal à (m,p)-substitutions *s'il existe deux relations rationnelles $\mathscr{E}$ de $\{0,1\}^*$ dans H^* et $\mathscr{D}$ de L^* dans $\{0,1\}^*$ telles que*

$$Id\left(\{0,1\}^*\right) \subseteq \mathscr{E} \cdot \mathscr{M} \cdot \mathscr{D} \subseteq Pro_p^m\left(\{0,1\}^*\right).$$

De la même façon que les retards ont été éliminés par transmission redondante, on peut obtenir un canal sans erreur, en modifiant un canal où les erreurs ne sont pas trop importantes. On définit à cet effet la relation $Vote_m$ qui transforme une suite de m bits en la valeur majoritairement lue, réalisée par le transducteur $\mathscr{A}_m$ de la figure 3.2. Formellement, les états de $\mathscr{A}_m$ sont les paires (i,j) pour des entiers natu-

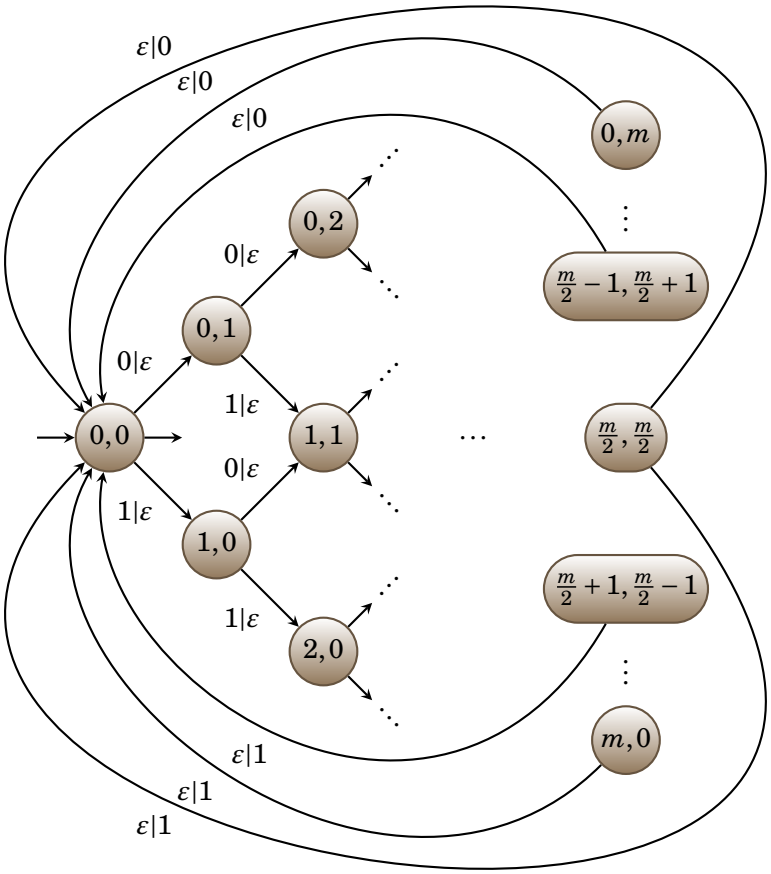


FIGURE 3.2: Transducteur $\mathscr{A}_m$ qui réalise la relation $Vote_m$. On a ici supposé que m était pair, le cas m impair étant similaire.

rels i et j tels que $i+j \leq m$. Le seul état initial et final est $(0,0)$. Les transitions sont construites par les trois règles suivantes :

- Si $i+j<m$, alors $\mathscr{A}_m$ contient les transitions $(i,j) \xrightarrow{0|\varepsilon} (i+1,j)$ et $(i,j) \xrightarrow{1|\varepsilon} (i,j+1)$.
- Si $i+j=m$ et que $i \geq j$ alors $\mathscr{A}_m$ contient la transition $(i,j) \xrightarrow{\varepsilon|0} (0,0)$.
- Si $i+j=m$ et que $j \geq i$ alors $\mathscr{A}_m$ contient la transition $(i,j) \xrightarrow{\varepsilon|1} (0,0)$.

Lemme 3.2 (Élimination des substitutions)**.** *Soient m et p deux entiers tels que $m > 2p$.*

$$\textit{Si } Id\left(\{0,1\}^*\right) \subseteq \mathscr{E}\cdot\mathscr{M}\cdot\mathscr{D} \subseteq Pro_m^p\left(\{0,1\}^*\right), \textit{ alors } Rep_m\cdot\mathscr{E}\cdot\mathscr{M}\cdot\mathscr{D}\cdot Vote_m = Id\left(\{0,1\}^*\right).$$

Démonstration. Soit $\beta \in \{0,1\}^*$ de longueur n. Par $Rep_m\cdot\mathscr{E}\cdot\mathscr{M}\cdot\mathscr{D}$, l'image de β est un ensemble de mots (m,p)-proches de $v = \beta[1]^m\cdots\beta[n]^m$. Soit w un tel mot. Comme $w \in Pro_p^m(v)$, w a la même longueur que v, à savoir nm. On peut donc l'écrire

$$w = \overbrace{b_{1,1}\cdots b_{1,m}}^{w_1}\cdot\overbrace{b_{2,1}\cdots b_{2,m}}^{w_2}\cdots\overbrace{b_{n,1}\cdots b_{n,m}}^{w_n}.$$

De plus pour $1 \leq i \leq n$, chaque facteur w_i diffère de $\beta[i]^m$ d'au plus p lettres. Donc dans w_i il y a au plus p bits de valeur $\overline{\beta[i]}$, et au moins $m-p$ bits de valeur $\beta[i]$. Comme $m-p>p$, l'image de w_i par $Vote_m$ est $\beta[i]$. Ainsi l'image de w par $Vote_m$ est β, et $Rep_m\cdot\mathscr{E}\cdot\mathscr{M}\cdot\mathscr{D}\cdot Vote_m = Id(\{0,1\}^*)$. □

On peut éliminer aisément soit les retards soit les erreurs, au vu des deux propriétés précédentes. Notons que l'on peut éliminer les deux à la fois, pourvu, là encore, qu'il n'y ait pas trop d'erreurs.

Pour deux entiers $m>k$ la relation $Vote'_{m,k}$ est similaire à la relation $Vote_m$ mais autorise cependant à arrêter totalement la lecture en entrée en sortant le résultat courant du vote, lorsque au moins $m-k$ lettres ont été lues (au sein du groupe de m lettres courant). Les états du transducteur $\mathscr{A}_{m,k}$ réalisant $Vote'_{m,k}$ sont ceux de $\mathscr{A}_m$ auxquels on ajoute un état end, final. De même, les transitions sont celles des $\mathscr{A}_{m,k}$ et dans $\mathscr{A}_m$, auxquelles il sont ajoutées les transitions construites par les règles suivantes :

- Si $0 \leq m-(i+j) \leq k$ et $i \geq j$, alors $\mathscr{A}'_{m,k}$ contient la transition $(i,j) \xrightarrow{\varepsilon|0} end$.
- Si $0 \leq m-(i+j) \leq k$ et $j \geq i$, alors $\mathscr{A}'_{m,k}$ contient la transition $(i,j) \xrightarrow{\varepsilon|1} end$.

Remarquons que depuis l'état end, le transducteur $\mathscr{A}'_{m,k}$ ne lit ni n'écrit plus rien. En particulier, pour qu'une paire de mots soit acceptée par une exécution de $\mathscr{A}_{m,k}$, il faut que tout le mot d'entrée ait été lu avant d'entrer dans end.

Lemme 3.3. *Soit k, m et p trois entiers tels que $m > 2(k+p)$.*

$$\textit{Si} \quad Id\left(\{0,1\}^*\right) \subseteq \mathscr{E}\cdot\mathscr{M}\cdot\mathscr{D} \subseteq \left\{(v,w) \,\middle|\, v\in\{0,1\}^*, w \in Pref_k\left(Pro_p^m(v)\right)\right\}$$

$$\textit{alors} \quad Rep_m\cdot\mathscr{E}\cdot\mathscr{M}\cdot\mathscr{D}\cdot Vote'_{m,k} = Id\left(\{0,1\}^*\right).$$

Démonstration. La relation $Vote'_{m,k}$ traite les éventuels bits manquant dans le mot comme des erreurs. Comme $m-(k+p) > k+p$, le résultat du « vote » est toujours la valeur souhaitée, et le mot est totalement transmis correctement. □

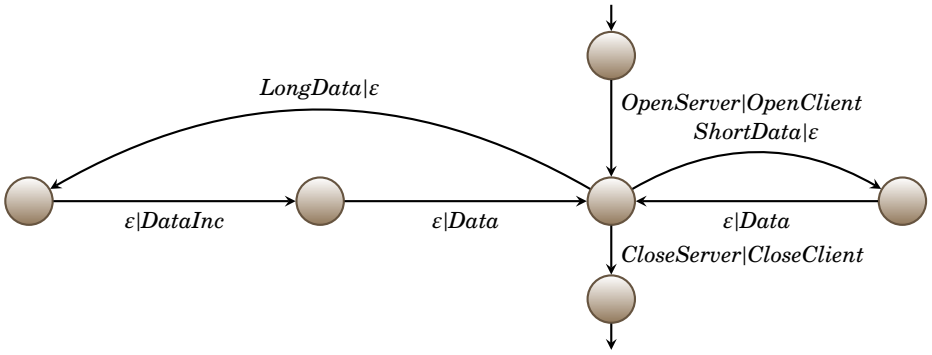


FIGURE 3.3: Transducteur $\mathscr{A}_{\text{pack}}$ modélisant un système de transmission de données. $\mathscr{A}_{\text{pack}}$ réalise la relation $\mathscr{M}_{\text{pack}}$.

Canaux parfaits

Puisque l'on peut éliminer les retards bornés et certaines erreurs, on adopte la définition suivante :

Définition 3.3 (Canal)**.** *Soit $\mathscr{E}$ une relation rationnelle de $\{0,1\}^*$ dans H^*, $\mathscr{M}$ une relation rationnelle de H^* vers L^* et $\mathscr{D}$ une relation rationnelle de L^* dans $\{0,1\}^*$. La paire $(\mathscr{E},\mathscr{D})$ est un* canal *pour $\mathscr{M}$ si*

$$\mathscr{E}\cdot\mathscr{M}\cdot\mathscr{D} = Id\left(\{0,1\}^*\right).$$

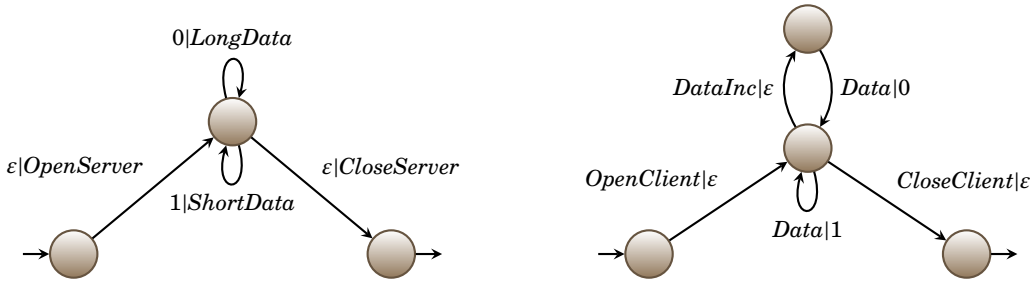
La relation $\mathscr{M}$ contient un canal si une telle paire $(\mathscr{E},\mathscr{D})$ existe.

Exemple 3.1. On considère le système de transmission de paquets de la figure 3.3, inspiré de [HZD05]. Le système peut transmettre de l'information de deux façons. Lorsqu'il reçoit une faible quantité de données (*ShortData*), il le transmet en un seul paquet (*Data*). Mais lorsqu'il reçoit une grande quantité de données (*LongData*), il doit les transmettre sous forme d'un paquet « incomplet » (*DataInc*), suivi d'un paquet normal (*Data*). Un paquet incomplet indique que d'autres paquets doivent suivre. Les actions *OpenServer* et *OpenClient* (resp. *CloseServer* et *CloseClient*) servent à l'ouverture (resp. la fermeture) de la session de transfert.

Un utilisateur peut tirer profit de ce fonctionnement pour transmettre de l'information, non pas *au sein* des données transmises, mais dans la *manière* dont celles-ci sont transmises. En effet, il lui suffit d'initier une session et d'envoyer une grande quantité de données (arbitraires) pour transmettre un 0 et une petite quantité de données pour transmettre un 1. Le décodeur peut alors décoder un paquet incomplet suivi d'un paquet normal comme un 0 et un paquet normal seul comme un 1, là encore sans se soucier du contenu de ces paquets. Ces relations de codage $\mathscr{E}_{\text{pack}}$ et de décodage $\mathscr{D}_{\text{pack}}$ sont rationnelles et réalisées par les transducteurs de la figure 3.4.

3.2.2 Vérification de la propriété de canal

Tout d'abord, on cherche à vérifier si, étant donné une paire $(\mathscr{E},\mathscr{D})$, celle-ci est effectivement un canal. Cela revient à vérifier que la composition $\mathscr{E}\cdot\mathscr{M}\cdot\mathscr{D}$ est bien l'identité.



(a) Encodeur réalisant $\mathscr{E}_{\text{pack}}$. (b) Décodeur réalisant $\mathscr{D}_{\text{pack}}$.

FIGURE 3.4: Canal sur $\mathscr{M}_{\text{pack}}$.

Rappelons que vérifier l'égalité de deux relations rationnelles est un problème indécidable en général. Cependant, l'égalité de deux relations *fonctionnelles* est décidable [Sak03]. D'autre part, la relation identité est fonctionnelle. Enfin, on peut décider si une relation est fonctionnelle [Sch75, BCPS00]. La procédure décide donc tout d'abord si $\mathscr{E} \cdot \mathscr{M} \cdot \mathscr{D}$ est fonctionnelle. Si $\mathscr{E} \cdot \mathscr{M} \cdot \mathscr{D}$ n'est pas fonctionnelle, elle ne peut pas être l'identité, et $(\mathscr{E}, \mathscr{D})$ n'est pas un canal. Si $\mathscr{E} \cdot \mathscr{M} \cdot \mathscr{D}$ est fonctionnelle, on décide alors l'égalité avec $Id(\{0,1\}^*)$, et donc si $(\mathscr{E}, \mathscr{D})$ est un canal.

Théorème 3.4 (Vérification de la propriété de canal)**.** *Soit $\mathscr{M}$ une relation rationnelle de H^* vers L^*, $\mathscr{E}$ une relation rationnelle de $\{0,1\}^*$ vers H^*, et $\mathscr{D}$ une relation rationnelle de L^* vers $\{0,1\}^*$. On peut décider si $(\mathscr{E}, \mathscr{D})$ est un canal pour $\mathscr{M}$.*

3.3 Le problème de synthèse

Le problème qui se pose en pratique est celui de la *détection* d'un canal. C'est-à-dire, étant donné un transducteur, la relation qu'il réalise contient-elle un canal ?

Définition 3.4 (Problème d'existence d'un canal)**.** *Étant donné une relation rationnelle $\mathscr{M}$ de H^* vers L^*, existe-t-il des relations $\mathscr{E}$ de $\{0,1\}^*$ vers H^* et $\mathscr{D}$ de L^* vers $\{0,1\}^*$ telles que $(\mathscr{E}, \mathscr{D})$ est un canal pour $\mathscr{M}$.*

3.3.1 Une condition nécessaire

Commençons par une condition nécessaire de l'existence d'un canal. Cette condition correspond à une propriété structurelle du transducteur $\mathscr{A}$ qui réalise $\mathscr{M}$.

Plus précisément, un canal n'est possible que si pour un certain état dit *encodant*, on trouve un code sur H qui est transmis de manière non ambiguë par $\mathscr{A}$ en un code sur L.

Définition 3.5 (État encodant)**.** *Soit $\mathscr{A} = \langle S, H^* \times L^*, I, \Delta, F \rangle$ un transducteur. Un état encodant est un état $s \in S$ pour lequel il existe quatre mots $v_0, v_1 \in H^*$ et $w_0, w_1 \in L^*$ tels que*

(i) *(v_0, w_0) et (v_1, w_1) étiquettent un cycle sur s, c'est-à-dire $s \xrightarrow{v_0|w_0} s$ et $s \xrightarrow{v_1|w_1} s$;*
(ii) *$\{v_0, v_1\}$ et $\{w_0, w_1\}$ forment des codes respectivement sur H^* et L^*, c'est-à-dire $v_1 \cdot v_0 \neq v_0 \cdot v_1$ et $w_1 \cdot w_0 \neq w_0 \cdot w_1$.*

Dans la suite, on prouve le théorème suivant :

Théorème 3.5 (Condition nécessaire d'existence d'un canal)**.** *Soit $\mathscr{M}$ une relation rationnelle réalisée par un transducteur $\mathscr{A}$. Si $\mathscr{M}$ a un canal, alors $\mathscr{A}$ a un état encodant.*

La définition d'état encodant apparaît sous une forme similaire dans [HZD05], cependant, l'existence d'un état encodant y est pris comme définition de canal, et non comme conséquence de celle-ci. Or l'existence d'un état encodant n'est pas une condition suffisante à l'existence d'un canal tel que défini ici. En effet, le transducteur $\mathscr{A}_{\overline{\text{cc}}}$ de $\{h,h_0,h_1\}^*$ vers $\{\ell,\ell_0,\ell_1\}^*$ de la figure 3.5 a bien des états encodants (ici s_1 et s_2), mais pas de canal. Plus précisément, en notant $\mathscr{M}_{\overline{\text{cc}}}$ la relation réalisée par $\mathscr{A}_{\overline{\text{cc}}}$:

$$\forall n \in \mathbb{N},\ \forall 1 \leq j \leq n,\ i_j \in \{0,1\} \quad \mathscr{M}_{\overline{\text{cc}}}\left(h \cdot h_{i_1} \cdots h_{i_n}\right) = \ell \cdot (\ell_0 + \ell_1)^n.$$

Ainsi, quelque soit les relations rationnelles $\mathscr{E}$ et $\mathscr{D}$, ce que lit $\mathscr{D}$ a pour antécédant par $\mathscr{M}_{\overline{\text{cc}}}$ tous les mots de même longueur. Donc $\mathscr{E}$ ne transmet à $\mathscr{D}$ que la longueur n de ce qu'il émet. Or, il n'existe pas de transducteur qui transforme un nombre binaire en unaire. Ainsi il est impossible de transmettre tout $\{0,1\}^*$, et donc $(\mathscr{E},\mathscr{D})$ ne peut pas être un canal.

L'existence d'un état encodant découle de propriétés combinatoires des automates et des transducteurs. Tout d'abord, on montre que si un canal existe, il peut être réalisé par des relations canoniques (théorème 3.7). On prouve ensuite que ce canal canonique traduit l'existence d'un état encodant dans le transducteur (lemme 3.9). Les preuves se basent sur des lemmes techniques, et en particulier le lemme de comptage suivant :

Lemme 3.6 (Existence des boucles codantes)**.** *Soit $\mathscr{A} = \langle S, Lab, \Delta, I, F\rangle$ un automate où $\langle Lab, \cdot, \varepsilon\rangle$ est un monoïde simplifiable (mais pas nécessairement libre). Si a, a_0, a_1 et a' sont des mots de Lab tels que $\{a_0,a_1\}$ forme un code et $a \cdot \{a_0,a_1\}^* \cdot a' \subseteq \mathscr{L}(\mathscr{A})$. Alors il existe*

- *des états $s_0 \in I$, $s \in S$, et $s_f \in F$;*
- *des mots $w \in a \cdot \{a_0,a_1\}^*$, $w' \in \{a_0,a_1\}^* \cdot a'$, $w_0 \in \{a_0\}^*$ et $w_1 \in \{a_0,a_1\}^* \cdot a_1 \cdot \{a_0,a_1\}^*$ tels que*

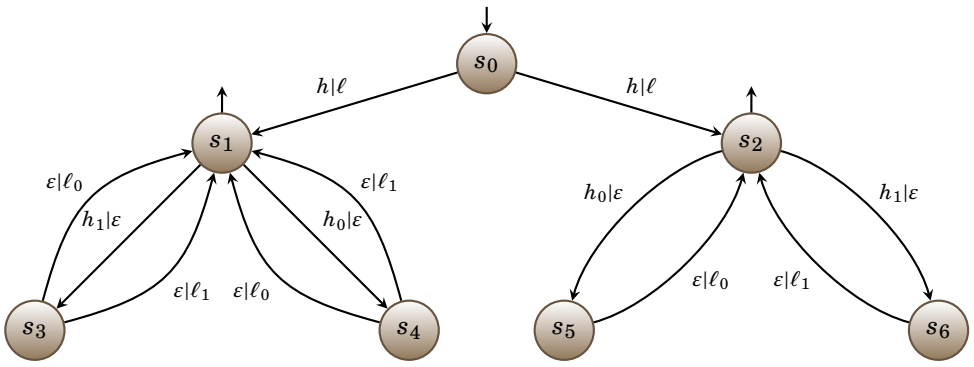


FIGURE 3.5: Transducteur $\mathscr{A}_{\overline{\text{cc}}}$ dont les états s_1 et s_2 sont encodants mais qui ne contient pas de canal.

– $\{w_0, w_1\}$ *forme un code sur* Lab*;*
– *il existe dans* $\mathscr{A}$ *des chemins* $s_0 \overset{w}{\Rightarrow} s$, $s \overset{w_0}{\Longrightarrow} s$, $s \overset{w_1}{\Longrightarrow} s$ *et* $s \overset{w'}{\Longrightarrow} s_f$.

Les *boucles codantes* détectées dans ce lemme sont une version plus faible des états encodants.

Démonstration. Soit $p = |S|$ le nombre d'états de $\mathscr{A}$. Un *cycle simple* dans $\mathscr{A}$ désigne une suite de transitions $s_1 \xrightarrow{a_1} s_2 \cdots s_n \xrightarrow{a_n} s_1$ où pour $1 \leq i, j \leq n$, $s_i = s_j \Rightarrow i = j$. Il y a un nombre fini de cycles simples dans $\mathscr{A}$; plus précisément il y en a au plus $(p+1)! \cdot |\Delta|^p$. En effet, l'ensemble des états traversés par un cycle est un sous ensemble de S muni d'un ordre. On peut représenter un tel cycle $s_1 \to s_2 \to \cdots \to s_n \to s_1$ par

$$s_1 < s_2 < \cdots < s_n < \# < s'_1 < s'_{p-n} \quad \text{où \# est un nouveau symbole}$$

et les s'_i sont les états de S n'apparaissant pas dans le cycle. Il y a donc au plus $(p+1)!$ cycles. D'autre part, entre deux états il y a au plus $|\Delta|$ transitions et un cycle est au plus de longueur p. Il y a donc au plus $|\Delta|^p$ choix possibles pour les transitions le long d'un cycle.

Soit $m > p$. Chaque chemin étiqueté par a_0^m contient au moins un cycle simple dont la trace est dans a_0^+. Un tel cycle est dénommé un a_0-cycle dans la suite. Il existe dans $\mathscr{A}$ un nombre $k \leq (p+1)! \cdot |\Delta|^p$ de a_0-cycles.

Soit $u = a_0^{p+1} \cdot a_1$ et $v = a \cdot u^{k+1} \cdot a'$. Remarquons que $v \in a \cdot \{a_0, a_1\}^* \cdot a'$, donc $v \in \mathscr{L}(\mathscr{A})$. Ainsi considérons une exécution ρ qui reconnaît v :

$$\rho = s_0 \overset{a}{\Rightarrow} s'_0 \overset{a_0^{p+1}}{\Longrightarrow} s_1 \overset{a_1}{\Rightarrow} s'_1 \cdots \overset{a_0^{p+1}}{\Longrightarrow} s_{k+1} \overset{a_1}{\Rightarrow} s'_{k+1} \overset{a'}{\Rightarrow} s_f$$

avec $s_0 \in I$ et $s_f \in F$. On note, pour $0 \leq i \leq k$ par ρ_j la sous-exécution de ρ qui reconnaît le $i+1$-ième a_0^{p+1}. Ainsi, $\rho_i = s'_i \overset{a_0^{p+1}}{\Longrightarrow} s_{i+1}$. Par pompage, on sait que chaque ρ_i contient un a_0-cycle. On obtient donc $k+1$ a_0-cycles, qui ne peuvent être tous distincts. Soit $s \overset{w_0}{\Longrightarrow} s$ un a_0-cycle répété dans ρ, dans les sous exécutions ρ_j et $\rho_{j'}$. On peut donc réécrire ρ de la façon suivante :

$$\rho = s_0 \overset{w}{\Rightarrow} s \overset{w_0}{\Longrightarrow} s \overset{w_1}{\Longrightarrow} s \overset{w_0}{\Longrightarrow} s \overset{w'}{\Longrightarrow} s_f.$$

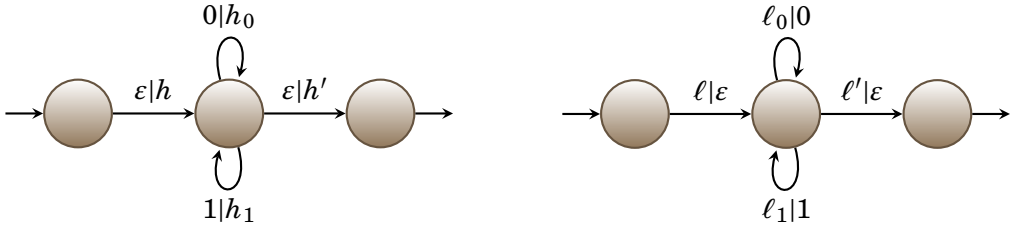
Puisque les deux instances du cycle $s \overset{w_0}{\Longrightarrow} s$ sont respectivement dans ρ_j et $\rho_{j'}$, qui sont séparées par au moins un a_1, le mot w_1 contient au moins un a_1. Comme w_0 est composé uniquement de a_0 et que Lab est simplifiable, w_0 et w_1 ne commutent pas. Toutes les conditions du lemme sont donc réunies pour les états s_0, s, s_f et les mots w, w_0, w_1 et w'. □

Ce lemme de comptage sur les transducteurs nous permet de déduire le résultat suivant, qui relie l'existence d'un canal à celle d'un canal de forme bien particulière, qui sera qualifié de *canonique*. Le canal de l'exemple 3.1 est d'ailleurs un canal sous cette forme.

Théorème 3.7 (Canal canonique)**.** *Si* $\mathscr{M}$ *a un canal, alors* $\mathscr{M}$ *a un canal* $(\mathscr{E}_C, \mathscr{D}_C)$ *où*

$$\begin{aligned} \mathscr{E}_C &= (\varepsilon, h) \cdot \{(0, h_0), (1, h_1)\}^* \cdot (\varepsilon, h') \\ \mathscr{D}_C &= (\ell, \varepsilon) \cdot \{(\ell_0, 0), (\ell_1, 1)\}^* \cdot (\ell', \varepsilon) \end{aligned}$$

avec $h, h_0, h_1, h' \in H^*$ *et* $\ell, \ell_0, \ell_1, \ell' \in L^*$ *tels que* $h_1 \cdot h_0 \neq h_0 \cdot h_1$ *et* $\ell_1 \cdot \ell_0 \neq \ell_0 \cdot \ell_1$*. Ces relations, paramétrées respectivement par les quadruplets de mots* (h, h_0, h_1, h') *et* $(\ell, \ell_0, \ell_1, \ell')$*, sont réalisées par les transducteurs de la figure 3.6.*



(a) Transducteur réalisant $\mathscr{E}_{\mathrm{C}}(h,h_0,h_1,h')$. (b) Transducteur réalisant $\mathscr{D}_{\mathrm{C}}(\ell,\ell_0,\ell_1,\ell')$.

FIGURE 3.6: Encodeur et décodeur canoniques.

Avant de prouver ce théorème, on démontre un lemme sur les encodeurs canoniques :

Lemme 3.8 (Composition d'encodeurs canoniques)**.** *Soit (u,u_0,u_1,u') un quadruplet de mots de $\{0,1\}^*$ tels que $\{u_0,u_1\}$ forme un code sur $\{0,1\}^*$. Soit A un alphabet fini et (v,v_0,v_1,v') un quadruplet de mots de A^* tels que $\{v_0,v_1\}$ forme un code sur A^*. Alors il existe un quadruplet (w,w_0,w_1,w') de mots de A^* tels que*

(i) $\{w_0,w_1\}$ forme un code sur A^*,
(ii) $\mathscr{E}_{\mathrm{C}}(w,w_0,w_1,w') = \mathscr{E}_{\mathrm{C}}(u,u_0,u_1,u') \cdot \mathscr{E}_{\mathrm{C}}(v,v_0,v_1,v')$.

Démonstration. Posons $\mathscr{E}_u = \mathscr{E}_{\mathrm{C}}(u,u_0,u_1,u')$, $\mathscr{E}_v = \mathscr{E}_{\mathrm{C}}(v,v_0,v_1,v')$ et $\mathscr{E}_{uv} = \mathscr{E}_u \cdot \mathscr{E}_v$. Pour encoder un mot binaire par $\mathscr{E}_{uv}$, $\mathscr{E}_u$ doit d'abord envoyer le préfixe u, une suite de u_0 et u_1 (selon le message), puis u', le tout constituant un mot (binaire) σ. Puis, pour transmettre σ, $\mathscr{E}_v$ envoie v, puis une séquence de v_0 et v_1, et enfin v'. Cependant chaque instance de u_0 (resp. de u_1) s'encode pareillement en une séquence de v_0 et v_1. Posons $|u| = k$, $|u_0| = n$, $|u_1| = m$ et $|u'| = p$. En assimilant un élément de $\{0,1\}$ à l'indice qu'il représente, on définit les mots w, w_0, w_1, w' par

$$w = v \cdot v_{u[1]} \cdots v_{u[k]}, \quad w_0 = v_{u_0[1]} \cdots v_{u_0[n]}, \quad w_1 = v_{u_1[1]} \cdots v_{u_1[m]}, \quad w' = v_{u'[1]} \cdots v_{u'[p]} \cdot v'$$

et l'on a aisément $\mathscr{E}_{\mathrm{C}}(w,w_0,w_1,w') = \mathscr{E}_{uv}$.

Il faut cependant s'assurer que $\{w_0,w_1\}$ forme un code. Supposons, par contradiction, que w_0 et w_1 commutent. On a donc

$$v_{u_0[1]} \cdots v_{u_0[n]} \cdot v_{u_1[1]} \cdots v_{u_1[m]} = v_{u_1[1]} \cdots v_{u_1[m]} \cdot v_{u_0[1]} \cdots v_{u_0[n]}.$$

Or $\{v_0,v_1\}$ forme un code, donc la seule manière d'obtenir cette égalité est d'avoir égalité des suites d'indices, c'est-à-dire

$$u_0[1] \cdots u_0[n] \cdot u_1[1] \cdots u_1[m] = u_1[1] \cdots u_1[m] \cdot u_0[1] \cdots u_0[n]$$

ou encore $u_0 \cdot u_1 = u_1 \cdot u_0$, ce qui est en contradiction avec le fait que $\{u_0,u_1\}$ forme un code. □

Démonstration du théorème 3.7. L'idée générale de la preuve réside dans la détection de boucles codantes, tout d'abord dans l'encodeur, puis dans le transducteur réalisant la relation composée encodeur-système. Dans le premier cas, cela permet de trouver un encodeur restreint à cet état, et dans le second cas, cela donne les mots utilisés par un décodeur canonique pour décoder le message, faisant fi des entrées ne correspondant pas au motif attendu. À chaque étape, transformer l'encodeur oblige à transformer le décodeur en conséquence, et *vice versa*.

Des boucles codantes dans l'encodeur. Soit $(\mathscr{E},\mathscr{D})$ un canal pour $\mathscr{M}$. Sans perte de généralité, on peut considérer que l'image de $\mathscr{E}$ est incluse dans le domaine de $\mathscr{M}\cdot\mathscr{D}$. En effet, en prenant $\mathscr{E}' = \mathscr{E}\cdot Id(Dom(\mathscr{M}\cdot\mathscr{D}))$, on obtient un encodeur ayant cette propriété. On note $\mathscr{A}_{\mathscr{E}} = \langle S, \{0,1\}^* \times H^*, I, \Delta, F\rangle$ le transducteur qui réalise $\mathscr{E}$.

Considérons $\pi_{\{0,1\}^*}(\mathscr{A}_{\mathscr{E}})$ la projection de $\mathscr{A}_{\mathscr{E}}$ sur son entrée, c'est-à-dire l'automate sur $\{0,1\}^*$ qui a le même ensemble d'états que $\mathscr{A}_{\mathscr{E}}$ et pour relation de transition Δ' telle que si $s \xrightarrow{v|w} s' \in \Delta$, alors $s \xrightarrow{v} s' \in \Delta'$. L'automate $\pi_{\{0,1\}^*}(\mathscr{A}_{\mathscr{E}})$ accepte tous les mots de $\{0,1\}^*$. Autrement, un mot $w \in \{0,1\}^*$ qui ne serait pas accepté n'aurait pas d'image par $\mathscr{E}$ et donc pas d'image par $\mathscr{E}\cdot\mathscr{M}\cdot\mathscr{D}$, ce qui est contradictoire avec le fait que $(\mathscr{E},\mathscr{D})$ est un canal. Ainsi, on peut appliquer le lemme 3.6 sur $\pi_{\{0,1\}^*}(\mathscr{A}_{\mathscr{E}})$, et obtenir des états $s_0 \in I$, $s \in S$ et $s_f \in F$ et des chemins $s_0 \overset{v}{\Rightarrow} s$, $s \overset{v_0}{\Longrightarrow} s$, $s \overset{v_1}{\Longrightarrow} s$ et $s \overset{v'}{\Rightarrow} s_f$ dans $\pi_{\{0,1\}^*}(\mathscr{A}_{\mathscr{E}})$, tels que $\{v_0, v_1\}$ forme un code.

Soit $s_0 \overset{v|w}{\Longrightarrow} s$, $s \overset{v_0|w_0}{\Longrightarrow} s$, $s \overset{v_1|w_1}{\Longrightarrow} s$ et $s \overset{v'|w'}{\Longrightarrow} s_f$ des chemins dans $\mathscr{A}_{\mathscr{E}}$, où w, w_0, w_1, w' sont des mots de H^*. Ces chemins existent par la définition de Δ'.

Montrons que $\{w_0, w_1\}$ forme un code. Supposons, par contradiction, que $w_0\cdot w_1 = w_1\cdot w_0$. Considérons alors le mot $\alpha = w\cdot w_0\cdot w_1\cdot w' = w\cdot w_1\cdot w_0\cdot w'$. Le mot α est dans l'image par $\mathscr{E}$ à la fois de $\beta = v\cdot v_0\cdot v_1\cdot v'$ et de $\gamma = v\cdot v_1\cdot v_0\cdot v'$, par les chemins acceptants ci-dessus. Ainsi, comme $\alpha \in \mathscr{E}(\beta)$, on a $(\mathscr{M}\cdot\mathscr{D})(\alpha) \subseteq (\mathscr{E}\cdot\mathscr{M}\cdot\mathscr{D})(\beta) = \{\beta\}$. De même, $\alpha \in \mathscr{E}(\gamma)$ donc $(\mathscr{M}\cdot\mathscr{D})(\alpha) \subseteq (\mathscr{E}\cdot\mathscr{M}\cdot\mathscr{D})(\gamma) = \{\gamma\}$. Par $\mathscr{M}\cdot\mathscr{D}$, α a au moins une image θ car $\mathscr{E}$ a pour image exactement le domaine de $\mathscr{M}\cdot\mathscr{D}$. Donc $\theta \in \{\beta\} \cap \{\gamma\}$, ce qui n'est possible que si $\beta = \gamma$, c'est-à-dire $v\cdot v_0\cdot v_1\cdot v' = v\cdot v_1\cdot v_0\cdot v'$, ce qui contredit le fait que $\{v_0, v_1\}$ forme un code.

Remplacement de l'encodeur. On va maintenant remplacer $\mathscr{E}$ par un encodeur qui utilise le quadruplet (w, w_0, w_1, w') trouvé ci-dessus pour encoder les messages. Cela revient à d'abord transformer un 0 en la suite de bits v_0, et un 1 en la suite de bits v_1. Il faut en conséquence modifier le décodage. En effet, lorsqu'il reçoit v_0, le décodeur doit maintenant l'interpréter en 0 (et de même pour v_1). Plus formellement, on utilise les encodeurs et décodeurs canoniques de $\{0,1\}^*$ vers $\{0,1\}^*$ qui réalisent ce réencodage : soit $\mathscr{E}_{\text{bin}} = \mathscr{E}_{\text{C}}(v, v_0, v_1, v')$ et $\mathscr{D}_{\text{bin}} = \mathscr{D}_{\text{C}}(v, v_0, v_1, v')$. Remarquons qu'on a bien

$$\begin{aligned}
(\mathscr{E}_{\text{bin}}\cdot\mathscr{E})\cdot\mathscr{M}\cdot(\mathscr{D}\cdot\mathscr{D}_{\text{bin}}) &= \mathscr{E}_{\text{bin}}\cdot(\mathscr{E}\cdot\mathscr{M}\cdot\mathscr{D})\cdot\mathscr{D}_{\text{bin}} \\
&= \mathscr{E}_{\text{bin}}\cdot Id(\{0,1\}^*)\cdot\mathscr{D}_{\text{bin}} \\
&= \mathscr{E}_{\text{bin}}\cdot\mathscr{D}_{\text{bin}} \\
(\mathscr{E}_{\text{bin}}\cdot\mathscr{E})\cdot\mathscr{M}\cdot(\mathscr{D}\cdot\mathscr{D}_{\text{bin}}) &= Id(\{0,1\}^*)
\end{aligned}$$

ainsi $(\mathscr{E}_{\text{bin}}\cdot\mathscr{E}, \mathscr{D}\cdot\mathscr{D}_{\text{bin}})$ est aussi un canal pour $\mathscr{M}$.

Considérons l'encodeur canonique $\mathscr{E}_{\text{in}}$ de $\{0,1\}^*$ vers H^* qui utilise le quadruplet (w, w_0, w_1, w') : $\mathscr{E}_{\text{in}} = \mathscr{E}_{\text{C}}(w, w_0, w_1, w')$. On a $\mathscr{E}_{\text{in}} \subseteq \mathscr{E}_{\text{bin}}\cdot\mathscr{E}$ par construction. Donc

$$\mathscr{E}_{\text{in}}\cdot\mathscr{M}\cdot\mathscr{D}\cdot\mathscr{D}_{\text{bin}} \subseteq Id(\{0,1\}^*).$$

Or l'image de $\mathscr{E}$ est toute entière dans le domaine de $\mathscr{M}\cdot\mathscr{D}$, donc en particulier un mot θ de $w\cdot\{w_0, w_1\}^*\cdot w'$ aura au moins une image par $\mathscr{M}\cdot\mathscr{D}$. Si

$$\theta = w\cdot w_{i_1}\cdot w_{i_2}\cdots w_{i_n}\cdot w' \quad \text{avec } \forall 1 \le j \le n,\ i_j \in \{0,1\},$$

on pose $\xi = v\cdot v_{i_1}\cdot v_{i_2}\cdots v_{i_n}\cdot v'$. Le mot θ est une image par $\mathscr{E}$ de ξ, donc l'image de θ par $\mathscr{M}\cdot\mathscr{D}$ est donc ξ. Et ξ est dans le domaine de $\mathscr{D}_{\text{bin}}$. Donc θ a bien une image par

$\mathscr{M}\cdot\mathscr{D}\cdot\mathscr{D}_{\text{bin}}$. Ainsi $Im(\mathscr{E}_{\text{in}}) \subseteq Dom(\mathscr{M}\cdot\mathscr{D}\cdot\mathscr{D}_{\text{bin}})$ et tout mot de $\{0,1\}^*$ a au moins une image par $\mathscr{E}_{\text{in}}\cdot\mathscr{M}\cdot\mathscr{D}\cdot\mathscr{D}_{\text{bin}}$. Donc

$$\mathscr{E}_{\text{in}}\cdot\mathscr{M}\cdot\mathscr{D}\cdot\mathscr{D}_{\text{bin}} = Id(\{0,1\}^*)$$

et, en posant $\mathscr{D}_{\text{out}} = \mathscr{D}\cdot\mathscr{D}_{\text{bin}}$, on obtient que la paire $(\mathscr{E}_{\text{in}},\mathscr{D}_{\text{out}})$ est aussi un canal pour $\mathscr{M}$.

Des boucles codantes dans l'agrégat encodeur-système. Il faut à présent transformer le décodeur, pour l'instant $\mathscr{D}_{\text{out}}$, en un décodeur canonique. Pour ce faire, on démontre l'existence de boucles codantes dans $\mathscr{E}_{\text{in}}\cdot\mathscr{M}$. On en déduit l'existence de la structure du décodeur canonique dans $\mathscr{D}_{\text{out}}$ à laquelle on le restreint.

Plus précisément, on considère le transducteur $\mathscr{A}_{\text{enc}}$ qui réalise la relation $\mathscr{M}_{\text{enc}} = \mathscr{E}_{\text{in}}\cdot\mathscr{M}\cdot Id(Dom(\mathscr{D}_{\text{out}}))$. Notons que $Id(Dom(\mathscr{D}_{\text{out}}))\cdot\mathscr{D}_{\text{out}} = \mathscr{D}_{\text{out}}$, donc $(\mathscr{E}_{\text{in}}, Id(Dom(\mathscr{D}_{\text{out}}))\cdot\mathscr{D}_{\text{out}})$ est aussi un canal. Ainsi $\mathscr{M}_{\text{enc}}\cdot\mathscr{D}_{\text{out}} = Id(\{0,1\}^*)$ et $\mathscr{M}_{\text{enc}}$ est complet sur $\{0,1\}^*$. On peut donc appliquer une nouvelle fois le lemme 3.6 à $\pi_{\{0,1\}^*}(\mathscr{A}_{\text{enc}})$, la projection sur l'entrée de $\mathscr{A}_{\text{enc}}$. Une fois encore, on peut replacer les chemins obtenus dans $\pi_{\{0,1\}^*}(\mathscr{A}_{\text{enc}})$ par le lemme 3.6 dans l'automate $\mathscr{A}_{\text{enc}}$. Ceci nous donne des états s'_0, s' et s'_f de $\mathscr{A}_{\text{enc}}$ (avec s'_0 initial et s'_f final) et les chemins $s'_0 \xRightarrow{u|\ell} s'$, $s' \xRightarrow{u_0|\ell_0} s'$, $s' \xRightarrow{u_1|\ell_1} s'$ et $s' \xRightarrow{u'|\ell'} s'_f$.

Remplacement du décodeur. De la même manière que l'on a réencodé sur le quadruplet (w,w_0,w_1,w') (en entrée et en sortie), on va pouvoir réencoder en utilisant le quadruplet u,u_0,u_1,u' en entrée et le quadruplet $(\ell,\ell_0,\ell_1,\ell')$ en sortie. On considère donc l'encodeur canonique $\mathscr{E}'_{\text{bin}} = \mathscr{E}_{\text{C}}(u,u_0,u_1,u')$ et le décodeur canonique $\mathscr{D}_{\text{ok}} = \mathscr{D}_{\text{C}}(\ell,\ell_0,\ell_1,\ell')$.

Un mot binaire η de longueur n a, entre autres, comme image par $\mathscr{E}'_{\text{bin}}\cdot\mathscr{M}_{\text{enc}}$, le mot $\lambda = \ell\cdot\ell_{\eta[1]}\cdots\ell_{\eta[n]}\cdot\ell'$, qui est lui même décodé en η par $\mathscr{D}_{\text{ok}}$. Ainsi $Id(\{0,1\}^*) \subseteq \mathscr{E}'_{\text{bin}}\cdot\mathscr{M}_{\text{enc}}\cdot\mathscr{D}_{\text{ok}}$.

Regardons toutes les images possibles de η par l'image par $\mathscr{E}'_{\text{bin}}\cdot\mathscr{M}_{\text{enc}}$. La seule image de η par $\mathscr{E}'_{\text{bin}}$ est le mot (binaire) $\zeta = u\cdot u_{\eta[1]}\cdots u_{\eta[n]}\cdot u'$. Les images de ζ par $\mathscr{M}_{\text{enc}}$ sont toutes bien décodées (en ζ) par $\mathscr{D}_{\text{out}}$. Soit μ l'une de ces images. Si $\mu \notin Dom(\mathscr{D}_{\text{ok}})$, alors elle n'est pas considérée dans la composition $\mathscr{E}'_{\text{bin}}\cdot\mathscr{M}_{\text{enc}}\cdot\mathscr{D}_{\text{ok}}$. Si $\mu \in Dom(\mathscr{D}_{\text{ok}})\setminus\{\lambda\}$, alors $\mu = \ell\cdot\ell_{i_1}\cdots\ell_{i_k}\cdot\ell'$ (où les indices i_j sont des bits). Dans ce cas, μ est aussi l'image de $\zeta' = u\cdot u_{i_1}\cdots u_{i_k}\cdot u'$. Comme ζ' est, lui aussi, bien transmis par $\mathscr{M}_{\text{enc}}\cdot\mathscr{D}_{\text{out}}$, on aurait $\zeta = \zeta'$, ce qui est contradictoire. Notons que l'on obtient de la même façon que ℓ_0 et ℓ_1 ne commutent pas. On a donc $\mathscr{E}'_{\text{bin}}\cdot\mathscr{M}_{\text{enc}}\cdot\mathscr{D}_{\text{ok}} = Id(\{0,1\}^*)$. On en déduit que $(\mathscr{E}'_{\text{bin}}\cdot\mathscr{E}_{\text{in}}, Id(Dom(\mathscr{D}_{\text{out}}))\cdot\mathscr{D}_{\text{ok}})$ est un canal pour $\mathscr{M}$.

Conclusion. Remarquons que $Dom(\mathscr{D}_{\text{ok}}) \subseteq Dom(\mathscr{D}\cdot\mathscr{D}_{\text{bin}})$ par construction, donc $Id(Dom(\mathscr{D}_{\text{out}}))\cdot\mathscr{D}_{\text{ok}} = \mathscr{D}_{\text{ok}}$. On pose $\mathscr{E}_{\text{ok}} = \mathscr{E}'_{\text{bin}}\cdot\mathscr{E}_{\text{in}}$. On a bien $\mathscr{E}_{\text{ok}}\cdot\mathscr{M}\cdot\mathscr{D}_{\text{ok}} = Id(\{0,1\}^*)$, donc $(\mathscr{E}_{\text{ok}},\mathscr{D}_{\text{ok}})$ est un canal pour $\mathscr{M}$. De plus,

$$\mathscr{E}_{\text{ok}} = \mathscr{E}'_{\text{bin}}\cdot\mathscr{E}_{\text{in}} = \mathscr{E}_{\text{C}}(u,u_0,u_1,u')\cdot\mathscr{E}_{\text{C}}(w,w_0,w_1,w')$$

où $\{u_0,u_1\}$ (resp. $\{w_0,w_1\}$) forme un code sur $\{0,1\}^*$ (resp. H^*). Selon le lemme 3.8, $\mathscr{E}_{\text{ok}}$ est une relation canonique $\mathscr{E}_{\text{C}}(h,h_0,h_1,h')$ pour un certain quadruplet, avec $\{h_0,h_1\}$ qui forme un code sur H^*.

On a donc un encodeur et un décodeur canoniques $\mathscr{E}_{\text{ok}} = \mathscr{E}_{\text{C}}(h, h_0, h_1, h')$ et $\mathscr{D}_{\text{ok}} = \mathscr{D}_{\text{C}}(\ell, \ell_0, \ell_1, \ell')$ tels que $(\mathscr{E}_{\text{ok}}, \mathscr{D}_{\text{ok}})$ est un canal pour $\mathscr{M}$, ce qui conclut la preuve. □

L'existence d'encodeurs canoniques fournit des informations sur la structure du transducteur réalisant $\mathscr{M}$, et en particulier l'existence d'un état encodant. La preuve du théorème 3.5 est immédiatement déduite du théorème 3.7 et du lemme suivant :

Lemme 3.9. *Soit $\mathscr{M}$ une relation rationnelle et $\mathscr{A}$ un transducteur la réalisant. Si $(\mathscr{E}_{\text{C}}(h, h_0, h_1, h'), \mathscr{D}_{\text{C}}(\ell, \ell_0, \ell_1, \ell'))$ est un canal pour $\mathscr{M}$, alors $\mathscr{A}$ a un état encodant.*

Démonstration. Posons $\mathscr{E}_{\text{C}} = \mathscr{E}_{\text{C}}(h, h_0, h_1, h')$ et $\mathscr{D}_{\text{C}} = \mathscr{D}_{\text{C}}(\ell, \ell_0, \ell_1, \ell')$. La relation $\mathscr{E}_{\text{C}}$ est une bijection entre $\{0,1\}^*$ et $h \cdot \{h_0, h_1\}^* \cdot h'$. Pareillement, $\mathscr{D}_{\text{C}}$ est une bijection entre $\ell \cdot \{\ell_0, \ell_1\}^* \cdot \ell'$ et $\{0,1\}^*$. Comme $\mathscr{E}_{\text{C}} \cdot \mathscr{M} \cdot \mathscr{D}_{\text{C}} = Id(\{0,1\}^*)$, la relation $\mathscr{M}$ contient $\mathscr{L}_0 = (h|\ell) \cdot \{(h_0|\ell_0), (h_1|\ell_1)\}^* \cdot (h', \ell')$. Donc $\mathscr{A}$ accepte $\mathscr{L}_0$. D'autre part, $(h_0|\ell_0)$ et $(h_1|\ell_1)$ ne commutent pas car les deux composantes ne commutent pas. Selon le lemme 3.6, il existe, en particulier, des mots $w_0 \in \{(h_0|\ell_0)\}^*$, $w_1 \in \{(h_0|\ell_0), (h_1|\ell_1)\}^* \cdot (h_1|\ell_1) \cdot \{(h_0|\ell_0), (h_1|\ell_1)\}^*$ et un état s de $\mathscr{A}$ tels que $s \xRightarrow{w_0} s$ et $s \xRightarrow{w_1} s$ sont des chemins dans $\mathscr{A}$ et $w_0 \cdot w_1 \neq w_1 \cdot w_0$.

Remarquons que dans ce cas, chaque composante de w_0 ne commute pas avec celle de w_1. En effet, en posant $w_0 = (u_0|v_0)$ et $w_1 = (u_1|v_1)$, on a $u_0 \in \{h_0\}^*$ et $u_1 \in \{h_0, h_1\}^* \cdot h_1 \cdot \{h_0, h_1\}^*$. Comme $\{h_0, h_1\}$ forme un code sur H^*, on obtient que $\{u_0, u_1\}$ aussi. De même, on déduit que $\{v_0, v_1\}$ est un code sur L^*. On a bien la définition d'un état encodant, ce qui conclut la preuve. □

3.3.2 Le cas des relations fonctionnelles

On a donc une condition nécessaire pour l'existence d'un canal. Comme il a été vu, cette condition n'est pas suffisante dans le cas général. Cependant, elle le devient dans le cas des relations *fonctionnelles*. À titre d'exemple, le modèle de la figure 3.5 n'est pas fonctionnel, et l'état encodant s_2 ne peut pas transmettre de message à cause de s_1 qui peut « brouiller » le message.

La preuve repose sur des propriétés structurelles des transducteurs fonctionnels, et en particulier de ce qui ce rapproche d'une notion faible de déterminisme. Le fait qu'un mot ne soit pas transmis par le système de manière ambiguë facilite l'existence d'un canal. Dans ce cas, la propriété de canal est purement structurelle, s'approchant de la notion de [HZD05]. On peut alors décider de l'existence d'un canal et synthétiser un encodeur et un décodeur, en temps polynomial par rapport à la taille du transducteur [1] modélisant le système. On a ainsi le résultat suivant :

Lemme 3.10 (Condition suffisante d'existence d'un canal)**.** *Soit $\mathscr{M}$ une fonction rationnelle réalisée par un transducteur $\mathscr{A}$. Si $\mathscr{A}$ a un état encodant, alors $\mathscr{M}$ a un canal.*

Démonstration. Soit s un état encodant et considérons les chemins associés $s_0 \xRightarrow{h|\ell} s$ et $s \xRightarrow{h'|\ell'} s_f$ ainsi que les boucles $s \xRightarrow{h_0|\ell_0} s$ et $s \xRightarrow{h_1|\ell_1} s$, où $\{h_0, h_1\}$ et $\{\ell_0, \ell_1\}$ forment des codes. Soit $\mathscr{E} = \mathscr{E}_{\text{C}}(h, h_0, h_1, h')$ et $\mathscr{D} = \mathscr{D}_{\text{C}}(\ell, \ell_0, \ell_1, \ell')$ les encodeurs canoniques

1. La taille d'un transducteur comprend aussi la taille des étiquettes de ses transitions, car celles-ci font partie de l'entrée de la machine de Turing qui exécute l'algorithme.

utilisant ces boucles. Rappelons que $\mathscr{A}$ est supposé émondé. Ainsi un mot $\beta \in \{0,1\}^*$ est transmis correctement en utilisant s. D'autre part, l'image (unique) de β par $\mathscr{E}$ a une seule image par $\mathscr{M}$, puisque celle-ci est fonctionnelle. Le décodeur $\mathscr{D}$ étant aussi fonctionnel, le décodage est lui aussi correct. Plus précisément, si $|\beta| = n$:

$$\begin{aligned}
\mathscr{E}(\beta) &= h \cdot h_{\beta[1]} \cdots h_{\beta[n]} \cdot h' \\
\mathscr{M}(\mathscr{E}(\beta)) &= \ell \cdot \ell_{\beta[1]} \cdots \ell_{\beta[n]} \cdot \ell' \\
\mathscr{D}(\mathscr{M}(\mathscr{E}(\beta))) &= \beta[1] \cdots \beta[n] \\
(\mathscr{E} \cdot \mathscr{M} \cdot \mathscr{D})(\beta) &= \beta
\end{aligned}$$

donc $(\mathscr{E}, \mathscr{D})$ est un canal pour $\mathscr{M}$. □

La condition nécessaire et suffisante permet donc de ramener le problème d'existence d'un canal, qui est par essence *globale* à la relation rationnelle, à une propriété structurelle *locale* du transducteur qui la réalise, et ainsi de décider le problème :

Théorème 3.11. *Soit $\mathscr{M}$ une fonction rationnelle réalisée par un transducteur $\mathscr{A}$. Il peut être décidé en temps polynomial par rapport à la taille de $\mathscr{A}$ si $\mathscr{M}$ contient un canal.*

Gilles Benattar [Ben11] a généralisé ce résultat dans le cas des relations formées par une union finie de fonctions rationnelles.

La preuve du théorème 3.11 repose sur la possibilité de décider si un état donné est encodant. Il suffit ensuite de tester pour chaque état du transducteur s'il vérifie cette propriété.

Lemme 3.12. *Soit $\mathscr{A}$ un transducteur fonctionnel et s un état de $\mathscr{A}$. Il peut être décidé en $O\left(|\mathscr{A}|^3\right)$ si s est un état encodant.*

La détection d'un état encodant se base sur la détection de deux boucles pour lesquels les mots produits ne commutent pas. On pose donc

$$NCom(w, \mathscr{M}) = \left\{ v \in H^* \;\middle|\; \mathscr{M}(w) \cdot \mathscr{M}(v) \neq \mathscr{M}(v) \cdot \mathscr{M}(w) \right\}$$

l'ensemble des mots dont l'image par $\mathscr{M}$ ne commute pas avec celle de v.

Proposition 3.13. *Soit $\mathscr{M}$ est une fonction rationnelle de $H^* \times L^*$ et $w \in Dom(\mathscr{M})$. L'ensemble $NCom(w, \mathscr{M})$ est un langage régulier de H^*.*

Démonstration. Soit $v = \mathscr{M}(w)$. On pose $C(v) = \{u \in L^* \mid u \cdot v = v \cdot u\}$ l'ensemble des mots qui commutent avec v. On sait donc qu'il existe un mot $t \in L^*$ tel que $C(v) = t^*$ (t étant le plus petit mot qui commute avec v). Ainsi $C(v)$ est régulier et $Im(\mathscr{M}) \setminus C(v)$ l'est aussi. Finalement $NCom(w, \mathscr{M}) = \mathscr{M}^{-1}(Im(\mathscr{M}) \setminus C(v))$ est régulier. □

La propriété d'état encodant peut donc être reformulée en terme de commutation. Plus précisément, pour une fonction rationnelle $\mathscr{M}$ réalisée par un transducteur $\mathscr{A}$ et pour un état s de $\mathscr{A}$, on note $\mathscr{A}_s$ le transducteur identique à $\mathscr{A}$ hormis que s est le seul état initial et final. $\mathscr{A}_s$ est lui aussi émondé, et réalise la relation $\mathscr{M}_s$, qui est elle aussi fonctionnelle. Le fait que s est encodant dans $\mathscr{A}_s$ se traduit en terme de commutation d'image, et si s est encodant dans $\mathscr{A}_s$ il l'est aussi dans $\mathscr{A}$ (en ajoutant des chemins entre un état initial et s et entre s et un état final).

Proposition 3.14. *S'il existe $w \in \mathscr{M}_s^{-1}(Im(\mathscr{M}_s) \setminus \{\varepsilon\})$ tel que $NCom(w, \mathscr{M}_s) \neq \emptyset$, alors s est un état encodant. Réciproquement, si s est un état encodant, alors pour tout mot $w \in \mathscr{M}_s^{-1}(Im(\mathscr{M}_s) \setminus \{\varepsilon\})$, $NCom(w, \mathscr{M}_s) \neq \emptyset$.*

Démonstration. Supposons que $h_0 \in \mathscr{M}_s^{-1}(Im(\mathscr{M}_s) \setminus \{\varepsilon\})$ est tel que $NCom(h_0, \mathscr{M}_s) \neq \emptyset$. Puisque $\mathscr{M}$ est fonctionnel et que $\mathscr{M}_s(\varepsilon) = \varepsilon$, on sait que $h_0 \neq \varepsilon$. On choisit un mot h_1 de $NCom(h_0, \mathscr{M}_s)$. On pose $\ell_0 = \mathscr{M}_s(h_0)$ et $\ell_1 = \mathscr{M}_s(h_1)$. Les mots ℓ_0 et ℓ_1 ne commutent pas par définition de $NCom(h_0, \mathscr{M})$. Par ailleurs, h_0 et h_1 ne commutent pas non plus. En effet, en supposant que $h_0 \cdot h_1 = h_1 \cdot h_0$, on a

$$\begin{aligned}
\ell_0 \cdot \ell_1 &= \mathscr{M}_s(h_0) \cdot \mathscr{M}_s(h_1) \\
&= \mathscr{M}_s(h_0 \cdot h_1) \quad (\text{car } \mathscr{M}_s \text{ est fonctionnel et } s \text{ est le seul état initial et final}) \\
&= \mathscr{M}_s(h_1 \cdot h_0) \quad (\text{car } h_0 \cdot h_1 = h_1 \cdot h_0) \\
&= \mathscr{M}_s(h_1) \cdot \mathscr{M}_s(h_0) \\
\ell_0 \cdot \ell_1 &= \ell_1 \cdot \ell_0
\end{aligned}$$

ce qui est contradictoire. Ainsi s est un état encodant avec les boucles étiquetées par $h_0|\ell_0$ et $h_1|\ell_1$.

Réciproquement, si s est un état encodant avec des boucles $h_0|\ell_0$ et $h_1|\ell_1$. On a bien $\ell_0 = \mathscr{M}_s(h_0)$ et $\ell_1 = \mathscr{M}_s(h_1)$. Soit $w \in \mathscr{M}_s^{-1}(Im(\mathscr{M}_s) \setminus \{\varepsilon\})$ et $v = \mathscr{M}_s(w)$; on a encore, $w \neq \varepsilon$. Supposons que $NCom(w, \mathscr{M}_s) = \emptyset$. En particulier, $h_0 \notin NCom(w, \mathscr{M}_s)$, donc h_0 commute avec v et il existe u_0 tel que $h_0, v \in u_0^*$. De même, $h_1 \notin NCom(w, \mathscr{M}_s)$, donc h_1 commute avec v et il existe u_1 tel que $h_1, v \in u_1^*$. On peut donc écrire $h_0 = u_0^n$, $h_1 = u_1^m$ et $v = u_0^k = u_1^p$, donc

$$h_0^k \cdot h_1^p = u_0^{k \times n} \cdot u_1^{p \times m} = v^{n+m} = u_1^{p \times m} \cdot u_0^{k \times n} = h_1^p \cdot h_0^k$$

et $\{h_0, h_1\}$ ne forme pas un code ce qui est contradictoire. □

Le problème de savoir si un état est encodant revient donc à faire un test du vide sur un langage régulier.

Démonstration du lemme 3.12. Soit $\mathscr{M}$ une fonction rationnelle réalisée par $\mathscr{A}$ et s un état de $\mathscr{A}$. On considère le transducteur $\mathscr{A}_s = \langle S, H^* \times L^*, \Delta, \{s\}, \{s\} \rangle$, que l'on peut construire en temps $O\left(|\mathscr{A}|^2\right)$ (pour l'émonder). Dans la suite de cette preuve, les chemins considérés seront dans $\mathscr{A}_s$.

On cherche ensuite un mot dont l'image par $\mathscr{M}_s$ n'est pas ε. Il suffit pour cela de trouver une transition de Δ qui produit un mot non vide : $s_1 \xrightarrow{v_0|w_0} s_2$ avec $w_0 \in L^+$. Le parcours de toutes les transitions s'effectue en $O(|\mathscr{A}_s|)$. Si une telle transition n'existe pas, alors $Im(\mathscr{M}_s) = \{\varepsilon\}$ et s n'est pas un état encodant, et l'algorithme retourne *faux*. Si cette transition existe, alors on construit, en $O\left(|\mathscr{A}_s|^2\right)$, des chemins $s \xRightarrow{v_1|w_1} s_1$ et $s_2 \xRightarrow{v_2|w_2} s$ (qui existent car $\mathscr{A}_s$ a été émondé auparavant).

On pose $v = v_1 \cdot v_0 \cdot v_2$ et $w = \mathscr{M}_s(v) = w_1 \cdot w_0 \cdot w_2$. Le but est de décider si l'ensemble des mots de $Im(\mathscr{M}_s)$ qui ne commutent pas avec w est vide, c'est-à-dire si $NCom(v, \mathscr{M}_s) = \emptyset$. L'ensemble $C(w)$ des mots qui commutent avec w (pas nécessairement des mots de $Im(\mathscr{M}_s)$) est un ensemble de la forme t^*, où t est un préfixe de w, donc de taille inférieure à $|\mathscr{A}_s|$. Un automate déterministe sur L^* qui accepte $\overline{t^*}$ et de taille $O(|t|)$ est construit, et intersecté avec l'automate qui reconnaît $Im(\mathscr{M}_s)$ (la

projection de $\mathscr{A}_s$ sur sa sortie, donc de même taille que $\mathscr{A}_s$). On obtient donc un automate $\mathscr{A}_{NCom}$ de taille $O\left(|\mathscr{A}_s|^2\right)$ qui reconnaît $Im(\mathscr{M}) \setminus C(w)$. Le test du vide sur cet automate peut être effectué en temps linéaire (vis à vis de la taille de $\mathscr{A}_{NCom}$) par un simple parcours en largeur depuis l'unique état initial.

Si $\mathscr{L}(\mathscr{A}_{NCom}) = \emptyset$, alors $NCom(v, \mathscr{M}_s) = \mathscr{M}^{-1}(Im(\mathscr{M}) \setminus C(w)) = \mathscr{M}^{-1}(\emptyset) = \emptyset$, et l'algorithme répond *faux*. Sinon, c'est-à-dire si $\mathscr{L}(\mathscr{A}_{NCom}) \neq \emptyset$, puisque les mots reconnus par $\mathscr{A}_{NCom}$ sont dans l'image de $\mathscr{M}_s$, $NCom(v, \mathscr{M}_s) = \mathscr{M}^{-1}(Im(\mathscr{M}) \setminus C(w)) \neq \emptyset$. Un mot $v' \in NCom(v, \mathscr{M}_s)$ de longueur en $O(|\mathscr{A}_{NCom}|)$ est alors trouvé par le test du vide, et son image $w' = \mathscr{M}_s(v')$ est calculée en $O(|\mathscr{A}_{NCom}| \times |\mathscr{A}_s|)$. L'algorithme revoie donc *vrai* ainsi que les chemins $s \overset{v|w}{\Longrightarrow} s$ et $s \overset{v'|w'}{\Longrightarrow} s$.

La complexité globale de l'algorithme est majorée par l'étape de calcul de w' qui est en $O(|\mathscr{A}_{NCom}| \times |\mathscr{A}_s|)$ c'est-à-dire en $O\left(|\mathscr{A}_s|^3\right)$, ce qui est équivalent à $O\left(|\mathscr{A}|^3\right)$. □

3.3.3 L'indécidabilité du problème

Malheureusement, dans le cas général, c'est-à-dire si l'on ne suppose plus les relations fonctionnelles, le problème d'existence d'un canal devient indécidable. On prouve par encodage du problème de correspondance de Post (ou simplement *problème de Post* ou PCP [Pos46]) le théorème suivant :

Théorème 3.15 (Indécidabilité de l'existence d'un canal)**.** *Le problème de savoir, étant donné un transducteur rationnel $\mathscr{A}$, si la relation qu'il réalise contient un canal est indécidable.*

Une instance $\mathscr{I}$ du problème de Post est un ensemble $\langle (x_1, y_1), \ldots, (x_n, y_n) \rangle$ de n paires de mots sur un alphabet A. Une solution σ de $\mathscr{I}$ est une suite finie d'indices $i_1, \ldots i_k$ (compris entre 1 et n) telle que $x_{i_1} \cdots x_{i_k} = y_{i_1} \cdots y_{i_k}$. L'instance $\mathscr{I}$ peut être vue comme une paire de morphismes (x, y) de $\{1, \ldots, n\}$ vers $\mathscr{A}^*$ défini par $z(i) = z_i$, où z désigne x ou y. Cette dernière notation est utilisée dans cette section chaque fois que l'on veut désigner x ou y indifféremment. Avec ces notations, une solution de $\mathscr{I}$ est donc une suite σ telle que $x(\sigma) = y(\sigma)$.

La suite vide étant toujours une solution à une instance de PCP, une solution non vide est appelée *non triviale*. Le problème de Post cherche à décider, étant donné $\mathscr{I}$ l'existence d'une solution non triviale pour $\mathscr{I}$. Si A a plus de deux lettres, le problème de Post est indécidable pour $n \geq 7$. De plus, on peut supposer qu'il n'y a pas dans $\mathscr{I}$ de paire $x_i = y_i$: l'instance a une solution de taille $k = 1$, ce qui peut être trivialement décidé.

Dans la suite, on construit à partir d'une instance $\mathscr{I}$ de PCP un transducteur $\mathscr{A}_{\mathscr{I}}$ réalisant $\mathscr{M}_{\mathscr{I}}$ tel que $\mathscr{M}_{\mathscr{I}}$ a un canal si et seulement si $\mathscr{I}$ a une solution non triviale. La construction de $\mathscr{A}_{\mathscr{I}}$ s'inspire de celle servant à prouver l'indécidabilité du problème d'égalité de langage de deux transducteurs rationnels [Gur89].

L'idée générale est la suivante : $\mathscr{A}_{\mathscr{I}}$ lit un bit et une suite d'indices. Si la suite d'indice est une solution à $\mathscr{I}$, alors le bit est transmis correctement. Sinon, le bit ou son opposé peut être transmis.

Plus précisément, soit σ une solution non triviale de $\mathscr{I}$ et b un bit. En lisant $b \cdot \sigma$ le transducteur $\mathscr{A}_{\mathscr{I}}$ produira n'importe quelle séquence $w \in A^+$ suivie d'un bit quelconque, à l'exception de la seule séquence $x(\sigma) \cdot \overline{b}$ (qui est aussi $y(\sigma) \cdot \overline{b}$), où $\overline{b} = 1 - b$

est l'opposé du bit b lu. En détectant ce « mot manquant », un observateur connaîtra la valeur de b. L'itération de ce motif permettra de transmettre n'importe quelle séquence de bits, et donc de réaliser un canal.

Construction de $\mathscr{A}_{\mathscr{J}}$

Soit $\mathbb{B} = \{\top, \bot\}$ avec $\overline{\top} = \bot$ et $\overline{\bot} = \top$. Les bits sont ici notés $\bot$ et $\top$ pour des raisons de lisibilité. On note $N = \{1, \dots, n\}$, ainsi que $A_{\mathbb{B}} = A \cup \mathbb{B}$ et $N_{\mathbb{B}} = N \cup \mathbb{B}$. On construit un transducteur $\mathscr{A}_{\mathscr{J}} = \langle Q, N_{\mathbb{B}}^* \times A_{\mathbb{B}}^*, \Delta, \{q_0\}, \{q_0\} \rangle$ qui réalise la relation rationnelle $\mathscr{M}_{\mathscr{J}}$ de $N_{\mathbb{B}}^*$ vers $A_{\mathbb{B}}^*$ telle que pour un bit $b \in \mathbb{B}$ et une séquence $\sigma \in N^*$

$$\mathscr{M}_{\mathscr{J}}(b \cdot \sigma) = \left(A^+ \cdot b\right) \cup \left((A^+ \setminus \{x(\sigma)\}) \cdot \overline{b}\right) \cup \left((A^+ \setminus \{y(\sigma)\}) \cdot \overline{b}\right)$$

Ce transducteur lit donc un bit et une suite d'indices et renvoie :

- soit un mot non vide suivi du même bit,
- soit un mot non vide qui n'est pas l'image de la séquence par x suivi du bit opposé,
- soit un mot non vide qui n'est pas l'image de la séquence par y suivi du bit opposé.

On étend $\mathscr{M}_{\mathscr{J}}$ à tout $N_{\mathbb{B}}^*$ en définissant $\mathscr{M}_{\mathscr{J}}(\varepsilon) = \varepsilon$ et pour des bits $b_1, \dots, b_p \in \mathbb{B}$ et des séquences $\sigma_1, \dots, \sigma_p \in N^*$,

$$\mathscr{M}_{\mathscr{J}}(b_1 \cdot \sigma_1 \cdots b_p \cdot \sigma_p) = \mathscr{M}_{\mathscr{J}}(b_1 \cdot \sigma_1) \cdots \mathscr{M}_{\mathscr{J}}(b_p \cdot \sigma_p),$$

et $\mathscr{M}_{\mathscr{J}}(w) = \emptyset$ si $w \notin (\mathbb{B} \cdot N^*)^*$.

Le transducteur $\mathscr{A}_{\mathscr{J}}$ est composé de deux parties (quasiment identiques) reliées par l'état initial et final q_0, et qui gardent en mémoire la valeur du bit lu (voir figure 3.7). On les appelle respectivement la $\top$ et la $\bot$-moitié de $\mathscr{A}_{\mathscr{J}}$.

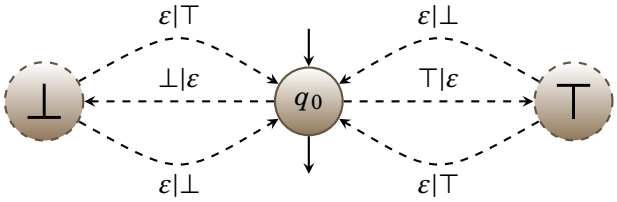


FIGURE 3.7: Structure symétrique de $\mathscr{A}_{\mathscr{J}}$.

La structure globale au sein d'une de ses moitiés (ici, $\top$) est représentée sur la figure 3.8. Dans l'état q_*, les entrées ne sont pas considérées, et n'importe quel mot de A^+ est produit, avant de revenir en q_0 en sortant le bit initialement lu. Cet état génère donc le langage $A^+ \cdot b$ mentionné dans l'image de $b \cdot \sigma$.

Le reste de cette $\top$-moitié se compose de deux parties assez similaires : l'une qui produira tout sauf $x(\sigma)$, suivi de $\overline{b}$, c'est-à-dire $(A^+ \setminus \{x(\sigma)\}) \cdot \overline{b}$; l'autre qui produira tout sauf $y(\sigma)$, suivi de $\overline{b}$, c'est-à-dire $(A^+ \setminus \{y(\sigma)\}) \cdot \overline{b}$. La partie de $\mathscr{A}_{\mathscr{J}}$ correspondant au « quart » $(\top, x)$ est représentée sur la figure 3.9. Dans les deux cas, le mot produit w est différent de $z(\sigma)$ (comme convenu, z désigne x ou y) selon trois possibilités d'erreurs :

- ou bien w est un préfixe strict de $z(\sigma)$ (mais pas le mot vide), auquel cas on atteint $q_<$;

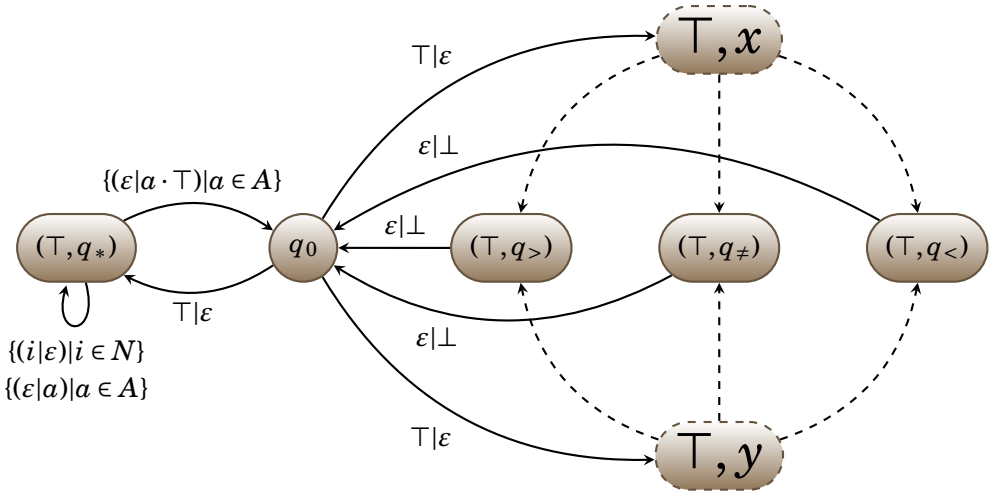


FIGURE 3.8: Structure de la ⊤-moitié de $\mathscr{A}_{\mathscr{I}}$.

– ou bien $z(\sigma)$ est un préfixe strict de w, auquel cas on atteint $q_>$;
– ou bien pour un certain i, la i-ième lettre de w est différente de la i-ième lettre de $z(\sigma)$, auquel cas on atteint $q_{\neq}$.

Dans ces trois états d'erreur, on peut alors lire et produire n'importe quoi, puis on revient dans q_0 en produisant $\overline{b}$. Ces états sont fusionnés car ils ne dépendent pas de si z représente x ou y.

Plus formellement, l'ensemble Q des états de $\mathscr{A}_{\mathscr{I}}$ est défini par

$$Q = \{q_0\} \cup \left(\mathbb{B} \times \left(\{q_*, q_x, q_y, q_x^\varepsilon, q_y^\varepsilon, q_>, q_<, q_{\neq}\} \cup Q_{\mathscr{I}} \cup Q_{\mathscr{I}}^\varepsilon\right)\right)$$

$$\text{où} \quad Q_{\mathscr{I}} = \left(\bigcup_{i=1}^{n} \bigcup_{j=1}^{|x_i|} \left\{q_x^{i,j}\right\}\right) \cup \left(\bigcup_{i=1}^{n} \bigcup_{j=1}^{|y_i|} \left\{q_y^{i,j}\right\}\right) \quad \text{et} \quad Q_{\mathscr{I}}^\varepsilon = \left(\bigcup_{i=1}^{n} \left\{q_x^{i,\varepsilon}\right\}\right) \cup \left(\bigcup_{i=1}^{n} \left\{q_y^{i,\varepsilon}\right\}\right)$$

sont des ensembles contenant respectivement un état par lettre de chaque mot de l'instance $\mathscr{I}$ et un état par mot de l'instance. L'état q_0 est le seul état initial et final.

L'ensemble Δ des transitions de $\mathscr{A}_{\mathscr{I}}$ est construit à l'aide des règles suivantes, pour $b \in \mathbb{B}$, $z \in \{x, y\}$, $i \in N$, et $a \in A$:

(R1) Pour $q \in \left\{q_*, q_x^\varepsilon, q_y^\varepsilon\right\}$, $q_0 \xrightarrow{b|\varepsilon} (b, q) \in \Delta$.
$\mathscr{A}_{\mathscr{I}}$ lit le bit b et fait le choix initial (de manière non déterministe) de savoir s'il va produire :
  – un mot (non vide) suivi de b (état q_*),
  – un mot (non vide) qui n'est pas $x(\sigma)$ suivi de $\overline{b}$ (état q_x),
  – ou un mot (non vide) qui n'est pas $y(\sigma)$ suivi de $\overline{b}$ (état q_y).

(R2) $(b, q_*) \xrightarrow{\varepsilon|a\cdot b} q_0 \in \Delta$.
Depuis q_*, le transducteur $\mathscr{A}_{\mathscr{I}}$ produit au moins une lettre (afin de ne pas sortir le mot vide) suivi du bit lu.

(R3) $(b, q_*) \xrightarrow{\varepsilon|a} (b, q_*) \in \Delta$.

(R4) $(b, q_*) \xrightarrow{i|\varepsilon} (b, q_*) \in \Delta$.
Cette règle et la précédente permettent à q_* de lire (sur N) et d'écrire (sur A) n'importe quelle lettre.

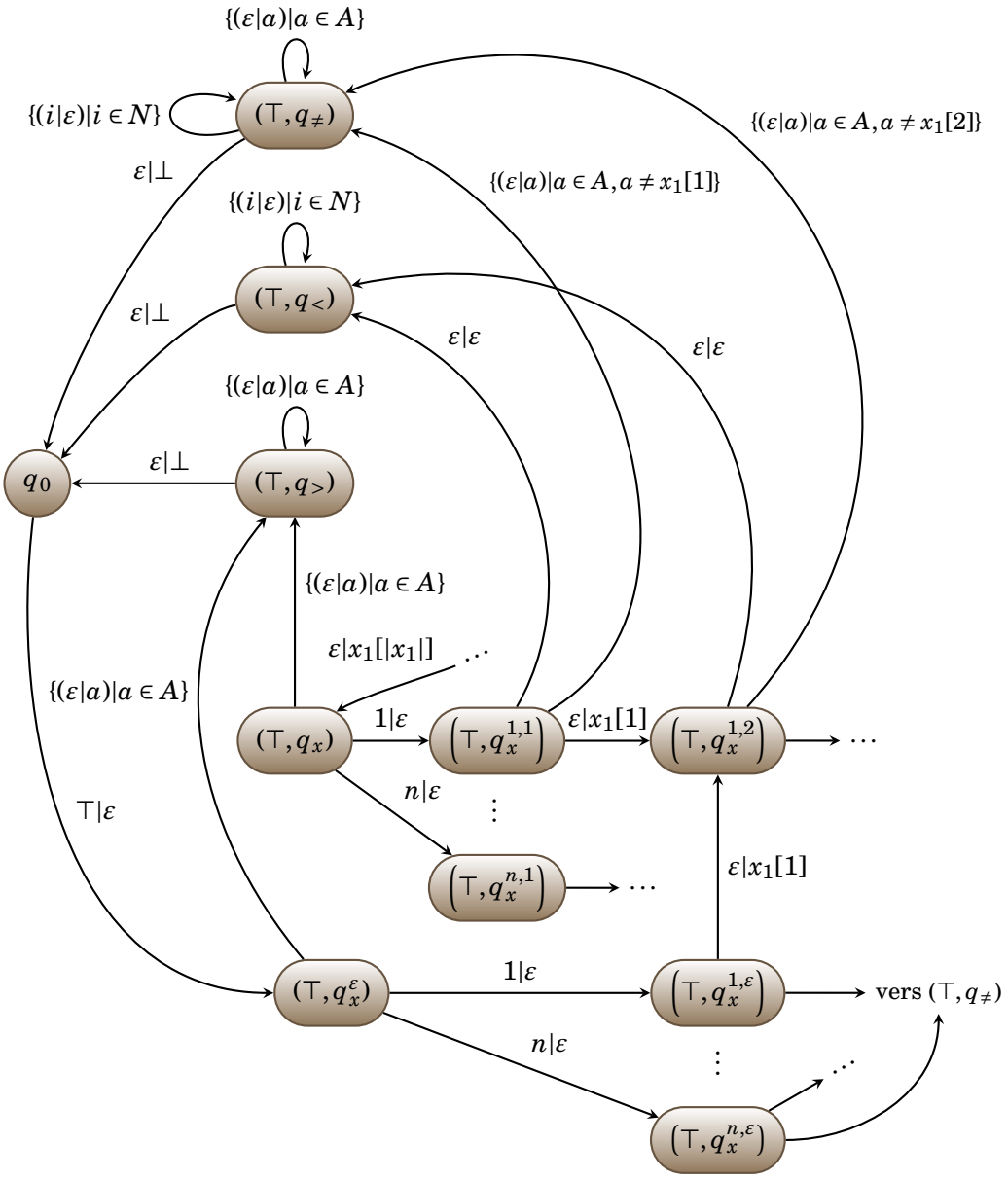


FIGURE 3.9: Structure de la partie du transducteur $\mathscr{A}_{\mathscr{J}}$ qui accepte la relation $\{(\top \cdot \sigma, u \cdot \bot) \,|\, u \in A^+ \setminus \{x(\sigma)\}\}^*$.

(R5) Si $|z_i| > 0$, alors $(b, q_z^\varepsilon) \xrightarrow{i|\varepsilon} \left(b, q_z^{i,\varepsilon}\right) \in \Delta$.

(R6) Si $|z_i| > 1$, alors $\left(b, q_z^{i,\varepsilon}\right) \xrightarrow{\varepsilon|z_i[1]} \left(b, q_z^{i,2}\right) \in \Delta$.

(R7) Si $|z_i| = 1$, alors $\left(b, q_z^{i,\varepsilon}\right) \xrightarrow{\varepsilon|z_i[|z_i|]} (b, q_z) \in \Delta$.

Les transitions créées par ces trois règles lisent l'indice i et commencent à écrire z_i, alors que rien n'a été encore produit.[2]

(R8) Si $|z_i| = 0$, alors $(b, q_z^\varepsilon) \xrightarrow{i|\varepsilon} (b, q_z^\varepsilon) \in \Delta$.

Cette règle est l'analogue des règles (R5-7) lorsque $z_i = \varepsilon$.

(R9) Si $|z_i| > 0$, alors $(b, q_z) \xrightarrow{i|\varepsilon} \left(b, q_z^{i,1}\right) \in \Delta$.

(R10) Pour $1 \leq j < |z_i|$, $(b, q_z^{i,j}) \xrightarrow{\varepsilon|z_i[j]} \left(b, q_z^{i,j+1}\right) \in \Delta$.

(R11) $\left(b, q_z^{i,|z_i|}\right) \xrightarrow{\varepsilon|z_i[|z_i|]} (b, q_z) \in \Delta$.

Les transitions créées par les trois règles précédentes permettent à $\mathscr{A}_\mathscr{J}$ dans q_z de lire l'indice i et de produire z_i, en retournant dans q_z.

(R12) Si $|z_i| = 0$, alors $(b, q_z) \xrightarrow{i|\varepsilon} (b, q_z) \in \Delta$.

Cette règle est l'analogue des règles (R9-11) lorsque $z_i = \varepsilon$.

(R13) Pour $1 \leq j < |z_i|$, $\left(b, q_z^{i,j}\right) \xrightarrow{\varepsilon|\varepsilon} (b, q_<) \in \Delta$.

Ces transitions interrompent la sortie de z_i, allant en $q_<$.

(R14) $(b, q_<) \xrightarrow{i|\varepsilon} (b, q_<) \in \Delta$.

Dans $q_<$, les entrées sont lues mais aucune sortie n'est produite.

(R15) $(b, q_z) \xrightarrow{\varepsilon|a} (b, q_>) \in \Delta$.

(R16) $(b, q_>) \xrightarrow{\varepsilon|a} (b, q_>) \in \Delta$.

Ces transitions produisent au moins une lettre de A sans lire d'entrées (elles devraient donc toutes avoir été lues auparavant).

(R17) Si $a \neq z_i[1]$, $\left(b, q_z^{i,\varepsilon}\right) \xrightarrow{\varepsilon|a} (b, q_\neq) \in \Delta$.

(R18) Pour $1 \leq j < |z_i|$, et si $a \neq z_i[j]$, $\left(b, q_z^{i,j}\right) \xrightarrow{\varepsilon|a} (b, q_\neq) \in \Delta$.

Le franchissement d'une transitions créé par l'une des deux règles précédentes insère une erreur dans z_i.

(R19) $(b, q_\neq) \xrightarrow{i|\varepsilon} (b, q_\neq) \in \Delta$.

(R20) $(b, q_\neq) \xrightarrow{\varepsilon|a} (b, q_\neq) \in \Delta$.

Dans l'état $q_\neq$, on peut lire n'importe quoi (sur N) et produire n'importe quoi (sur A).

(R21) Pour $q \in \{q_<, q_>, q_\neq\}$, $(b, q) \xrightarrow{\varepsilon|\overline{b}} q_0 \in \Delta$.

Le retour depuis un état d'erreur produit le bit opposé à celui qui avait été lu.

Ainsi les règles (R5-12) construisent une sous partie de $\mathscr{A}_\mathscr{J}$ qui peut produire $z(\sigma)$. Cependant, comme q_z^ε et q_z ne sont pas des état finaux, ce mot précis ne peut pas être produit dans cette partie. En effet, les règles (R13-20) introduisent des erreurs dans le mot w produit. Comme il a été mentionné, w peut être un prefixe strict de $z(\sigma)$ (règles (R13-14)), contenir $z(\sigma)$ comme préfixe strict (règles (R15-16)), ou contenir des erreurs (règles (R18-20)).

Si rien n'a été produit depuis le début de la séquence d'indice, on reste dans les états q_z^ε et $q_z^{i,\varepsilon}$ (règles (R5-8)), que l'on quitte en lisant la première lettre de z_i (règle (R6)). Notons qu'il n'y a pas d'analogue à la règle (R13) depuis des états $q_z^{i,\varepsilon}$, afin de ne pas sortir le mot vide.

2. Du moins depuis que l'on a lu le dernier bit b.

Les transitions produites par les règles (R2) et (R21) retournent à l'état q_0, ce qui permet au processus d'être répété. Ainsi, on aura bien l'extension de $\mathscr{M}_{\mathscr{J}}$ à tous les mots de $(\mathbb{B}\cdot N^*)^*$.

Correction du codage

On prouve maintenant que le codage de PCP est correct, c'est-à-dire :

Proposition 3.16 (Correction du codage de PCP)**.** *Soit $\mathscr{I}$ une instance du problème de Post et $\mathscr{M}_{\mathscr{J}}$ la relation rationnelle construite ci-dessus. L'instance $\mathscr{I}$ a une solution non triviale si et seulement si $\mathscr{M}_{\mathscr{J}}$ contient un canal.*

Cette proposition se prouve par les deux lemmes suivants.

Lemme 3.17. *Si $\mathscr{I}$ a une solution non triviale, alors $\mathscr{M}_{\mathscr{J}}$ contient un canal.*

Démonstration. Supposons que $\sigma = i_1\cdots i_k$ est une solution de $\mathscr{I}$, avec $k>0$ et posons $w = x(\sigma) = y(\sigma)$.

Considérons $\mathscr{E}_\sigma$ la relation rationnelle de $\{0,1\}^* \times N_{\mathbb{B}}^*$

$$\mathscr{E}_\sigma = \{(0|\bot\cdot\sigma),(1|\top\cdot\sigma)\}^*$$

réalisée par le transducteur de la figure 3.10(a). Soit β un mot de longueur n de $\{0,1\}^*$. L'unique image de β par $\mathscr{E}_\sigma$ est le mot

$$u = \beta[1]\cdot\sigma\cdots\beta[n]\cdot\sigma.$$

Pour chaque sous mot $\beta[j]\cdot\sigma$, le transducteur $\mathscr{A}_{\mathscr{J}}$ produira le langage

$$\left(A^+\cdot\beta[j]\right)\cup\left(\left(A^+\setminus\{w\}\right)\cdot\overline{\beta[j]}\right)$$

car w n'est produit ni par le quart $(\beta[j],x)$, ni par le quart $(\beta[j],y)$. Donc l'image de u par $\mathscr{M}_{\mathscr{J}}$ est l'ensemble de mots

$$\mathscr{M}_{\mathscr{J}}(u) = \left\{v_1\cdot b_1\cdots v_n\cdot b_n \;\middle|\; \forall 1\le j\le n : v_j\in A^+ \wedge b_j\in\mathbb{B}\wedge\left(v_j = w\Rightarrow b_j=\beta[j]\right)\right\}.$$

En particulier, le mot $v = w\cdot\beta[1]\cdots w\cdot\beta[n]$ est dans $\mathscr{M}_{\mathscr{J}}(u)$, tandis que tout mot de la forme $w\cdot b_1\cdots w\cdot b_n$, avec $(b_1,\ldots,b_n)\neq\left(\beta[1],\ldots,\beta[n]\right)$, n'y est pas.

Considérons maintenant la relation rationnelle $\mathscr{D}_\sigma$ de $A_{\mathbb{B}}^*$ vers $\{0,1\}^*$

$$\mathscr{D}_\sigma = \{(w\cdot\bot|0),(w\cdot\top|1)\}^*$$

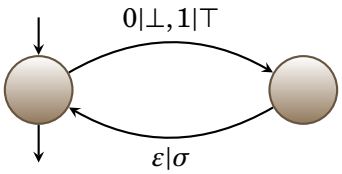
réalisée par le transducteur de la figure 3.10(b). Ce transducteur renvoie, pour un mot de la forme $w\cdot b_1\cdots w\cdot b_m$ en entrée, le mot $b_1\cdots b_m$. Les mots qui ne sont pas dans $(w\cdot\mathbb{B})^*$ n'ont pas d'image par $\mathscr{D}_\sigma$. Ainsi,

$$\mathscr{D}_\sigma\left(\mathscr{M}_{\mathscr{J}}(u)\right) = \mathscr{D}_\sigma(v) = \beta[1]\cdots\beta[n] = \beta.$$

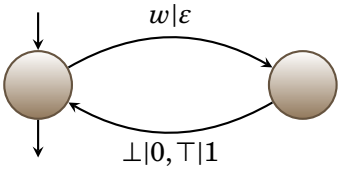
On a donc bien $\mathscr{E}_\sigma\cdot\mathscr{M}_{\mathscr{J}}\cdot\mathscr{D}_\sigma(\beta)=\beta$ pour tout mot binaire β, et donc que $(\mathscr{E}_\sigma,\mathscr{D}_\sigma)$ est un canal pour $\mathscr{M}_{\mathscr{J}}$. □

Exemple 3.2. Considérons l'instance de PCP $\mathscr{J} = \langle(abb,a),(b,abb),(a,bb)\rangle$. Le quart correspondant à x ainsi que l'état q_* de la $\top$-moitié du transducteur correspondant $\mathscr{A}_{\mathscr{J}}$ est représenté sur la figure 3.11.

Cette instance admet pour solution la séquence $\sigma = 1311322$ qui produit le mot $w = abbaabbabbabb$. En lisant le mot $\top 1311322$, le transducteur $\mathscr{A}_{\mathscr{J}}$ peut produire



(a) Transducteur de $\{0,1\}^*$ vers $N_{\mathbb{B}}^*$ qui réalise la relation $\mathscr{E}_\sigma$.



(b) Transducteur de $A_{\mathbb{B}}^*$ vers $\{0,1\}^*$ qui réalise la relation $\mathscr{D}_\sigma$.

FIGURE 3.10: Encodeur et decodeur réalisant respectivement $\mathscr{E}_\sigma$ et $\mathscr{D}_\sigma$, où σ est une solution non triviale de $\mathscr{I}$ et w le mot correspondant.

n'importe quel mot non vide suivi de $\top$, en ne visitant que les états q_* et q'_*. En particulier, $w \cdot \top$ peut être produit de cette façon.

Sur la même entrée, d'autres mots suivis de $\bot$ peuvent être produits. Par exemple le mot $abbaabbabaa\bot$ est la sortie de l'exécution

$$q_0 \xrightarrow{\top|\varepsilon} q_x^{\varepsilon} \xrightarrow{1|\varepsilon} q_x^{1,\varepsilon} \xrightarrow{\varepsilon|a} q_x^{1,2} \xrightarrow{\varepsilon|b} q_x^{1,3} \xrightarrow{\varepsilon|b} q_x \xrightarrow{3|\varepsilon} q_x^{3,1} \xrightarrow{\varepsilon|a} q_x \xrightarrow{1|\varepsilon} q_x^{1,1} \xrightarrow{\varepsilon|a} \cdots$$
$$\cdots \xrightarrow{\varepsilon|a} q_x^{1,2} \xrightarrow{\varepsilon|b} q_x^{1,3} \xrightarrow{\varepsilon|b} q_x \xrightarrow{1|\varepsilon} q_x^{1,1} \xrightarrow{\varepsilon|a} q_x^{1,2} \xrightarrow{\varepsilon|b} q_x^{1,3} \xrightarrow{\varepsilon|a} q_{\neq} \xrightarrow{\varepsilon|a} q_{\neq} \xrightarrow{\varepsilon|\bot} q_0.$$

Cependant, $w \cdot \bot$ n'est pas produit par $\mathscr{A}_{\mathscr{J}}$. En effet, après avoir lu $\top 1311322$ et produit w, l'exécution est soit dans q_*, soit dans q_x, soit dans q_y. Depuis, q_*, on ne peut pas produire $\bot$. Depuis q_x (ou, sans perte de généralité, q_y), on doit nécessairement lire ou écrire au moins une lettre afin de pouvoir retourner dans q_0 (en écrivant alors $\bot$).

Ainsi, en encodant 0 en $\bot 1311322$ et 1 en $\top 1311322$, alors que l'on décode $w \cdot \bot$ en 0 et $w \cdot \top$ en 1, on peu transmettre n'importe quelle suite binaire. L'utilisation de la solution σ et du mot correspondant w fournit donc un canal pour $\mathscr{M}_{\mathscr{J}}$.

Lemme 3.18. *Si $\mathscr{I}$ n'a pas de solution non triviale, alors $\mathscr{M}_{\mathscr{J}}$ n'a pas de canal.*

Démonstration. Si $\mathscr{I}$ n'a pas de solution non triviale, alors pour tout bit $b \in \mathbb{B}$ et toute suite $\sigma \in N^*$, $\mathscr{M}_{\mathscr{J}}(b \cdot \sigma) = A^+ \cdot \mathbb{B}$. Supposons que $\mathscr{M}_{\mathscr{J}}$ contienne un canal. Par le théorème 3.7, on peut supposer qu'une paire $(\mathscr{E}_{\mathrm{C}}, \mathscr{D}_{\mathrm{C}})$ constituée d'un encodeur et d'un décodeur canoniques, où $\mathscr{E}_{\mathrm{C}}$ est paramétré par (h, h_0, h_1, h), est un canal pour $\mathscr{M}_{\mathscr{J}}$.

Considérons les mots $\beta = 01$ et $\beta' = 10$. Leurs images respectives par $\mathscr{E}_{\mathrm{C}}$ sont $u = h \cdot h_0 \cdot h_1 \cdot h'$ et $u' = h \cdot h_1 \cdot h_0 \cdot h'$. Puisque u et u' ont au moins une image par $\mathscr{M}_{\mathscr{J}}$, ce sont des mots de $(\mathbb{B} \cdot N^*)^*$, et on peut les réécrire

$$u = \overbrace{b_0 \cdot \sigma_0 \cdots b_i \cdot \sigma_i}^{h} \cdot \overbrace{\sigma'_i \cdot b_{i+1} \cdots b_j \cdot \sigma_j}^{h_0} \cdot \overbrace{\sigma'_j \cdot b_{j+1} \cdots b_k \cdot \sigma_k}^{h_1} \cdot \overbrace{\sigma'_k \cdot b_{k+1} \cdots b_n \cdot \sigma_n}^{h'}$$
$$u' = \overbrace{b_0 \cdot \sigma_0 \cdots b_i \cdot \sigma_i}^{h} \cdot \overbrace{\sigma'_j \cdot b_{j+1} \cdots b_k \cdot \sigma_k}^{h_1} \cdot \overbrace{\sigma'_i \cdot b_{i+1} \cdots b_j \cdot \sigma_j}^{h_0} \cdot \overbrace{\sigma'_k \cdot b_{k+1} \cdots b_n \cdot \sigma_n}^{h'}$$

Ainsi, par $\mathscr{M}_{\mathscr{J}}$ les images de u et u' sont tous deux le langage

$$\mathscr{M}_{\mathscr{J}}(u) = \mathscr{M}_{\mathscr{J}}(u') = \left(A^+ \cdot \mathbb{B}\right)^n$$

Le décodage par $\mathscr{D}_{\mathrm{C}}$ est donc fait pareillement pour u et u'. Ainsi la paire $(\mathscr{E}_{\mathrm{C}}, \mathscr{D}_{\mathrm{C}})$ n'est pas un canal pour $\mathscr{M}_{\mathscr{J}}$, ce qui est contradictoire. □

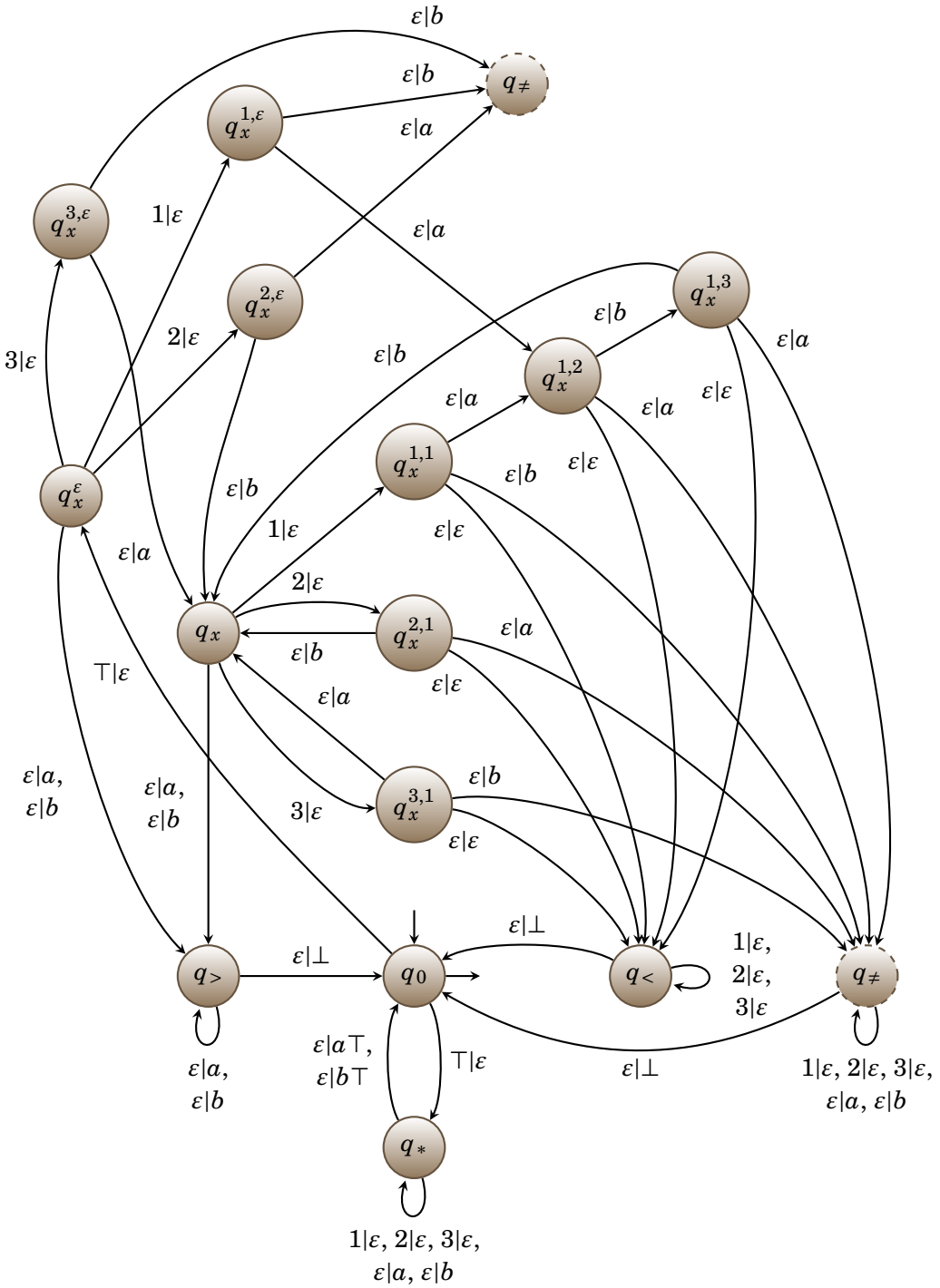


FIGURE 3.11: Une partie de la $\top$-moitié du transducteur $\mathscr{A}_{\mathscr{J}}$ qui code l'instance $\mathscr{J}$ du problème de Post. L'état $q_{\neq}$ est dupliqué par souci de clarté.

3.4 Canaux et propriétés de sécurité

3.4.1 Canaux et non interférence : deux propriétés orthogonales

Le modèle des canaux ici présenté pose des contraintes assez fortes en terme de sécurité – quoiqu'il soit souvent possible de retourner à ce cas. Cette recherche de la sécurité du système par l'absence de canal est à rapprocher de la notion bien connue de *non interférence*.

Tout comme la non interférence, l'absence de canal donne des garanties sur le confinement d'information au sein d'une zone protégée. Un système qui n'est pas k-non interférent pour tout entier k poserait, selon l'intuition, un grand problème de sécurité. Cependant, cela n'implique pas l'existence d'un canal tel que défini plus haut.

Afin de pouvoir réellement comparer les deux notions, il faut tout d'abord traduire l'automate sur $H \uplus L$ en un transducteur. Cela est fait syntaxiquement en remplaçant chaque transition $q \xrightarrow{h} q'$ en une transition $q \xrightarrow{h|\varepsilon} q'$ si $h \in H$ et $q \xrightarrow{\ell} q'$ en une transition $q \xrightarrow{\varepsilon|\ell} q'$ si $\ell \in L$. Pour $\mathscr{A}$ un automate sur $H \uplus L$, on note $trans(\mathscr{A})$ le transducteur obtenu par la transformation ci-dessus.

Nous pouvons alors prouver que la propriété de canal ne se résume pas à une propriété de k-non interférence.

Proposition 3.19. *Il existe un automate $\mathscr{A}$ qui n'est pas k-non interférent pour tout entier k et tel que $trans(\mathscr{A})$ ne contient pas de canal.*

Démonstration. Considérons l'automate $\mathscr{A}_{\text{int}}$ sur $H \uplus L$ de la figure 3.12, où $H = \{h\}$ et $L = \{\ell\}$. Soit $k > 0$ un entier et $v = (h \cdot \ell)^k$. La projection de v sur l'alphabet L est

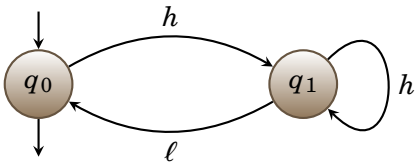


FIGURE 3.12: Un automate qui n'est pas k-non interférent pour tout k et dont le transducteur associé ne contient pas de canal.

$\pi_L(v) = \ell^k$. Soit $w \in \mathscr{L}(\mathscr{A}_{\text{int}})$ tel que $\pi_L(w) = \pi_L(v) = \ell^k$. Une exécution de $\mathscr{A}_{\text{int}}$ qui accepte w prend au moins k fois la transition $q_1 \xrightarrow{\ell} q_0$, et donc au moins k fois la transition $q_0 \xrightarrow{h} q_1$. Ainsi $w \in (L^* \cdot H)^k \cdot A^*$, c'est-à -dire $w \notin \bigcup_{i=0}^{k-1} (L^* \cdot H)^i \cdot L^*$. Donc $\mathscr{A}_{\text{int}}$ n'est pas k-non interférent.

Le transducteur associé $trans(\mathscr{A}_{\text{int}})$ ne contient pas d'état encodant. En effet, il est impossible de trouver deux mots de $H^* = h^*$ qui ne commutent pas, et donc qui forment un code. Donc la condition nécessaire (voir théorème 3.5) d'existence d'un canal n'est pas vérifiée et $trans(\mathscr{A}_{\text{int}})$ ne contient pas de canal. □

Cependant, la propriété de canal n'est pas strictement plus forte que la propriété d'interférence :

Proposition 3.20. *Il existe un automate $\mathscr{A}$ 1-non interférent tel que $trans(\mathscr{A})$ contient un canal.*

Démonstration. Soit $\mathscr{A}_{cc}$ l'automate sur $A = H \uplus L$ de la figure 3.13, où H est l'ensemble des lettres[3] $\{h, h_0, h_1, h'\}$ et $L = \{\ell, \ell_0, \ell_1, \ell'\}$. Tout mot qui contient une lettre de h est accepté par une exécution commençant par $q_i \xrightarrow{h} q_h \xrightarrow{\ell}$, donc sa projection sur L commence nécessairement par ℓ. Étant donné que tout mot de $\ell \cdot L^*$ est accepté par $\mathscr{A}_{cc}$ (en bouclant dans l'état q_ℓ), pour tout mot w de $\mathscr{L}(\mathscr{A}_{cc})$ contenant un h, $\pi_L(w)$ est aussi dans le langage $\mathscr{L}(\mathscr{A}_{cc})$. Ainsi $\mathscr{A}_{cc}$ est non interférent.

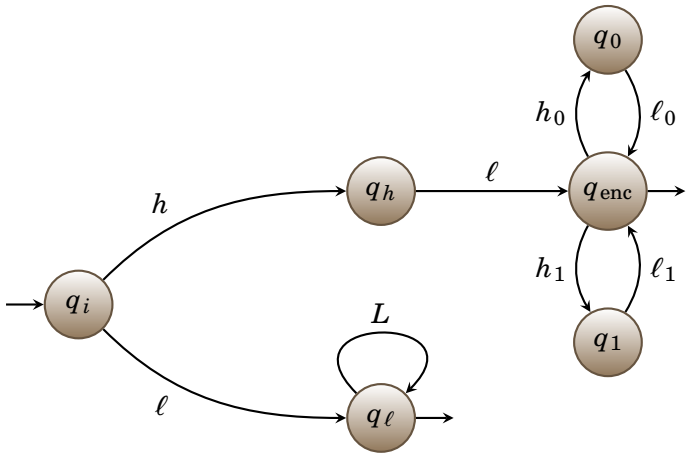


FIGURE 3.13: Automate 1-non interférent dont le transducteur associé contient un canal.

Considérons les relations $\mathscr{E}$ et $\mathscr{D}$ suivantes :

$$\mathscr{E} = (\varepsilon|h) \cdot \{(0|h_0), (1, h_1)\}^* \quad \text{et} \quad \mathscr{D} = (\ell|\varepsilon) \cdot \{(\ell_0|0), (\ell_1|1)\}^*.$$

Il est clair que tout mot binaire est exactement transmis par $\mathscr{E} \cdot trans(\mathscr{A}_{cc}) \cdot \mathscr{D}$. En effet, l'état q_ℓ qui pourrait introduire des erreurs dans ce que lit $\mathscr{D}$ n'est jamais visité. Donc $trans(\mathscr{A}_{cc})$ contient un canal. □

Ainsi les interférences, même de manière répétée, ne sont pas comparables à un canal. Il faut remarquer que ce fait n'est pas dû à une absurdité du modèle, puisque Hélouët et Roumy [HR10] obtiennent des résultats similaires pour des canaux modélisés dans le cadre de la théorie de l'information.

3.4.2 Mesure de la sécurité

Dans ce chapitre nous avons présenté un modèle de canal dans des transducteurs. Cette modélisation requiert une propriété assez forte : que la transmission soit parfaite. Cependant nous avons vu qu'une certaine quantité d'erreur pouvait être éliminée. Ainsi, un canal n'étant pas totalement parfait pourrait être transformé en un canal parfait.

Néanmoins, la redondance nécessaire à rendre parfait un canal qui ne l'est pas le rend moins « efficace ». Malheureusement, aucune notion d'efficacité n'est définie

3. Ici, h, ℓ, etc sont des lettres et non des mots.

dans le modèle ici présenté. Les canaux sont donc totalement qualitatifs. On pourrait envisager de définir des mesures de capacité de canal. Le but serait alors de s'approcher de ce que Moskovitz et Kang appellent le *small message criterion* [MK94], ou une mesure de *bit-rate* à la manière de Asarin et Dima [AD10]. Cependant dans le premier cas cela impose une mesure du *temps* de l'exécution, et dans le second cas la mesure se base sur le taux d'erreur, ce qui n'est pas considéré ici, puisque le canal est supposé parfait.

D'autres critères peuvent cependant être envisagés, tels que la longueur du message à la sortie de l'encodeur (ou à l'entrée du décodeur). Ainsi l'élimination d'un retard k réduit d'autant l'efficacité du canal. Une telle mesure dépend alors du canal $(\mathscr{E},\mathscr{D})$ choisi, et il est pour l'instant difficile de savoir lorsqu'un canal est optimal.

C'est pourtant une mesure de l'efficacité d'un canal optimal qui permettrait de donner une idée au concepteur du modèle du danger encouru (en terme de sécurité des données) par la mise en place du système à l'étude. Ceci est d'autant plus vrai que les solutions habituelles pour contrer un canal découvert impliquent de « brouiller » le message, souvent au prix d'une moindre efficacité du système. Un compromis entre la perte d'efficacité d'un système et sa sécurité serait plus aisé à trouver en connaissant une mesure de cette dernière.

Cependant les mesures de capacité de canal se basent bien souvent sur la théorie de l'information, et donc sur des modèles probabilistes. Les modèles de transducteurs auxquels sont ajoutées des probabilités souffrent malheureusement de l'indécidabilité de problèmes très simple, tels que le calcul de la sortie la plus probable en fonction d'une entrée, dès que le système contient du non déterminisme [BC03, Moh09]. Dans le chapitre suivant, on simplifie donc le modèle afin d'omettre les informations d'entrée/sortie propres aux transducteurs et on se place dans le cadre des automates purement probabilistes (c'est-à-dire sans non déterminisme). Cela permet d'évaluer des propriétés plus complexes que la transmission parfaite d'information à travers le concept très général de l'opacité.

Chapitre 4

Mesures d'opacité

Οἷον πασῶν που τεχνῶν ἄν τις ἀριθμητικὴν χωρίζῃ καὶ μετρητικὴν καὶ στατικήν, ὡς ἔπος εἰπεῖν φαῦλον τὸ καταλειπόμενον ἑκάστης ἂν γίγνοιτο. [1]

Platon, *Philèbe*.

Afin de donner une mesure à des propriétés de sécurité, ce chapitre définit plusieurs versions quantitatives de propriétés d'opacité.

Mesurer l'opacité permet au concepteur du modèle d'évaluer la sécurité du système, et surtout de comparer différentes versions d'un même système. Deux types de mesures duales peuvent être définies, répondant aux questions suivantes.

- Si la propriété d'opacité n'est pas vérifiée, dans quelle mesure cela impacte-il la sécurité ?
- Même si la propriété d'opacité est vérifiée, que peut apprendre l'observateur ?

Ici, les mesures d'opacité se basent sur des mesures de probabilités sur les comportements du système, qui est modélisé par un automate purement probabiliste.

On étend ainsi le concept d'opacité aux systèmes probabilistes, afin de mesurer soit la taille de la fuite d'information soit la robustesse de l'opacité. Dans les deux cas, le choix est fait de prendre 0 comme frontière entre l'opacité et la non opacité : si le système est opaque, la mesure de la fuite vaut 0. À l'inverse, si le système n'est pas opaque, c'est la mesure de robustesse qui vaut 0.

Ces mesures sont instanciées afin de mesurer des propriétés de sécurité telles que la non interférence et l'anonymat.

Lorsque les fonctions d'observations et le secret sont réguliers, on montre qu'il est possible de calculer automatiquement les valeurs des mesures d'opacité.

Les mesures se comportant différemment en fonction du type de prédicat qui doit être gardé secret, leur pertinence est comparée tout d'abord de manière purement abstraite, puis dans le cadre d'une étude de cas du protocole *Crowds*.

Enfin, on introduit le non déterminisme dans le modèle afin de modéliser un complice de l'observateur à l'intérieur du système.

1. « Par exemple, si on sépare de tous les arts l'art de compter, de mesurer, de peser, on peut dire que ce qui restera de chacun d'eux n'aura pas grande valeur. » (trad. Émile CHAMBRY).

4.1 Notations

4.1.1 Opacité asymétrique et opacité symétrique

Lorsque l'on cherche à garder une information secrète, telle que l'occurrence d'une certaine action dans le système, la sécurité de l'information est tout autant compromise par un observateur capable de savoir que l'action *s'est* produite que par un observateur capable de savoir que l'action *ne s'est pas* produite. Plus généralement, il est parfois aussi problématique pour la confidentialité qu'un observateur puisse déduire qu'une propriété est fausse, que s'il peut déduire si elle est vraie. La définition classique de l'opacité ne concerne que la possibilité pour un observateur de savoir que la propriété était vraie. Il est alors possible de renforcer les exigences de sécurité : le prédicat φ que l'on souhaite garder secret ainsi que sa négation $\overline{\varphi}$ doivent tous les deux être opaques.

Dans le but de donner une seule mesure de la sécurité qui engloberait le degré d'opacité de φ et de $\overline{\varphi}$, on définit l'*opacité symétrique* comme l'opacité de φ et de $\overline{\varphi}$: le prédicat φ est symétriquement opaque si pour toute observation $o \in Obs$, $\mathcal{O}^{-1}(o) \not\subseteq \varphi$ et $\mathcal{O}^{-1}(o) \not\subseteq \overline{\varphi}$. L'opacité de φ est donc aussi appelée *opacité asymétrique*.

4.1.2 Prédicats, observation, et variables aléatoires

Dans la suite de ce chapitre, on considère des systèmes modélisés par des automates probabilistes. Rappelons qu'un automate probabiliste $\mathcal{A}$ définit une loi de probabilité sur l'ensemble de ses exécutions complètes : $\mathbf{P}_{\mathcal{A}} \in \mathbb{D}(TExec(\mathcal{A}))$. On note parfois cette loi $\mathbf{P}$, lorsqu'aucune ambiguïté n'est possible sur l'identité du système.

Ainsi une fonction d'observation $\mathcal{O} : TExec(\mathcal{A}) \to Obs$ peut être vue comme une variable aléatoire. De même, un prédicat $\varphi \subseteq TExec(\mathcal{A})$ est aussi une fonction $\varphi : TExec(\mathcal{A}) \to \mathbb{B}$, donc une variable aléatoire. Afin de limiter la lourdeur des notations, on note l'évènement $[\varphi = 1]$ simplement $[\varphi]$ et $[\varphi = 0]$ simplement $[\overline{\varphi}]$ lorsque la confusion entre la variable aléatoire et l'évènement est impossible.

L'observation comme le prédicat portent sur des exécutions totales, donc se terminant par $\checkmark$. On les utilise de la même manière sur la version non-probabiliste de l'automate. Par exemple, on écrit abusivement $\mathcal{O}(\rho)$ pour $\mathcal{O}(\rho\checkmark)$.

Enfin, on suppose dans la suite que les fonctions d'observation sont surjectives.

Par exemple, considérons l'automate probabiliste $\mathcal{A}_2$ reproduit figure 4.1. Soit $\mathcal{O}_b$ la fonction d'observation qui projette l'exécution sur la lettre b et φ_a le prédicat vrai lorsqu'il y a au moins un a dans l'exécution. On a ici, entre autres,

$$\mathbf{P}_{\mathcal{A}_2}(\varphi_a) = \frac{1}{2} \qquad \mathbf{P}_{\mathcal{A}_2}(\mathcal{O}_b = b) = \frac{1}{2} \qquad \mathbf{P}_{\mathcal{A}_2}(\mathcal{O}_b = \varepsilon) = \frac{1}{2} \qquad \mathbf{P}_{\mathcal{A}_2}(\varphi_a, \mathcal{O}_b = b) = \frac{1}{4}.$$

4.2 Mesure de la faille de sécurité

La première quantité qui peut être mesurée est donc la taille de la faille de sécurité détectée par la violation de l'opacité. Ainsi, l'opacité est violée lorsqu'une classe d'observation est complètement incluse dans le prédicat (ou dans le complémentaire du prédicat, dans le cas symétrique). On peut relaxer cette condition en mesurant

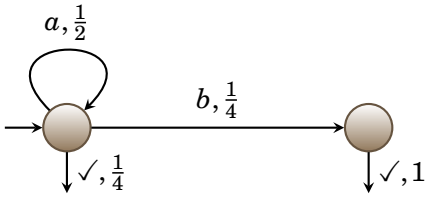


FIGURE 4.1: L'automate probabiliste $\mathscr{A}_2$.

les classes d'observation qui permettent à un observateur de connaître la valeur du prédicat. Plus précisément, on mesure la probabilité pour une exécution tirée aléatoirement (selon la distribution définie par l'automate $\mathscr{A}$) que l'observation trahisse complètement l'appartenance (ou non) au prédicat φ.

Définition 4.1 (Opacité probabiliste faible)**.** *Soit $\mathscr{A}$ un automate probabiliste, $\mathscr{O}$: $TExec(\mathscr{A}) \to Obs$ une fonction d'observation, et $\varphi \subseteq TExec(\mathscr{A})$ un prédicat. L'*opacité (asymétrique) probabiliste faible *(LPO [2]) de φ sur $\mathscr{A}$ relativement à $\mathscr{O}$ est donnée par :*

$$PO^{A}_{\ell}(\mathscr{A}, \varphi, \mathscr{O}) = \sum_{\substack{o \in Obs \\ \mathscr{O}^{-1}(o) \subseteq \varphi}} \mathbf{P}(\mathscr{O} = o)$$

*L'*opacité symétrique probabiliste faible *(LPSO [3]) de φ sur $\mathscr{A}$ relativement à $\mathscr{O}$ est donnée par :*

$$PO^{S}_{\ell}(\mathscr{A}, \varphi, \mathscr{O}) = \sum_{\substack{o \in Obs \\ \mathscr{O}^{-1}(o) \subseteq \varphi}} \mathbf{P}(\mathscr{O} = o) + \sum_{\substack{o \in Obs \\ \mathscr{O}^{-1}(o) \subseteq \overline{\varphi}}} \mathbf{P}(\mathscr{O} = o)$$

Comme on le montre dans la suite, ces valeurs sont nulles lorsque le système est – en dehors des lois de probabilités qui régissent son fonctionnement – opaque au sens classique du terme, donc peut être considéré comme sûr.

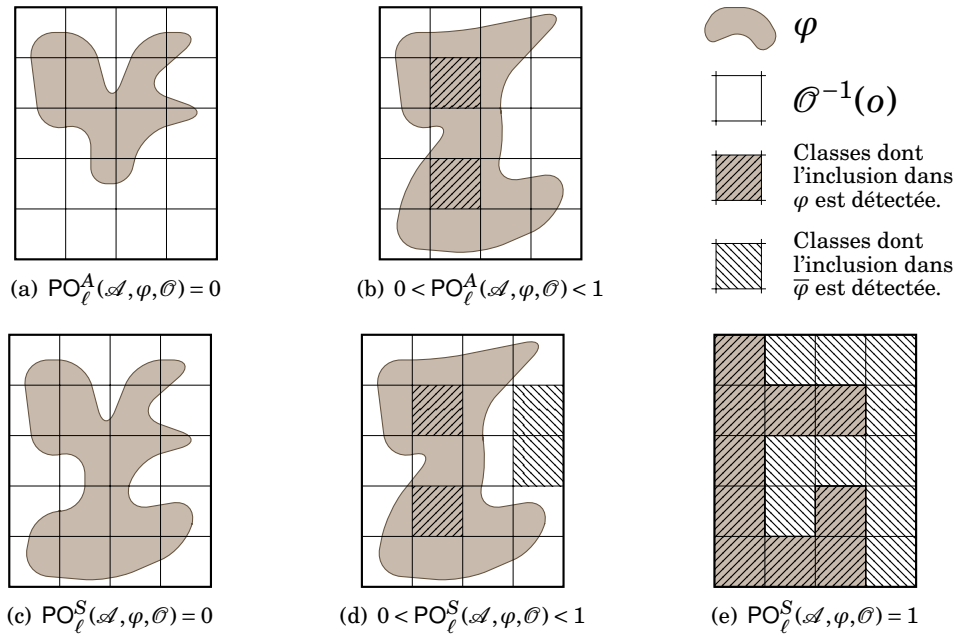
Dans le cas de la LPO, cela correspond au cas où chaque classe d'observation contient au moins une exécution dans $\overline{\varphi}$, comme sur la figure 4.2(a). En effet, puisque la LPO ne mesure que la probabilité des classes dont l'inclusion dans φ est révélée, celles pour lesquelles c'est l'inclusion dans $\overline{\varphi}$ qui est connue ne sont pas prises en compte. À l'inverse, l'autre valeur extrémale $\mathsf{PO}^{A}_{\ell}(\mathscr{A}, \varphi, \mathscr{O}) = 1$ est atteinte lorsque le prédicat φ est toujours vrai.

Lorsque la LPSO est nulle, cela signifie que chaque classe d'observation contient à la fois des exécutions dans φ et d'autres dans $\overline{\varphi}$, comme sur la figure 4.2(c). À l'opposé, le système est totalement non sûr lorsque l'observation permet de connaître la valeur de vérité de φ. Dans ce cas, φ est une union de classes d'observation comme sur la figure 4.2(e). Ceci peut aussi être interprété en terme de théorie de l'information : connaître $\mathscr{O}$ donne toute l'information sur φ. Enfin, dans le cas intermédiaire, certaines classes d'observations, mais pas toutes, ne contiennent que des exécutions dans φ ou seulement des exécution dans $\overline{\varphi}$, comme sur la figure 4.2(d).

Ces propriétés sont formalisées dans les propositions suivantes :

Proposition 4.1.

2. Pour *Liberal Probabilistic Opacity*.
3. Pour *Liberal Probabilistic Symmetrical Opacity*.



(a) $\mathsf{PO}^{A}_{\ell}(\mathscr{A},\varphi,\mathcal{O}) = 0$ (b) $0 < \mathsf{PO}^{A}_{\ell}(\mathscr{A},\varphi,\mathcal{O}) < 1$

(c) $\mathsf{PO}^{S}_{\ell}(\mathscr{A},\varphi,\mathcal{O}) = 0$ (d) $0 < \mathsf{PO}^{S}_{\ell}(\mathscr{A},\varphi,\mathcal{O}) < 1$ (e) $\mathsf{PO}^{S}_{\ell}(\mathscr{A},\varphi,\mathcal{O}) = 1$

FIGURE 4.2: Opacité (asymétrique et symétrique) probabiliste faible.

(1) $0 \leq PO^{A}_{\ell}(\mathscr{A},\varphi,\mathcal{O}) \leq 1$
(2) $PO^{A}_{\ell}(\mathscr{A},\varphi,\mathcal{O}) = 0$ *si et seulement si* φ *est asymétriquement opaque sur* $Unprob(\mathscr{A})$ *relativement à* $\mathcal{O}$.
(3) $PO^{A}_{\ell}(\mathscr{A},\varphi,\mathcal{O}) = 1$ *si et seulement si* $\varphi = TExec(\mathscr{A})$.

Proposition 4.2.
(1) $0 \leq PO^{S}_{\ell}(\mathscr{A},\varphi,\mathcal{O}) \leq 1$
(2) $PO^{S}_{\ell}(\mathscr{A},\varphi,\mathcal{O}) = 0$ *si et seulement si* φ *est symétriquement opaque sur* $Unprob(\mathscr{A})$ *relativement à* $\mathcal{O}$.
(3) $PO^{S}_{\ell}(\mathscr{A},\varphi,\mathcal{O}) = 1$ *si et seulement si* $H(\varphi|\mathcal{O}) = 0$.

Démonstration des propositions 4.1 et 4.2.

(1) Les évènements considérés étant mutuellement exclusifs, la somme de leur probabilités ne dépasse jamais 1.

(2) Tout d'abord, remarquons que l'exécution totale $\rho\checkmark$ a une probabilité non nulle dans $\mathscr{A}$ si et seulement si c'est une exécution acceptante dans $Unprob(\mathscr{A})$.

Supposons que $\mathsf{PO}^{A}_{\ell}(\mathscr{A},\varphi,\mathcal{O}) = 0$. Alors il n'y a pas d'observable $o \in Obs$ telle que $\mathbf{P}(\mathcal{O} = o) > 0$ et $\mathcal{O}^{-1}(o) \subseteq \varphi$. Donc pour toute observable o, $\mathcal{O}^{-1}(o) \not\subseteq \varphi$.

Réciproquement, si φ est opaque sur $Unprob(\mathscr{A})$, il n'y a pas d'observable o telle que $\mathcal{O}^{-1}(o) \subseteq \varphi$, donc $\mathsf{PO}^{A}_{\ell}(\mathscr{A},\varphi,\mathcal{O})$ est nulle.

Le cas de la LPSO est analogue, la seule différence étant qu'il faut aussi traiter le cas de $\overline{\varphi}$, dual à celui de φ.

(3) Dans le cas de la LPO, cela découle directement de la définition.

Pour la LPSO, $H(\varphi|\mathcal{O}) = 0$ si et seulement si

$$\sum_{\substack{o\in Obs \\ i\in \mathbb{B}}} \mathbf{P}(\varphi = i|\mathcal{O} = o)\cdot \log\big(\mathbf{P}(\varphi = i|\mathcal{O} = o)\big) = 0.$$

Puisque tous les termes de la somme ont le même signe (en l'occurrence négatif), la somme n'est nulle que si chaque terme l'est. En posant, pour $o \in Obs$,

$$f(o) = \mathbf{P}(\varphi|\mathscr{O} = o) = 1 - \mathbf{P}(\overline{\varphi}|\mathscr{O} = o),$$

on obtient $H(\varphi|\mathscr{O}) = 0$ si et seulement si

$$\forall o \in Obs,\ f(o)\cdot\log(f(o)) + (1-f(o))\cdot\log(1-f(o)) = 0.$$

Étant donné que l'équation $x\cdot\log(x) + (1-x)\cdot\log(1-x) = 0$ n'admet que 1 et 0 comme solutions, cela revient à dire que pour toute observable o, les exécutions (totales) ρ telles que $\mathscr{O}(\rho) = o$ sont soit toutes dans φ, soit toutes hors de φ. Donc $H(\varphi|\mathscr{O}) = 0$ si et seulement si pour toute observable o, soit $\mathscr{O}^{-1} \subseteq \varphi$ soit $\mathscr{O}^{-1} \subseteq \overline{\varphi}$, c'est-à-dire $\mathsf{PO}^S_\ell(\mathscr{A}, \varphi, \mathscr{O}) = \sum_{o\in Obs} \mathbf{P}(\mathscr{O} = o) = 1$. □

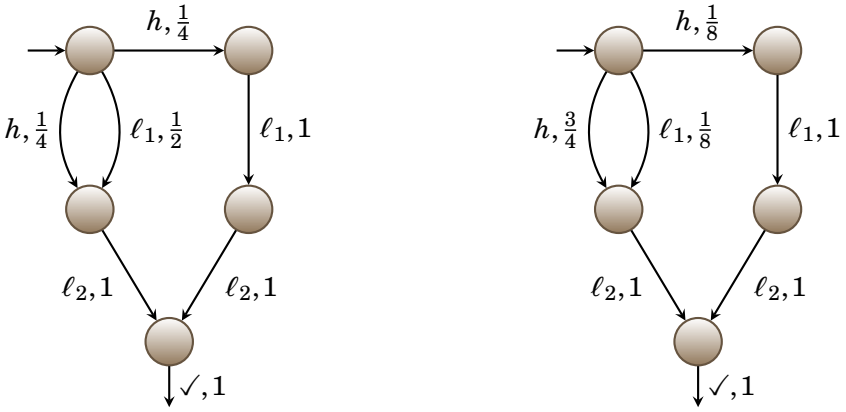
Exemple 4.1 (Non interférence)**.** On considère les systèmes $\mathscr{A}_5$ et $\mathscr{A}_6$ de la figure 4.3. Ces automates sont tous deux étiquetés par $A = \{h, \ell_1, \ell_2\}$. Soit φ_{NI} le prédicat correspondant à la 1-non interférence, où h est la seule action de haut niveau : $\varphi_{NI} = tr^{-1}(A^* h A^*)$. De même, la fonction d'observation est celle qui ne garde que la trace sur $\{\ell_1, \ell_2\}$: $\mathscr{O}_L = \pi_{\{\ell_1,\ell_2\}} \circ tr$.

Dans le cas non probabiliste, les systèmes $\mathscr{A}_5$ et $\mathscr{A}_6$ se comportent identiquement. Et dans les deux cas, la propriété de 1-non interférence n'est pas respectée car l'observation de ℓ_2 seul (c'est-à-dire qui n'est pas précédé d'un ℓ_1) trahit la présence d'une action h.

Cependant, si l'on prend en compte les probabilités de ces systèmes, on remarque que l'exécution $h\ell_2\checkmark$, qui pose un problème de sécurité, apparaît moins fréquemment dans $\mathscr{A}_5$ que dans $\mathscr{A}_6$. En effet, considèrons toutes les exécutions de ces automates, comme il est fait sur la table 4.1. L'ensemble des exécutions dont l'observation est $\ell_1\ell_2$, qui comprend les exécutions ρ_1 et ρ_2, intersecte à la fois φ_{NI} (avec l'exécution ρ_2) et $\overline{\varphi_{NI}}$ (avec ρ_1). *A contrario*, l'ensemble des exécutions d'observation ℓ_2, en réalité réduit au singleton ρ_3, est tout entier contenu dans φ_{NI}.

Le calcul de l'opacité probabiliste faible donne alors :

$$\mathsf{PO}^A_\ell\left(\mathscr{A}_5, \varphi_{NI}, \mathscr{O}_L\right) = \frac{1}{4} \qquad \mathsf{PO}^A_\ell\left(\mathscr{A}_6, \varphi_{NI}, \mathscr{O}_L\right) = \frac{3}{4}.$$



(a) Automate probabiliste $\mathscr{A}_5$ (b) Automate probabiliste $\mathscr{A}_6$

FIGURE 4.3: Automates probabilistes qui ne sont pas 1-non interférents.

$tr(\rho)$	$\mathbf{P}_{\mathscr{A}_5}(\rho)$	$\mathbf{P}_{\mathscr{A}_6}(\rho)$	φ_{NI}	$\mathscr{O}_L(\rho)$
$tr(\rho_1) = \ell_1\ell_2\checkmark$	1/2	1/8	0	$\ell_1\ell_2$
$tr(\rho_2) = h\ell_1\ell_2\checkmark$	1/4	1/8	1	$\ell_1\ell_2$
$tr(\rho_3) = h\ell_2\checkmark$	1/4	3/4	1	ℓ_2

TABLE 4.1: Exécutions totales de $\mathscr{A}_3$ et $\mathscr{A}_4$.

On peut en conclure que le système modélisé par $\mathscr{A}_5$ est plus sûr que celui modélisé par $\mathscr{A}_6$, car l'exécution qui transmet de l'information sur le prédicat arrive plus souvent.

Dans cet exemple, la LPO et la LPSO coincident, mais ce n'est évidemment pas toujours le cas. Même dans le cas non probabiliste, les notions asymétrique ou symétrique de l'opacité de φ_{NI} relativement à $\mathscr{O}_L$ expriment des interprétations légèrement différentes de la propriété « un observateur extérieur ne sait pas si l'action h a eu lieu ». Dans le cas asymétrique, cela correspond exactement à la *non interférence non déterministe forte* de [GM82], tandis que le cas symétrique est appelé *propriété de sécurité parfaite* dans [AČZ06].

4.3 Robustesse de l'opacité

Un point du vue plus paranoïaque peut être adopté quant à la mesure de l'opacité d'une propriété : quelle information transparaît du fonctionnement probabiliste du système ? Par exemple, sur la figure 4.2(c), bien que chaque classe d'observation intersecte à la fois φ et $\overline{\varphi}$ (et donc que le système soit opaque), certaines classes sont *presque* incluses dans φ (et d'autres dans $\overline{\varphi}$). À l'inverse, d'autres classes sont plus équilibrées.

Ainsi, pour chaque classe d'observation, on cherchera à mesurer si elle « viole presque » la condition d'opacité. Dans le cas asymétrique, cela revient à mesurer la probabilité que φ soit faux au sein de la classe. Dans le cas symétrique, cela revient à mesurer, toujours au sein de la classe d'observation, l'équilibre entre φ et $\overline{\varphi}$. Dans ce dernier cas, il est aussi possible de comparer l'équilibre entre φ et $\overline{\varphi}$ au sein de la classe d'observation avec l'équilibre global. Cela revient à mesurer l'information transmise par le biais de l'observation.

4.3.1 Cas de l'opacité asymétrique

Dans le cas de l'opacité asymétrique, l'opacité est plus robuste lorsque au sein de chaque classe d'observation, φ a peu de chances d'être vérifiée. Dans le cas extrême, φ est le prédicat vide (c'est-à-dire toujours faux), et le système est toujours opaque, même pour la fonction d'observation Id, qui observe tout. Ceci mesure donc, intuitivement, lorsque l'on a une exécution du système qui vérifie φ, la difficulté d'obtenir (par tir aléatoire) une exécution observée pareillement mais n'étant pas dans φ.

La valeur définie ici donne une mesure globale de la robustesse de chaque classe. Elle donne plus de poids aux classes les moins robustes, de manière à implémenter le principe bien connu que la sécurité d'un système se mesure par celle de son point

faible. Ainsi, dans le cas où une classe d'observation est entièrement incluse dans φ, donc lorsque le système n'est pas opaque, la robustesse est nulle.

Définition 4.2 (Opacité asymétrique probabiliste robuste)**.** *Soit $\mathscr{A}$ un automate probabiliste, $\mathscr{O} : TExec(\mathscr{A}) \to Obs$ une fonction d'observation et $\varphi \subseteq TExec(\mathscr{A})$ un prédicat. L'*opacité (asymétrique) probabiliste robuste *(RPO [4]) de φ sur $\mathscr{A}$ relativement à $\mathscr{O}$ est définie par :*

$$\frac{1}{PO_r^A(\mathscr{A},\varphi,\mathscr{O})} = \sum_{o \in Obs} \mathbf{P}(\mathscr{O} = o) \cdot \frac{1}{\mathbf{P}(\overline{\varphi} | \mathscr{O} = o)}.$$

La RPO est donc la moyenne harmonique, pondérée par la « taille » de chaque classe d'observation, de la probabilité que φ soit fausse dans cette classe. Lors du calcul de cette moyenne, les valeurs nulles pour $\mathbf{P}(\overline{\varphi} | \mathscr{O} = o)$ font tendre l'inverse de la RPO vers $+\infty$ (et donc la RPO vers 0), bien que ces limites ne soient pas notées explicitement.

Notons que cette mesure de la robustesse de l'opacité n'a de sens que sur des systèmes opaques au sens classique du terme. En effet, sur des systèmes non opaques, cette robustesse est nulle et la mesure de la faille de sécurité présentée à la section précédente est plus pertinente.

Les classes où φ est toujours vraie feront donc tendre l'inverse de la RPO vers l'infini, et donc la RPO vers 0. Pour que la RPO soit 1, au contraire, il faut que chaque valeur de la moyenne soit 1, et donc que φ soit toujours fausse.

Proposition 4.3.
(1) $0 \leq PO_r^A(\mathscr{A},\varphi,\mathscr{O}) \leq 1$
(2) $PO_r^A(\mathscr{A},\varphi,\mathscr{O}) = 0$ *si et seulement si φ n'est pas opaque sur $\mathscr{A}$ relativement à $\mathscr{O}$.*
(3) $PO_r^A(\mathscr{A},\varphi,\mathscr{O}) = 1$ *si et seulement si $\varphi = \emptyset$.*

Exemple : un système de carte bancaire

On considère le système de carte bancaire dans un magasin. Lorsqu'une carte est insérée, un montant x à payer est entré par le vendeur, et le client rentre alors son code [5] ; toutes ces actions sont abstraites par l'étiquette $Buy(x)$. Le montant de la transaction est donné par une probabilité qui abstrait des données statistiques sur les montants de telles transactions. Une fois le code rentré, le système peut soit accepter directement (*Accept*), soit interroger le serveur de la banque émettrice (*Call*) afin de vérifier la validité de la carte (et l'approvisionnement du compte en banque sous-jacent). La vérification peut ensuite résulter en l'acceptation ou le refus (*Reject*) de la transaction. De la même manière que pour le montant de l'achat, la probabilité de refus provient de statistiques, la décision de la banque étant *a priori* déterministe.

La vérification de la carte ne se fait pas toujours afin de limiter les coûts en terme de bande-passante et d'utilisation des serveurs de la banque. Cependant, une carte invalide ou un client insolvable représentent aussi un coût pour la banque, qui varie avec le montant de la transaction. Ainsi la probabilité de vérifier la carte varie en fonction du montant de l'achat.

L'automate probabiliste $\mathscr{A}_{\text{CB}}$ modélisant ce système est représenté à la figure 4.4.

4. Pour *Restrictive Probabilistic Opacity*.
5. On suppose ici que le code est correct, et les cas d'erreurs ne sont pas modélisés.

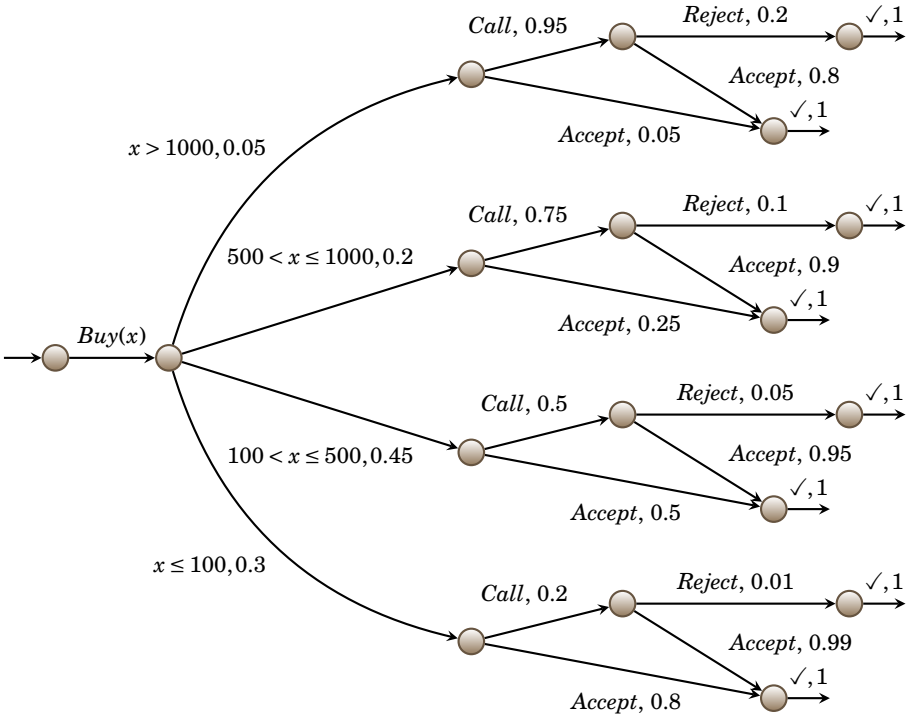


FIGURE 4.4: Le système de carte bancaire $\mathscr{A}_{\text{CB}}$.

On suppose maintenant qu'un observateur extérieur est capable de savoir si la carte a été vérifiée ou non. En pratique, cela est possible en mesurant le temps que prend la transaction à s'effectuer (entre le moment où le code a été rentré et celui où la transaction s'est terminée). Une vérification de la carte prenant du temps, si la transaction est longue, c'est que la banque a dû être interrogée. Un autre moyen d'observer si une vérification a eu lieu est une observation fine du réseau : un trafic plus intense trahit une interrogation des serveurs de la banque.

Le but de l'attaquant est de découvrir si le montant de l'achat dépasse les 500€. La RPO permet d'évaluer la difficulté qu'a un attaquant à découvrir cette information.

Formellement, la fonction d'observation $\mathscr{O}_{\text{Vérif}}$ est la projection de la trace de l'exécution sur $\{Call\}$. Les deux observables possibles sont donc $Call$ et ε. On note A l'alphabet de $\mathscr{A}_{\text{CB}}$. Le prédicat devant rester secret est donc

$$\varphi_{>500} \;=\; tr^{-1}\left(A^*(\text{“}500 < x \leq 1000\text{”} + \text{“}x > 1000\text{”})A^*\right).$$

Dans la suite, on considère les évènements correspondant à la présence de l'action étiquetée par l'intervalle contenant la valeur de x. Par exemple $\varphi_{>500}$ correspond à la disjonction des évènements $[500 < x \leq 1000]$ et $[x > 1000]$. La définition de la RPO devient dans ce cas :

$$\frac{1}{\mathsf{PO}_r^A\left(\mathscr{A}_{\text{CB}}, \varphi_{>500}, \mathscr{O}_{\text{Vérif}}\right)} = \frac{\mathbf{P}(\mathscr{O}_{\text{Vérif}} = \varepsilon)}{\mathbf{P}\left(\overline{\varphi_{>500}} | \mathscr{O}_{\text{Vérif}} = \varepsilon\right)} + \frac{\mathbf{P}(\mathscr{O}_{\text{Vérif}} = Call)}{\mathbf{P}\left(\overline{\varphi_{>500}} | \mathscr{O}_{\text{Vérif}} = Call\right)}$$

Le calcul de la PO_r^A, détaillé sur la table 4.2, donne :

$$\mathsf{PO}_r^A\left(\mathscr{A}_{\text{CB}}, \varphi_{>500}, \mathscr{O}_{\text{Vérif}}\right) \;=\; \frac{28272}{39377} \;\simeq\; 0.718.$$

$$
\begin{array}{rcl}
\mathbf{P}(\mathscr{O}_{\text{Vérif}} = \varepsilon) & = & \mathbf{P}(\mathscr{O}_{\text{Vérif}} = \varepsilon, x > 1000) + \mathbf{P}(\mathscr{O}_{\text{Vérif}} = \varepsilon, 500 < x \leq 1000) \\
& & + \mathbf{P}(\mathscr{O}_{\text{Vérif}} = \varepsilon, 100 < x \leq 500) + \mathbf{P}(\mathscr{O}_{\text{Vérif}} = \varepsilon, x \leq 100) \\
& = & \mathbf{P}(\mathscr{O}_{\text{Vérif}} = \varepsilon) \cdot \mathbf{P}(x > 1000) \\
& & + \mathbf{P}(\mathscr{O}_{\text{Vérif}} = \varepsilon) \cdot \mathbf{P}(500 < x \leq 1000) \\
& & + \mathbf{P}(\mathscr{O}_{\text{Vérif}} = \varepsilon) \cdot \mathbf{P}(100 < x \leq 500) \\
& & + \mathbf{P}(\mathscr{O}_{\text{Vérif}} = \varepsilon) \cdot \mathbf{P}(x \leq 100) \\
& = & 0.05 \cdot 0.05 + 0.25 \cdot 0.2 + 0.5 \cdot 0.45 + 0.8 \cdot 0.3 \\
\mathbf{P}(\mathscr{O}_{\text{Vérif}} = \varepsilon) & = & 0.5175
\end{array}
$$

$$
\mathbf{P}(\mathscr{O}_{\text{Vérif}} = Call) = 1 - \mathbf{P}(\mathscr{O}_{\text{Vérif}} = \varepsilon) = 0.4825
$$

$$
\begin{array}{rcl}
\mathbf{P}(\overline{\varphi_{>500}} | \mathscr{O}_{\text{Vérif}} = \varepsilon) & = & \dfrac{\mathbf{P}(\overline{\varphi_{>500}}, \mathscr{O}_{\text{Vérif}} = \varepsilon)}{\mathbf{P}(\mathscr{O}_{\text{Vérif}} = \varepsilon)} \\
& = & \dfrac{\mathbf{P}(x \leq 100, \mathscr{O}_{\text{Vérif}} = \varepsilon) + \mathbf{P}(100 < x \leq 500, \mathscr{O}_{\text{Vérif}} = \varepsilon)}{\mathbf{P}(\mathscr{O}_{\text{Vérif}} = \varepsilon)} \\
& = & \dfrac{\mathbf{P}(\mathscr{O}_{\text{Vérif}} = \varepsilon | x \leq 100) \cdot \mathbf{P}(x \leq 100)}{\mathbf{P}(\mathscr{O}_{\text{Vérif}} = \varepsilon)} \\
& & + \dfrac{\mathbf{P}(\mathscr{O}_{\text{Vérif}} = \varepsilon | 100 < x \leq 500) \cdot \mathbf{P}(100 < x \leq 500)}{\mathbf{P}(\mathscr{O}_{\text{Vérif}} = \varepsilon)} \\
& = & \dfrac{0.8 \cdot 0.3 + 0.5 \cdot 0.45}{0.5175} \\
\mathbf{P}(\overline{\varphi_{>500}} | \mathscr{O}_{\text{Vérif}} = \varepsilon) & = & \dfrac{0.465}{0.5175} \simeq 0.899
\end{array}
$$

$$
\begin{array}{rcl}
\mathbf{P}(\overline{\varphi_{>500}} | \mathscr{O}_{\text{Vérif}} = Call) & = & \dfrac{\mathbf{P}(\overline{\varphi_{>500}}, \mathscr{O}_{\text{Vérif}} = Call)}{\mathbf{P}(\mathscr{O}_{\text{Vérif}} = Call)} \\
& = & \dfrac{\mathbf{P}(x \leq 100, \mathscr{O}_{\text{Vérif}} = Call) + \mathbf{P}(100 < x \leq 500, \mathscr{O}_{\text{Vérif}} = Call)}{\mathbf{P}(\mathscr{O}_{\text{Vérif}} = Call)} \\
& = & \dfrac{\mathbf{P}(\mathscr{O}_{\text{Vérif}} = Call | x \leq 100) \cdot \mathbf{P}(x \leq 100)}{\mathbf{P}(\mathscr{O}_{\text{Vérif}} = Call)} \\
& & + \dfrac{\mathbf{P}(\mathscr{O}_{\text{Vérif}} = Call | 100 < x \leq 500) \cdot \mathbf{P}(100 < x \leq 500)}{\mathbf{P}(\mathscr{O}_{\text{Vérif}} = Call)} \\
& = & \dfrac{0.2 \cdot 0.3 + 0.5 \cdot 0.45}{0.4825} \\
\mathbf{P}(\overline{\varphi_{>500}} | \mathscr{O}_{\text{Vérif}} = Call) & = & \dfrac{0.285}{0.4825} \simeq 0.591
\end{array}
$$

$$
\begin{array}{rcl}
\dfrac{1}{\mathsf{PO}_r^A(\mathscr{A}_{\text{CB}}, \varphi_{>500}, \mathscr{O}_{\text{Vérif}})} & = & 0.5175 \cdot \dfrac{0.5175}{0.465} + 0.4825 \cdot \dfrac{0.4825}{0.285} \\
\dfrac{1}{\mathsf{PO}_r^A(\mathscr{A}_{\text{CB}}, \varphi_{>500}, \mathscr{O}_{\text{Vérif}})} & = & \dfrac{39377}{28272}
\end{array}
$$

TABLE 4.2: Calcul de la RPO du système de carte bancaire.

On peut comparer cette valeur à celle obtenue lorsque l'observateur n'observe pas s'il y a eu communication avec la banque, mais le résultat final du paiement : l'acceptation ou le refus. On définit la fonction d'observation $\mathscr{O}_{\text{Rés}}$ par la projection de la trace sur l'ensemble d'actions $\{Accept, Reject\}$. Des calculs similaires donnent

$$\begin{aligned}\frac{1}{\mathsf{PO}_r^A\left(\mathscr{A}_{\text{CB}}, \varphi_{>500}, \mathscr{O}_{\text{Rés}}\right)} &= 0.03635 \cdot \frac{0.03635}{0.028525} + 0.96365 \cdot \frac{0.96365}{0.73815} \\ \frac{1}{\mathsf{PO}_r^A\left(\mathscr{A}_{\text{CB}}, \varphi_{>500}, \mathscr{O}_{\text{Rés}}\right)} &= \frac{62775445549}{48127380000}\end{aligned}$$

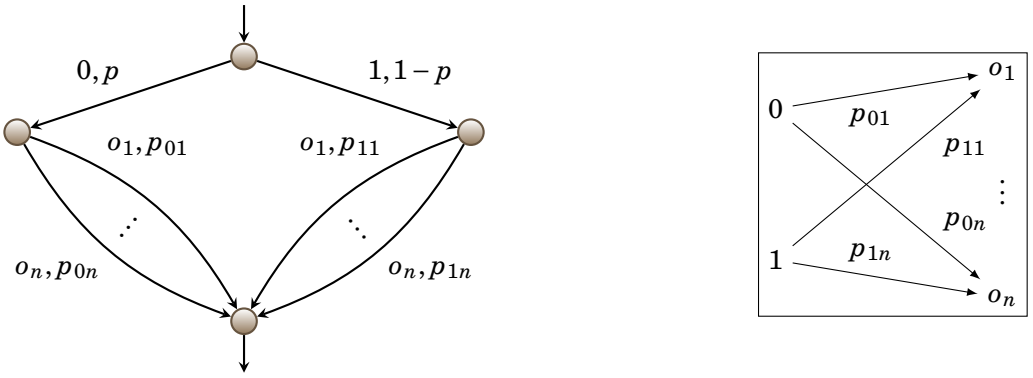
D'où

$$\mathsf{PO}_r^A\left(\mathscr{A}_{\text{CB}}, \varphi_{>500}, \mathscr{O}_{\text{Rés}}\right) = \frac{48127380000}{62775445549} \simeq 0.767.$$

Ainsi, l'observation de la vérification ou non de la carte est un meilleur moyen pour un attaquant de connaître le prédicat.

4.3.2 Cas de l'opacité symétrique

L'opacité symétrique permet en particulier de modéliser la sécurité d'une valeur binaire. Par exemple, considèrons un canal à entrée binaire et à n sorties possibles. Ce canal peut être modélisé par un système branchant sur les valeurs 0 et 1 depuis l'état initial, puis sur les n observables $\{o_1, \ldots, o_n\}$, comme sur la figure 4.5(a).



(a) Système à secret initial.

(b) Canal à entrée binaire.

FIGURE 4.5: Un système modélisant un canal.

Afin de protéger la valeur de l'entrée, il est aussi important de protéger l'opacité du prédicat « la valeur d'entrée est 0 » que celle de sa négation (« la valeur d'entrée est 1 »). Ce cas est un exemple d'*opacité initiale* [BKMR08], puisque le secret n'apparaît qu'au commencement de l'exécution. En fait, n'importe quel système avec opacité initiale et un nombre fini d'observables peut être transformé en un canal [APvRS10].

La notion d'opacité asymétrique ne prend pas en compte *en même temps* l'opacité d'un prédicat et de sa négation. La mesure d'opacité asymétrique probabiliste robuste souffre donc des même défauts lorsqu'il s'agit d'évaluer conjointement la robustesse de φ et de $\overline{\varphi}$.

Définition 4.3 (Opacité symétrique probabiliste robuste)**.** *Soit $\mathscr{A}$ un automate probabiliste, $\mathscr{O} : TExec(\mathscr{A}) \to Obs$ une fonction d'observation et $\varphi \subseteq TExec(\mathscr{A})$ un prédicat. L'*opacité symétrique probabiliste robuste *(RPSO [6]) de φ sur $\mathscr{A}$ relativement à $\mathscr{O}$ est définie par :*

$$\frac{-1}{PO_r^S(\mathscr{A},\varphi,\mathscr{O})} = \sum_{o \in Obs} \mathbf{P}(\mathscr{O}=o) \cdot \log\big(1 - V(\varphi|\mathscr{O}=o)\big)$$

où V désigne la vulnérabilité (ici conditionnelle) de la variable aléatoire.

De la même manière que la RPO mesure combien la propriété d'opacité asymétrique est proche d'être violée dans chaque classe d'observation, la RPSO mesure cette quantité dans le cas symétrique. Cela revient à mesurer à la fois si la classe est pratiquement incluse dans φ ou si elle est pratiquement incluse dans $\overline{\varphi}$. La vulnérabilité conditionnelle mesure exactement ceci : plus elle est proche de 1, plus il est facile pour un observateur de deviner de manière correcte la valeur de vérité du prédicat. Ainsi la robustesse chute avec l'augmentation de la vulnérabilité.

D'autres mesures utilisant la vulnérabilité sont envisageables, telles que la min-entropie conditionnelle de φ sachant $\mathscr{O}$:

$$H_\infty(\varphi|\mathscr{O}) = -\log\left(\sum_{o \in Obs} \mathbf{P}(\mathscr{O}=o) \cdot V(\varphi|\mathscr{O}=o)\right).$$

Celle-ci est une mesure en bits de la moyenne des vulnérabilités au sein de chaque classe d'observation. À l'inverse, la RPSO fait une moyenne de mesures prises au sein de chaque classe, de manière similaire à la RPO. Le choix de la définition de la RPSO a donné la préférence à une définition plus proche de celle de la RPO, plutôt qu'à une mesure d'entropie. La différence de comportement entre la min-entropie conditionnelle et la RPSO vis-à-vis de l'opacité reste à étudier plus en détails.

Comme dans le cas de la RPO, plus de poids est donné aux classes d'observation les moins robustes, ici en utilisant le logarithme. De plus, lorsque la classe est le plus robuste possible, c'est-à-dire que φ et $\overline{\varphi}$ ont tous deux probabilité $\frac{1}{2}$ de se produire, $\log\big(1 - V(\varphi|\mathscr{O}=o)\big) = -1$. La robustesse globale est ensuite calculée par la moyenne de la robustesse de chaque classe d'observation (pondérée par leur taille). Puisque le logarithme tend vers $-\infty$ en 0, la RPSO est normalisée afin d'obtenir une valeur entre 0 et 1, une valeur 0 signifiant aucune robustesse car violation de l'opacité symétrique, une valeur 1 signifiant que chaque classe d'observation est partagée également entre φ et $\overline{\varphi}$.

Proposition 4.4.

(1) $0 \leq PO_r^S(\mathscr{A},\varphi,\mathscr{O}) \leq 1$

(2) $PO_r^S(\mathscr{A},\varphi,\mathscr{O}) = 0$ *si et seulement si φ n'est pas symétriquement opaque sur $\mathscr{A}$ relativement à $\mathscr{O}$.*

(3) $PO_r^S(\mathscr{A},\varphi,\mathscr{O}) = 1$ *si et seulement si* $\forall o \in Obs, \mathbf{P}(\varphi|\mathscr{O}=o) = \frac{1}{2}$.

Exemples

Système de vente en ligne. On considère le système de vente en ligne de la figure 4.6, inspiré de [AAP10]. Deux types de produits peuvent être mis en vente : des produits bon marché (*cheap*) et des produits coûteux (*expensive*), et deux clients

6. Pour *Restrictive Probabilistic Symmetrical Opacity*.

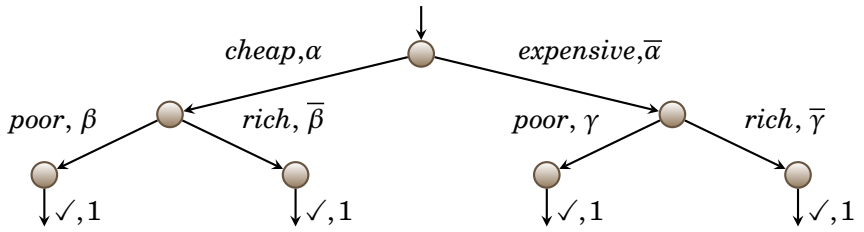


FIGURE 4.6: Un système de vente modélisé par un automate $\mathscr{A}_{\text{vente}}$.

peuvent les acheter : un client aisé (*rich*) ou un client pauvre (*poor*). Les produits sont mis en vente selon un distribution paramétrée par α (avec $\overline{\alpha} = 1 - \alpha$). Les acheteurs ont une tendance différente d'achat selon leurs revenus et le type de produit, modélisé par des distributions de probabilistés paramétrées par β et γ.

Bien que le prix de vente soit public, l'identité de l'acheteur, c'est-à-dire son niveau de revenus, doit rester secrète. En terme d'opacité, la fonction d'observation est $\mathscr{O}_{\text{prix}}$ qui projette chaque exécution vers *cheap* ou *expensive* selon le prix de l'objet effectivement mis en vente. Le secret est, sans perte de généralité, le prédicat φ_{pauvre} qui est vrai lorsque l'exécution contient l'action *poor*. Le biais introduit par une préférence, par exemple, du client pauvre pour les objets bon marché peut trahir l'identité de l'acheteur. La RPSO mesure ce biais. Dans le cas présent, on peut même l'exprimer en fonction des paramètres α, β, et γ. On a :

$$\mathbf{P}(\mathscr{O}_{\text{prix}} = \textit{cheap}) = \alpha \qquad \mathbf{P}(\mathscr{O}_{\text{prix}} = \textit{expensive}) = \overline{\alpha}$$

$$V(\varphi_{\text{pauvre}} | \mathscr{O}_{\text{prix}} = \textit{cheap}) = \max(\beta, \overline{\beta}) \qquad V(\varphi_{\text{pauvre}} | \mathscr{O}_{\text{prix}} = \textit{expensive}) = \max(\gamma, \overline{\gamma})$$

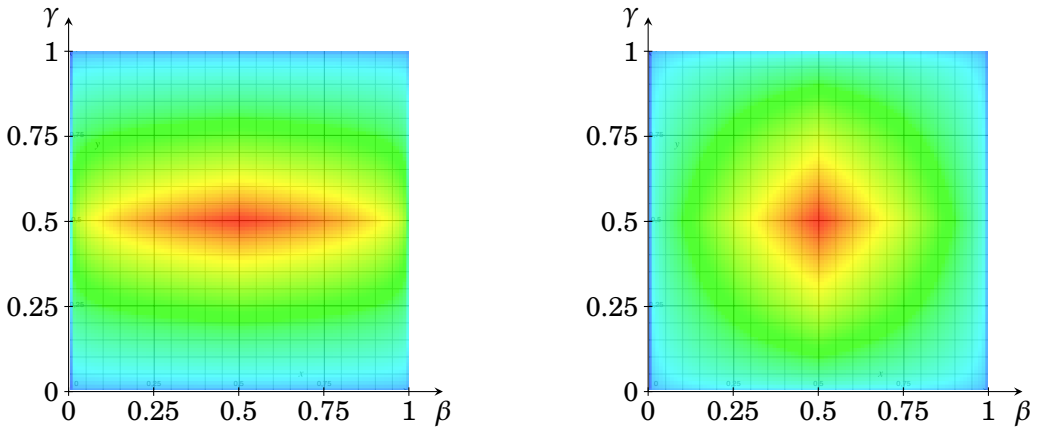
$$\mathsf{PO}_r^S\left(\mathscr{A}_{\text{vente}}, \varphi_{\text{pauvre}}, \mathscr{O}_{\text{prix}}\right) = \frac{-1}{\alpha \cdot \log(\min(\beta, \overline{\beta})) + \overline{\alpha} \cdot \log(\min(\gamma, \overline{\gamma}))}$$

qui est représenté, pour plusieurs valeurs de α sur la figure 4.7.

On y remarque que la sécurité du système est plus robuste lorsque les biais introduits par β et γ sont les plus faibles (c'est-à-dire que β et γ sont proches de $\frac{1}{2}$). Le paramètre α, quant à lui, donne plus d'influence à β ou à γ, mais n'influe sur la sécurité qu'à travers ces autres paramètres.

Protocole du dîner des cryptographes. Le problème du *dîner des cryptographes* de Chaum [Cha88] considère trois cryptographes C_0, C_1, C_2 qui mangent dans un restaurant avec leur directeur. À la fin du repas, le directeur transmet à chaque cryptographe s'il doit s'acquitter de la note. Au plus un seul cryptographe paye, mais le directeur peut choisir de payer lui même. On note p_i la variable valant 1 lorsque « le cryptographe C_i paye ».

Les cryptographes cherchent à savoir si l'un d'entre eux a payé, ou si le directeur l'a fait, et ce, sans que l'identité du payeur soit divulguée. Ils procèdent donc au jeu suivant. Chaque paire de cryptographes adjacents tire une pièce, le résultat du tir n'étant connu que d'eux. On note $f_{i,j}$ le résultat du tir de la pièce entre C_i et C_j (1 pour pile et 0 pour face). Chaque cryptographe annonce ensuite la parité du nombre de « pile » obtenus, en mentant s'il a payé. Plus formellement, le cryptographe C_i annonce la valeur $r_i = f_{i,i+1} \oplus f_{i,i-1} \oplus p_i$ (où les indices sont calculés modulo 3 et $\oplus$



(a) $\mathsf{PO}^S_r\left(\mathscr{A}_{\text{vente}}, \varphi_{\text{pauvre}}, \mathscr{O}_{\text{prix}}\right)$ lorsque $\alpha = \frac{1}{8}$

(b) $\mathsf{PO}^S_r\left(\mathscr{A}_{\text{vente}}, \varphi_{\text{pauvre}}, \mathscr{O}_{\text{prix}}\right)$ lorsque $\alpha = \frac{1}{2}$

FIGURE 4.7: RPSO du système de vente en ligne. Le rouge indique une valeur proche de 1 et le bleu une valeur proche de 0.

est l'addition modulo 2). Ainsi, si $\bigoplus_{i=0}^{2} r_i = 0$, c'est qu'aucun cryptographe n'a menti, donc payé. À l'inverse, si $\bigoplus_{i=0}^{2} r_i = 1$, c'est que l'un d'eux a menti, donc payé le repas. Cependant, les autres cryptographes, ne connaissant chacun que le résultat d'un des tir où chaque autre était concerné, ne peuvent déterminer s'il a menti, donc quelle est l'identité du payeur.

Ici, on ne modélise qu'une instance du protocole où l'un des cryptographes a payé le repas, observée selon le point de vue de C_0, qui est supposé ne pas avoir payé. L'ordre du tir des pièces est fixé : on élimine ainsi les attaques où celui qui paye tire toujours en premier, qui ne compromettent pas la sécurité du système [Kir10]. Le résultat du lancer de pièce entre i et j donne comme résultat pile (action $t_{i,j}$) ou face (action $h_{i,j}$). Le tir de la pièce entre C_1 et C_2 peut être biaisé via un paramètre q. On pourrait aussi modéliser un biais dans les pièces tirées par C_0, mais cela n'affecte pas la sécurité du système. Le choix du directeur pour désigner celui qui paye est modélisé par un tir uniforme entre C_1 et C_2 (actions p_1 et p_2, respectivement). Le modèle est représenté par l'automate probabiliste $\mathscr{A}_{\text{DCP}}$ de la figure 4.8.

L'anonymat du payeur est préservé si C_0 ne peut pas savoir qui de C_1 ou C_2 a payé. Le secret φ_2 est donc la présence dans l'exécution de l'action p_2 (sans perte de généralité). On note $\mathscr{O}_{C_0}$ la fonction d'observation qui projette chaque exécution vers la suite des tirs observée par C_0 (les actions $t_{0,1}$, $h_{0,1}$, $t_{0,2}$ et $h_{0,2}$) ainsi que des résultats donnés par C_1 et C_2 (c'est-à-dire les valeurs de r_1 et r_2, contenues dans le dernier état de l'exécution).

Il y a seize exécutions possibles dans ce système, qui donnent huit observations possibles, toutes équiprobables :

$$\begin{aligned} Obs_{\text{DCP}} = \{&(h_{0,1} \cdot h_{0,2} \cdot (r_1 = 1, r_2 = 0)), (h_{0,1} \cdot h_{0,2} \cdot (r_1 = 0, r_2 = 1)), \\ &(h_{0,1} \cdot t_{0,2} \cdot (r_1 = 0, r_2 = 0)), (h_{0,1} \cdot t_{0,2} \cdot (r_1 = 1, r_2 = 1)), \\ &(t_{0,1} \cdot h_{0,2} \cdot (r_1 = 0, r_2 = 0)), (t_{0,1} \cdot h_{0,2} \cdot (r_1 = 1, r_2 = 1)), \\ &(t_{0,1} \cdot t_{0,2} \cdot (r_1 = 1, r_2 = 0)), (t_{0,1} \cdot t_{0,2} \cdot (r_1 = 0, r_2 = 1))\} \end{aligned}$$

Plus précisément, chaque observation est générée par deux exécutions : l'une où C_1

paye (donc φ_2 est faux) et l'autre où C_2 paye (donc φ_2 est vrai). La différence entre ces deux exécutions est masquée par le tir de la pièce entre C_1 et C_2. Par exemple, les exécutions $\rho_h = h_{0,1} \cdot h_{0,23} \cdot h_{1,2} \cdot p_1 \cdot (r_1 = 1, r_2 = 0)$ et $\rho_t = h_{0,1} \cdot h_{0,2} \cdot t_{1,2} \cdot p_2 \cdot (r_1 = 1, r_2 = 0)$ sont touts deux observée en $o_0 = h_{0,1} \cdot h_{0,2} \cdot (r_1 = 1, r_2 = 0)$. Le prédicat φ_2 est faux dans le cas de ρ_h, mais est vrai dans celui de ρ_t.

En conséquence, si $0 < q < 1$, la version non probabiliste de $\mathscr{A}_{\text{DCP}}$ est symétriquement opaque. Cependant, si $q \neq \frac{1}{2}$, pour chaque observable, l'un des deux cryptographe a une plus forte chance d'avoir triché sur le résultat qu'il annonce, et donc d'avoir payé. Dans l'exemple d'exécutions ci-dessus, lorsque o_0 est observée, ρ_h s'est exécuté avec probabilité q et ρ_t avec probabilité $1-q$. La RPSO mesure l'avantage offert à C_0 par le biais sur la pièce de manière globale.

Pour chaque classe d'observation, la vulnérabilité de φ_2 est $\max(q, 1-q)$. La RPSO sera donc

$$\mathsf{PO}_r^S\left(\mathscr{A}_{\text{DCP}}, \varphi_2, \mathscr{O}_{C_0}\right) = \frac{-1}{\log(\min(q, 1-q))}$$

L'évolution de la PO_r^S en fonction de q est représentée sur la figure 4.9. On observe par exemple que lorsque $q = \frac{1}{2}$, aucune information ne fuit. À l'opposé, lorsque $q = 0$ ou $q = 1$, c'est-à-dire que le tir entre C_1 et C_2 est déterminé, C_0 sait lequel des deux a payé en observant ses tirs et les réponses des deux autres cryptographes.

4.3.3 Mesures du flux d'information

Une autre manière d'appréhender la robustesse de la sécurité d'un système est de mesurer l'information gagnée à travers l'observation. Ou encore, on peut mesurer l'avantage qu'un adversaire disposant de l'observation a par rapport à un observateur aveugle, c'est-à-dire qui connaît le système mais rien de la manière dont celui-ci s'est exécuté.

Ces mesures comparent alors la probabilité de φ au sein de chaque classe d'observation à la probabilité globale de φ. En terme d'information sur le prédicat, cela se traduit par des comparaisons d'entropie, et donc traduit d'une seule manière la ro-

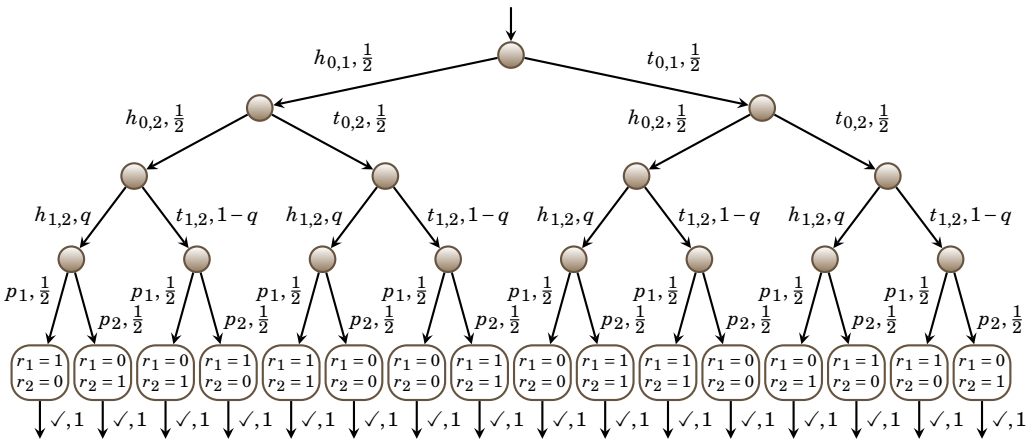


FIGURE 4.8: Automate probabiliste $\mathscr{A}_{\text{DCP}}$ pour le protocole du dîner des cryptographes.

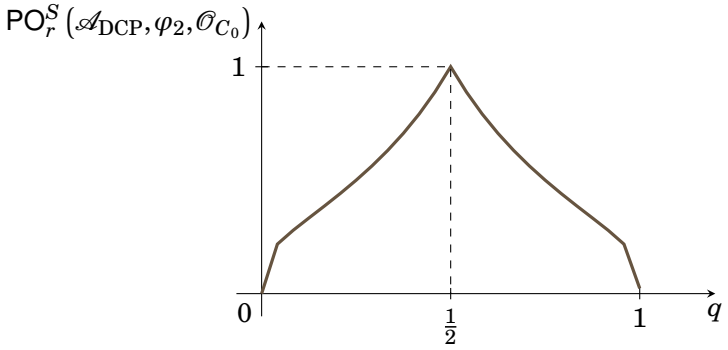


FIGURE 4.9: Variations de la RPSO pour le protocole du dîner des cryptographes en fonction du biais sur la pièce.

bustesse de φ et de $\overline{\varphi}$. Ainsi ces mesures peuvent dans une certaine manière être vues comme des généralisations de l'opacité symétrique, mais pas de l'opacité asymétrique.

Définition 4.4. *Soit $\mathscr{A}$ un automate probabiliste, $\mathscr{O} : TExec(\mathscr{A}) \to Obs$ une fonction d'observation et $\varphi \subseteq TExec(\mathscr{A})$ un prédicat. L'*opacité symétrique probabiliste par entropie *(EPSO) de φ sur $\mathscr{A}$ relativement à $\mathscr{O}$ est définie par :*

$$PO^S_{en}(\mathscr{A}, \varphi, \mathscr{O}) \;=\; 1 - I(\varphi; \mathscr{O}).$$

*L'*opacité symétrique probabiliste par min-entropie *(minPSO) de φ sur $\mathscr{A}$ relativement à $\mathscr{O}$ est définie par :*

$$PO^S_{min}(\mathscr{A}, \varphi, \mathscr{O}) \;=\; 1 - I_\infty(\varphi; \mathscr{O}).$$

*L'*opacité symétrique probabiliste du pire cas *(WCPSO) de φ sur $\mathscr{A}$ relativement à $\mathscr{O}$ est définie par :*

$$PO^S_{wc}(\mathscr{A}, \varphi, \mathscr{O}) \;=\; 1 - \max_{o, o' \in Obs} \left| \mathbf{P}(\varphi | \mathscr{O} = o) - \mathbf{P}(\varphi | \mathscr{O} = o') \right|.$$

La minPSO provient de [Smi09] qui plaide pour l'utilisation de la min-entropie comme mesure de la sécurité. Selon Smith, c'est la vulnérabilité qui est importante lorsque l'on souhaite préserver un secret. Ainsi c'est l'augmentation de la vulnérabilité (de manière globale grâce à la min-entropie) qui mesure un affaiblissement de la sécurité dû à l'observation et donc permet d'évaluer la robustesse.

La WCPSO est le complément à 1 du *flux d'information* de Beauquier, Duflot et Minea [BDM05]. Elle évalue le pouvoir maximal de discrimination que permet l'observation. Cette mesure se différencie de toutes celles définies auparavant car elle ne calcule pas de moyenne [7], mais cherche les classes d'observations « extrêmes » : celles dont la probabilité que le prédicat soit vérifié est maximale ou minimale.

Proposition 4.5.

(1) $0 \le PO^S_{en}(\mathscr{A}, \varphi, \mathscr{O}) \le 1$, $0 \le PO^S_{min}(\mathscr{A}, \varphi, \mathscr{O}) \le 1$, $0 \le PO^S_{wc}(\mathscr{A}, \varphi, \mathscr{O}) \le 1$.

(2) Si $PO^S_{en}(\mathscr{A}, \varphi, \mathscr{O}) = 0$ *ou* $PO^S_{min}(\mathscr{A}, \varphi, \mathscr{O}) = 0$, *alors* $PO^S_\ell(\mathscr{A}, \varphi, \mathscr{O}) = 1$, *donc ni φ ni $\overline{\varphi}$ n'est (symétriquement) opaque sur $\mathscr{A}$ relativement à $\mathscr{O}$.*

7. Au sens large. Par exemple l'entropie est la moyenne des logarithmes des probabilités et la RPO utilise une moyenne harmonique.

(3) Si $PO^S_{wc}(\mathscr{A},\varphi,\mathscr{O}) = 0$*, alors ni* φ *ni* $\overline{\varphi}$ *n'est (symétriquement) opaque sur* $\mathscr{A}$ *relativement à* $\mathscr{O}$*.*

(4) Si $\forall o \in Obs, \mathbf{P}(\varphi|\mathscr{O} = o) = \mathbf{P}(\varphi)$*, alors* $PO^S_{en}(\mathscr{A},\varphi,\mathscr{O}) = 1$ *et* $PO^S_{min}(\mathscr{A},\varphi,\mathscr{O}) = 1$*.*

(5) $PO^S_{wc}(\mathscr{A},\varphi,\mathscr{O}) = 1$ *si et seulement si* $\forall o \in Obs, \mathbf{P}(\varphi|\mathscr{O} = o) = \mathbf{P}(\varphi)$*.*

Démonstration.

(1) On a $0 \leq H(\varphi|\mathscr{O}) \leq H(\varphi) \leq \log(2) = 1$ (et de même pour la min-entropie H_∞) car la variable aléatoire φ ne peut prendre que deux valeurs et l'entropie ne peut que décroître avec la connaissance de $\mathscr{O}$. Dans le cas de la WCPSO, on a une différence de probabilités, ce qui donne directement l'encadrement.

(2) Pour EPSO, ce cas ne peut se produire seulement si $H(\varphi) = 1$ et $H(\varphi|\mathscr{O}) = 0$, ce qui correspond au cas (3) de la proposition 4.2, c'est-à-dire $\mathsf{PO}^S_\ell(\mathscr{A},\varphi,\mathscr{O}) = 1$. Dans le cas de minPSO, on a, de la même manière, $H_\infty(\varphi) = 1$ et $H_\infty(\varphi|\mathscr{O}) = 0$. c'est-à-dire $V(\varphi) = \frac{1}{2}$ et $\sum_{o\in Obs} \mathbf{P}(\mathscr{O} = o) \cdot V(\varphi|\mathscr{O} = o) = 1$. Donc pour toute observable $o \in Obs$, $V(\varphi|\mathscr{O}) = 1$: soit $\mathbf{P}(\varphi|\mathscr{O} = o) = 0$, soit $\mathbf{P}(\varphi|\mathscr{O} = o) = 1$, c'est-à-dire $\mathsf{PO}^S_\ell(\mathscr{A},\varphi,\mathscr{O}) = 1$.

(3) Si $\mathsf{PO}^S_{wc}(\mathscr{A},\varphi,\mathscr{O}) = 0$, alors il existe $o, o' \in Obs$ tels que $\mathbf{P}(\varphi|\mathscr{O} = o) = 1$ et $\mathbf{P}(\varphi|\mathscr{O} = o') = 0$. Étant donné que la fonction d'observation est supposée surjective, on a $\mathbf{P}(\mathscr{O} = o) > 0$ et $\mathbf{P}(\mathscr{O} = o') > 0$, ce qui donne le résultat.

(4) Si pour toute observable $o \in Obs$, $\mathbf{P}(\varphi|\mathscr{O} = o) = \mathbf{P}(\varphi)$, les entropies conditionnelles ou non sont égales, d'où le résultat.

(5) Pour toutes observables $o, o' \in Obs$, $\mathbf{P}(\varphi|\mathscr{O} = o) = \mathbf{P}(\varphi|\mathscr{O} = o') = \mathbf{P}(\varphi)$, donc les différences de ces probabilités sont toujours nulles, donc $\mathsf{PO}^S_{wc}(\mathscr{A},\varphi,\mathscr{O}) = 1$. Réciproquement le WCPSO ne peut être nul que si toutes les probabilités conditionnelles sont égales. Elles sont donc aussi toutes égales à leur moyenne, $\mathbf{P}(\varphi)$.

□

Exemple : retour sur le protocole du dîner des cryptographes

On peut évaluer la fuite d'information induite par le biais q sur la pièce avec les mesures décrites ci-dessus.

EPSO. On a $H(\varphi_2) = 1$ car l'identité du payeur est tirée aléatoirement par une pièce non biaisée. D'autre part

$$H(\varphi_2|\mathscr{O}_{C_0}) \;=\; -\sum_{\substack{o\in Obs_{\mathrm{DCP}}\\ i\in\mathbb{B}}} \mathbf{P}(\mathscr{O}_{C_0} = o)\cdot\mathbf{P}(\varphi_2 = i|\mathscr{O}_{C_0} = o)\cdot\log\big(\mathbf{P}(\varphi_2 = i|\mathscr{O}_{C_0} = o)\big)$$

Puisque pour toute observable $o \in Obs_{\mathrm{DCP}}$, $\mathbf{P}(\varphi_2 = 0|\mathscr{O}_{C_0} = o) = 1 - \mathbf{P}(\varphi_2 = 1|\mathscr{O}_{C_0} = o)$, et que $\mathbf{P}(\varphi_2 = 0|\mathscr{O}_{C_0} = o)$ est soit q soit $1-q$, on obtient :

$$\mathsf{PO}^S_{en}\left(\mathscr{A}_{\mathrm{DCP}},\varphi_2,\mathscr{O}_{C_0}\right) \;=\; -(q\cdot\log(q) + (1-q)\cdot\log(1-q))$$

Les variations de la EPSO sont représentées sur la figure 4.10. On remarque les mêmes extrêmes que dans le cas de la RPSO, même si la forme de la courbe est différente.

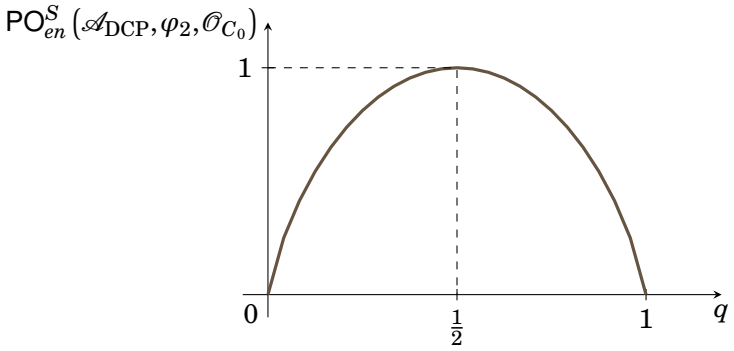


FIGURE 4.10: Variations de la EPSO pour le protocole du dîner des cryptographes en fonction du biais sur la pièce.

minPSO. On a $H_\infty(\varphi_2) = 1$ pour les mêmes raisons que dans le cas de la EPSO. Par ailleurs $H_\infty(\varphi_2|\mathscr{O}_{C_0}) = -\log\left(V(\varphi_2|\mathscr{O}_{C_0})\right)$ et

$$\begin{aligned} V(\varphi_2|\mathscr{O}_{C_0}) &= \sum_{o \in Obs_{\mathrm{DCP}}} \mathbf{P}(\mathscr{O}_{C_0} = o) \cdot V(\varphi|\mathscr{O}_{C_0} = o) \\ &= \sum_{o \in Obs_{\mathrm{DCP}}} \mathbf{P}(\mathscr{O}_{C_0} = o) \cdot \max(q, 1-q) \\ V(\varphi_2|\mathscr{O}_{C_0}) &= \max(q, 1-q) \end{aligned}$$

$$\text{Donc} \quad \mathsf{PO}^S_{min}\left(\mathscr{A}_{\mathrm{DCP}}, \varphi_2, \mathscr{O}_{C_0}\right) = -\log(\max(q, 1-q)).$$

L'évolution de la minPSO est représentée sur la figure 4.11. Une fois encore, la forme de la courbe change, mais les points extrémaux sont les mêmes.

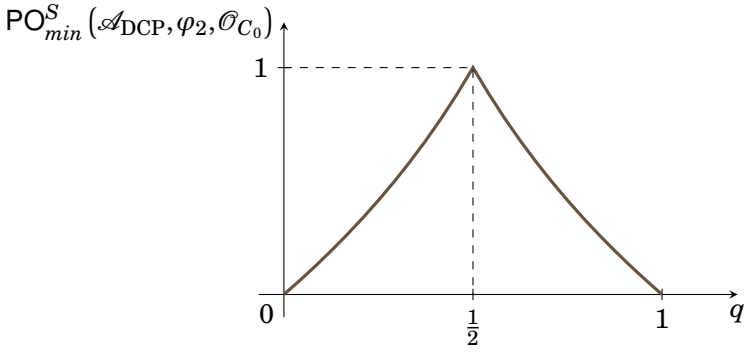


FIGURE 4.11: Variations de la minPSO pour le protocole du dîner des cryptographes en fonction du biais sur la pièce.

WCPSO. Pour toute observable $o \in Obs_{\mathrm{DCP}}$, on a soit $\mathbf{P}(\varphi|\mathscr{O}_{C_0} = o) = 1 - q$, soit $\mathbf{P}(\varphi|\mathscr{O}_{C_0} = o) = q$. Donc $\mathsf{PO}^S_{wc}\left(\mathscr{A}_{\mathrm{DCP}}, \varphi_2, \mathscr{O}_{C_0}\right) = 1 - |1 - 2q| = 2 \cdot \min(q, 1-q)$. Cette fonction vaut elle aussi 1 si $q = \frac{1}{2}$ et 0 si $q = 1$ ou $q = 0$.

4.4 Calcul automatique des mesures d'opacité

On montre à présent comment calculer automatiquement les valeurs des mesures définies dans ce chapitre. On se place dans le cas restreint où le prédicat φ est un sous ensemble régulier d'exécutions et où l'ensemble des observables *Obs* est fini. De plus on suppose que chaque classe d'observation est un ensemble régulier d'exécutions.

Définition 4.5 (Fonction d'observation régulière)**.** *Soit $\mathscr{A}$ un automate probabiliste et Obs un ensemble fini. Une fonction d'observation $\mathscr{O} : TExec(\mathscr{A}) \to Obs$ est dite* régulière *si pour toute observable $o \in Obs$, $\mathscr{O}^{-1}(o)$ est un ensemble régulier d'exécutions.*

Un calcul automatique dans le cas général est impossible car l'opacité est indécidable [BKMR08]. Une comparaison à 0 ou à 1, correspondant souvent à un test d'opacité, est cependant possible dans certain cas. Les travaux de Cassez, Dubreil et Marchand [CDM09] étudient l'opacité dans le cas où le prédicat correspond à des états terminaux et les observations sont des projections sur un sous-alphabet. Beauquier, Duflot et Lifshits [BDL07] donnent une procédure de décision pour l'égalité de la WCPSO avec 1, lorsque φ est donné par un automate de Muller et l'observation est une projection.

La méthode de calcul utilisée ici est dérivée de celles pour la vérification probabiliste [HJ94, BdA95, CY98]. Elle se base sur la synchronisation d'un automate probabiliste $\mathscr{A}$ avec un automate fini déterministe $\mathscr{A}_L$ qui reconnaît un langage L. Ce produit contraint la version non probabiliste de $\mathscr{A}$ en la synchronisant avec $\mathscr{A}_L$. La probabilité de L est ensuite obtenue par résolution d'un système d'équations linéaires obtenues à partir du produit synchronisé. Le calcul de chaque mesure consiste en plusieurs calculs de telles probabilités.

Nous montrons dans la suite de cette section le résultat suivant :

Théorème 4.6 (Calcul des mesures d'opacité)**.** *Soit $\mathscr{A}$ un automate probabiliste et $\varphi \subseteq TExec(\mathscr{A})$ un ensemble régulier. Soit $\mathscr{O} : TExec(\mathscr{A}) \to Obs$ une fonction d'observation régulière. Alors pour $PO^* \in \{PO^A_\ell, PO^S_\ell, PO^A_r, PO^S_r, PO^S_{en}, PO^S_{min}, PO^S_{wc}\}$, la valeur de $PO^*(\mathscr{A}, \varphi, \mathscr{O})$ peut être calculée.*

Calcul de la probabilité associée à un automate sous-stochastique

Étant donné un automate sous-stochastique, on construit un système d'équations dont la solution est, pour chaque état, la probabilité d'atteindre un état final. Ceci donne en particulier la probabilité de toutes les exécutions totales de cet automate.

Définition 4.6 (Système linéaire associé à un automate sous-stochastique)**.** *Soit $\mathscr{A} = \langle S, Lab, \Delta, s_I \rangle$ un automate sous-stochastique. Le* système linéaire associé *à $\mathscr{A}$ est le système $\mathscr{S}_\mathscr{A}$ d'équations linéaires sur $\mathbb{R}$ défini par :*

$$\mathscr{S}_\mathscr{A} = \left(X_s = \sum_{s' \in S} \alpha_{s,s'} X_{s'} + \beta_s \right)_{s \in S} \quad \text{où} \quad \alpha_{s,s'} = \sum_{a \in Lab} \Delta(s)(a,s') \quad \text{et} \quad \beta_s = \Delta(s)(\checkmark)$$

Le cas non déterministe des processus de décision markoviens [BdA95, CY98] nécessite deux systèmes d'inéquations afin de calculer les probabilités minimales et maximales. Ici, ces deux valeurs se rejoignent, et on peut les calculer en temps polynomial en résolvant le système associé.

Lemme 4.7 (Solution du système)**.** *Soit $\mathscr{A} = \langle S, Lab, \Delta, s_I \rangle$ un automate sous-stochastique. On pose pour tout état $s \in S$, $K_s^{\mathscr{A}} = \mathbf{P}(TExec_s(\mathscr{A}))$. Alors $\left(K_s^{\mathscr{A}}\right)_{s \in S}$ est l'unique solution du système $\mathscr{S}_{\mathscr{A}}$.*

La probabilité associé à $\mathscr{A}$ est donc $K_{s_I}^{\mathscr{A}} = \mathbf{P}_{\mathscr{A}}\left(TExec_{s_I}(\mathscr{A})\right)$.

Calcul de la probabilité d'un langage régulier

Afin de calculer la probabilité d'un langage régulier L au sein d'un automate probabiliste $\mathscr{A}$, c'est-à-dire $\mathbf{P}_{\mathscr{A}}(L)$, on construit un automate sous-stochastique correspondant à l'intersection du système avec le langage. On calcule ensuite la probabilité associée à ce produit à l'aide du système linéaire ci-dessus.

Définition 4.7 (Automate sur les exécutions)**.** *Soit $\mathscr{A} = \langle S, Lab, \Delta, s_I \rangle$ un automate sous-stochastique. Un* automate fini sur les exécutions de $\mathscr{A}$ *est un automate fini $\mathscr{A}_L$ sur l'alphabet $S \times Lab \times S$. Une exécution totale $\rho = s_0 \xrightarrow{a_1} s_1 \cdots \xrightarrow{a_n} s_n \checkmark$ est* acceptée *par $\mathscr{A}_L$ si le mot $(s_0, a_1, s_1)(s_1, a_2, s_2) \cdots (s_{n-1}, a_n, s_n)$ est accepté par $\mathscr{A}_L$. Le* langage *de $\mathscr{A}_L$ est l'ensemble des exécutions qu'il accepte.*

Dans la suite de cette section, les automates finis que l'on considère sont des automates sur les exécutions.

Définition 4.8 (Produit synchronisé)**.** *Soit $\mathscr{A} = \langle S, Lab, \Delta, s_I \rangle$ un automate sous-stochastique et $\mathscr{A}_L = \langle Q, S \times Lab \times S, \Delta_L, q_I, F \rangle$ un automate fini déterministe sur les exécutions de $\mathscr{A}$. Le* produit synchronisé *$\mathscr{A} \,||\, \mathscr{A}_L = \left\langle S \times Q, Lab, \Delta', (s_I, q_I) \right\rangle$ est l'automate sous-stochastique où les transitions de Δ' sont définies par : si $s \to \mu \in \Delta$, alors $(s,q) \to \nu \in \Delta'$ avec, pour $a \in Lab$ et $(s', q') \in S \times Q$,*

$$\nu\left(a, (s', q')\right) = \begin{cases} \mu(a, s') & \text{si } q \xrightarrow{s,a,s'} q' \in \Delta_L \\ 0 & \text{sinon} \end{cases} \qquad \text{et} \qquad \nu(\checkmark) = \begin{cases} \mu(\checkmark) & \text{si } q \in F \\ 0 & \text{sinon.} \end{cases}$$

Le produit synchronisé est défini en toute généralité sur les automates sous-stochastiques, bien qu'il soit souvent utilisé dans le cas particulier des automates probabilistes. Cette définition plus générale autorise néanmoins de composer successivement un automate probabiliste avec plusieurs automates finis déterministes [8].

Dans ce produit synchronisé, les actions de l'automate sous-stochastique sont contraintes par celles de l'automate fini. Ainsi les actions qui ne sont pas autorisées par l'automate fini sont « coupées », et les états ne peuvent terminer l'exécution que si l'automate fini les y autorise.

Le lien entre la probabilité d'un langage dans un automate probabiliste et ce produit synchronisé est formalisé par le résultat suivant.

Lemme 4.8 (Probabilité d'un langage régulier)**.** *Soit $\mathscr{A} = \langle S, Lab, \Delta, s_I \rangle$ un automate sous-stochastique et L un langage régulier sur $S \times Lab \times S$ accepté par l'automate fini déterministe $\mathscr{A}_L = \langle Q, S \times Lab \times S, \Delta_L, q_I, F \rangle$. Alors*

$$\mathbf{P}_{\mathscr{A}}(L) = K_{(s_I, q_I)}^{\mathscr{A} || \mathscr{A}_L} \qquad \text{où } \left(K_{(s,q)}^{\mathscr{A} || \mathscr{A}_L}\right)_{(s,q) \in S \times Q} \text{ est la solution du système } \mathscr{S}_{\mathscr{A} || \mathscr{A}_L}.$$

8. Cela revient à composer l'automate probabiliste avec l'intersection des automates déterministes. Une séquence de compositions est cependant plus intuitive dans l'utilisation qui en est faite par la suite.

Démonstration. Soit $\rho \in TExec(\mathscr{A}) \cap L$, avec $\rho = s_I \xrightarrow{a_1} s_1 \cdots s_{n-1} \xrightarrow{a_n} s_n\checkmark$. On considère l'exécution σ dans $\mathscr{A}_L$ qui accepte ρ (cette exécution est unique car $\mathscr{A}_L$ est déterministe). Plus précisément, $\sigma = q_I \xrightarrow{s_I,a_1,s_1} q_1 \cdots q_{n-1} \xrightarrow{s_{n-1},a_n,s_n} q_n$, avec $q_n \in F$. L'exécution $\theta = (s_I,q_I) \xrightarrow{a_1} (s_1,q_1)\cdots(s_{n-1},q_{n-1}) \xrightarrow{a_n} (s_n,q_n)\checkmark$ est donc une exécution totale de $\mathscr{A} \,||\, \mathscr{A}_L$. Il y a en fait bijection entre les paires d'exécutions dans $\mathscr{A}$ et $\mathscr{A}_L$ correspondant à la même exécution (ici la paire (ρ,σ)) et les exécutions de $\mathscr{A} \,||\, \mathscr{A}_L$ (ici θ). De plus

$$\begin{aligned}
\mathbf{P}_{\mathscr{A}||\mathscr{A}_L}(\theta) &= \big(\Delta'(s_I,q_I)(a_1,(s_1,q_1))\big)\cdots\big(\Delta'(s_n,q_n)(\checkmark)\big) \\
&= (\Delta(s_I)(a_1,s_1))\cdots(\Delta(s_n)(\checkmark)) \\
\mathbf{P}_{\mathscr{A}||\mathscr{A}_L}(\theta) &= \mathbf{P}_{\mathscr{A}}(\rho).
\end{aligned}$$

En conséquence

$$\begin{aligned}
\mathbf{P}_{\mathscr{A}}(L) &= \sum_{\rho\in L} \mathbf{P}_{\mathscr{A}}(\rho) \\
&= \sum_{\theta\in TExec(\mathscr{A}||\mathscr{A}_L)} \mathbf{P}_{\mathscr{A}||\mathscr{A}_L}(\rho') \\
\mathbf{P}_{\mathscr{A}}(L) &= \mathbf{P}_{\mathscr{A}||\mathscr{A}_L}(TExec(\mathscr{A} \,||\, \mathscr{A}_L))
\end{aligned}$$

et par le lemme 4.7, $\mathbf{P}_{\mathscr{A}}(L) = K^{\mathscr{A}||\mathscr{A}_L}_{(s_I,q_I)}$. □

Calcul des mesures d'opacité

On peut remarquer que le calcul de toutes les mesures définies auparavant repose sur le calcul des valeurs de

$$\mathbf{P}(\varphi = i) \qquad \mathbf{P}(\mathscr{O} = o) \qquad \mathbf{P}(\varphi = i, \mathscr{O} = o)$$

pour chaque observable $o \in Obs$ et pour $i \in \mathbb{B}$. Par exemple, le test de savoir si $\mathscr{O}^{-1}(o) \subseteq \varphi$ peut être effectué en testant si on a à la fois $\mathbf{P}(\mathscr{O} = o) > 0$ et [9] $\mathbf{P}(\overline{\varphi}, \mathscr{O} = o) = 0$.

Rappelons que l'on a supposé ici que φ est un ensemble régulier d'exécutions, accepté par un automate $\mathscr{A}_\varphi$ (sur les exécutions de $\mathscr{A}$). Notons que dans ce cas $\overline{\varphi}$ est lui aussi un ensemble régulier, accepté par l'automate $\mathscr{A}_{\overline{\varphi}}$. On a de plus supposé que Obs était de cardinal fini et que pour toute observable $o \in Obs$, l'ensemble d'exécutions $\mathscr{O}^{-1}(o)$ est régulier, chacun accepté par l'automate $\mathscr{A}_o$ (sur les exécutions de $\mathscr{A}$).

Lorsque l'on synchronise $\mathscr{A}$ et $\mathscr{A}_\varphi$, on obtient l'automate $\mathscr{A} \,||\, \mathscr{A}_\varphi$. Le lemme 4.8 implique que $\mathbf{P}(\varphi)$ est la valeur de la première ligne [10] de la solution du système linéaire associé à $\mathscr{A} \,||\, \mathscr{A}_\varphi$. La probabilité $\mathbf{P}(\overline{\varphi})$ est obtenue de la même manière, avec le produit $\mathscr{A} \,||\, \mathscr{A}_{\overline{\varphi}}$.

La synchronisation de $\mathscr{A}$ avec chaque $\mathscr{A}_o$ permet d'obtenir $\mathbf{P}(\mathscr{O} = o)$. Les produits entre $\mathscr{A} \,||\, \mathscr{A}_\varphi$ (respectivement $\mathscr{A} \,||\, \mathscr{A}_{\overline{\varphi}}$) et $\mathscr{A}_o$ donnent les probabilités $\mathbf{P}(\varphi, \mathscr{O} = o)$ (respectivement $\mathbf{P}(\overline{\varphi}, \mathscr{O} = o)$).

Le calcul des mesures d'opacités est effectué en temps polynomial en la taille de Obs, des automates $\mathscr{A}_\varphi$, $\mathscr{A}_{\overline{\varphi}}$ et de tous les $\mathscr{A}_o$, pour $o \in Obs$.

La procédure de calcul pour ces mesures a été implémentée en Java par Adrian Eftenie dans l'outil TPOT [Eft10]. Cet outil produit des valeurs numériques qui permettent, par exemple, de comparer différentes version d'un modèle du point de vue de la sécurité.

9. Bien que l'on suppose $\mathscr{O}$ surjective, il est en pratique plus aisé de vérifier cette surjectivité au moment du calcul des probabilités.

10. En supposant que l'équation correspondant à l'état initial se trouve toujours à la première ligne du système.

4.5 Comparaison des mesures

Les propriétés de chaque mesure donnent une première intuition sur ce qu'elles mesurent réellement. On cherche donc à comparer plus finement ces mesures. Pour cela, on cherche à savoir ce qu'elles permettent de différencier dans le comportement d'un système.

4.5.1 Des exemples abstraits

On étudie dans un premier temps des systèmes très simples, réduits aux distributions de probabilités de φ au sein de chaque classe. Plus précisément, on considère des systèmes ayant tous quatre observables $\{a,b,c,d\}$ équiprobables, mais dont la probabilité de φ au sein de la classe d'observation est paramétrée.

De tels systèmes peuvent être modélisés par des automates probabilistes comme celui de la figure 4.12. En prenant φ le prédicat qui contient les exécutions se terminant dans l'état φ, le paramètre α donne $\mathbf{P}(\varphi|\mathscr{O} = a)$ (et de même pour les autres lettres grecques, qui donnent la probabilité de φ sachant que leur correspondant latin est observé).

On représente ces systèmes sous forme d'un carré découpé en quatre quarts identiques (les quatre classes d'observation). La partie colorée correspond à φ. Ainsi la surface colorée au sein de chaque classe donne la probabilité de φ dans cette classe. Les systèmes que l'on considère sont ceux de la figure 4.13. Les paramètres α, β, γ et δ n'y prennent comme valeur que 0, $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$ ou 1. Par exemple, le système $\mathscr{A}_7$ correspond à $\alpha = \beta = \gamma = \delta = \frac{1}{2}$, tandis que $\mathscr{A}_{13}$ correspond aux valeurs $\alpha = \frac{1}{4}$, $\beta = \frac{1}{2}$, $\gamma = \frac{3}{4}$, $\delta = 0$. Les valeurs des différentes mesures d'opacité sont rassemblées sur la table 4.3.

Le système $\mathscr{A}_7$ est intuitivement très sûr : avec ou sans observation, un attaquant n'a pas plus d'information sur la valeur de vérité de φ que sur le résultat d'un tir d'une pièce non biaisée. Cette intuition est confirmée par toutes les mesures de l'opacité symétrique. Cependant, la RPO donne un moins bon score : en effet, l'opacité asymétrique n'est parfaite (donc atteint un score de 1) uniquement si le prédicat et toujours faux.

Le cas de $\mathscr{A}_8$ ne diffère de $\mathscr{A}_7$ que par la proportion globale de φ. Ainsi, l'information disponible à l'attaquant ne provient pas de son observation, mais de cette proportion globale. Les mesures issues de la théorie de l'information ne mesurant que ce qui est « gagné » à travers l'observation, EPSO, minPSO et WCPSO mesurent toujours ce système comme très sûr. La RPSO, au contraire, prend en compte la sécurité du système après l'observation, sans préjuger de la sécurité du système sans observation. Ainsi, $\mathscr{A}_8$ est jugé moins opaque par la RPSO. Cependant, la RPO l'évalue comme plus sûr : ce système est plus proche de l'idéal d'opacité asymétrique. On peut remarquer que si l'on cherchait à évaluer l'opacité de $\overline{\varphi}$, cela ne changerait rien pour les mesures symétriques, mais le système serait évalué comme moins sûr par la RPO (on a en effet $\mathsf{PO}_r^A\left(\mathscr{A}_8, \overline{\varphi}, \mathscr{O}\right) = \frac{1}{4}$).

Puisque chaque classe d'observation est considérée individuellement, la RPSO ne permet pas de distinguer les systèmes $\mathscr{A}_8$ et $\mathscr{A}_9$. Ici, l'information au sein de chaque classe d'information est la même dans $\mathscr{A}_8$ que dans $\mathscr{A}_9$ (répartition $\left(\frac{1}{4}, \frac{3}{4}\right)$ dans tous

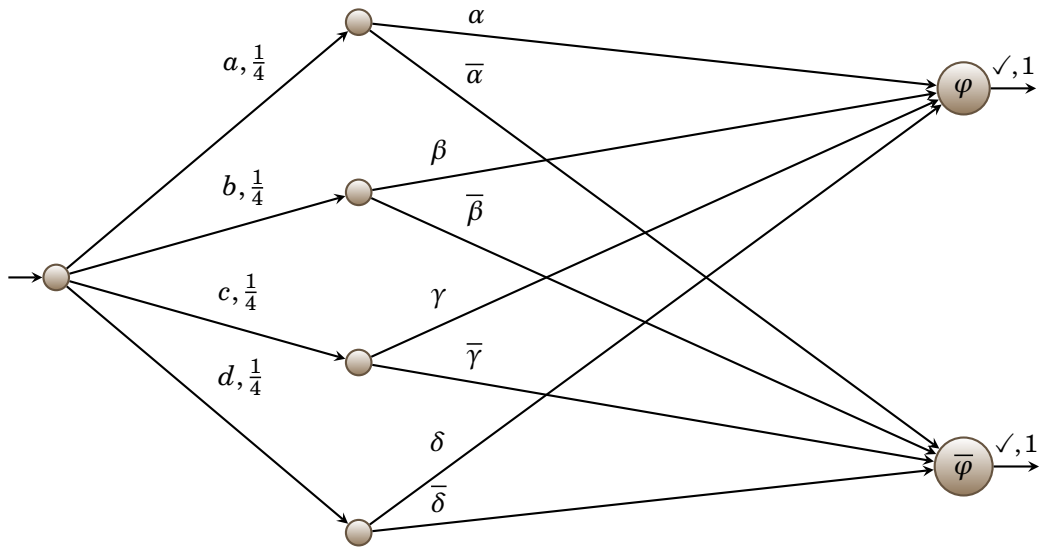
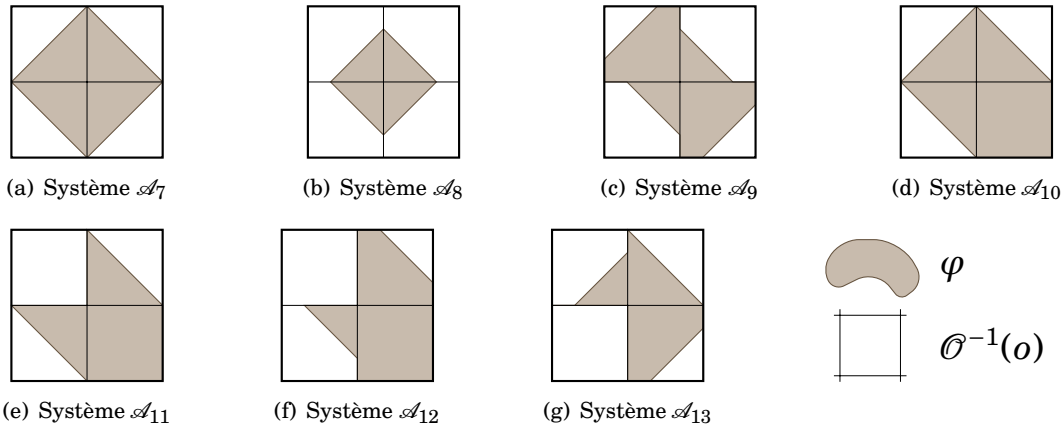


FIGURE 4.12: Un système paramétré par $\alpha, \beta, \gamma, \delta$.



(a) Système $\mathscr{A}_7$ (b) Système $\mathscr{A}_8$ (c) Système $\mathscr{A}_9$ (d) Système $\mathscr{A}_{10}$

(e) Système $\mathscr{A}_{11}$ (f) Système $\mathscr{A}_{12}$ (g) Système $\mathscr{A}_{13}$

FIGURE 4.13: Instances du système de la figure 4.12 avec différentes valeurs des paramètres.

	LPO	LPSO	RPO	RPSO	EPSO	minPSO	WCPSO
$\mathscr{A}_7$	0	0	$\frac{1}{2}$	1	1	1	1
$\mathscr{A}_8$	0	0	$\frac{3}{4}$	$\frac{1}{2}$	1	1	1
$\mathscr{A}_9$	0	0	$\frac{3}{8}$	$\frac{1}{2}$	$2-\frac{3}{4}\log 3 \simeq 0.81$	$2-\log 3 \simeq 0.42$	$\frac{1}{2}$
$\mathscr{A}_{10}$	$\frac{1}{4}$	$\frac{1}{4}$	0	0	$\frac{\log 15-10}{8} \simeq 0.80$	1	$\frac{1}{2}$
$\mathscr{A}_{11}$	$\frac{1}{4}$	$\frac{1}{2}$	0	0	$\frac{1}{2}$	$2-\log 3 \simeq 0.42$	0
$\mathscr{A}_{12}$	$\frac{1}{4}$	$\frac{1}{2}$	0	0	$1-\frac{3}{8}\log 3 \simeq 0.41$	$3-\log 7 \simeq 0.19$	0
$\mathscr{A}_{13}$	0	$\frac{1}{4}$	$\frac{12}{25}$	0	$\frac{5\log 5-6}{8} \simeq 0.70$	$\log\frac{5}{3} \simeq 0.74$	$\frac{1}{4}$

TABLE 4.3: Valeurs des mesures d'opacité pour les systèmes de la figure 4.13.

les cas). Cependant, la répartition globale de φ dans le cas de $\mathscr{A}_9$ ($\frac{1}{2}$) ne donne aucun avantage à un observateur aveugle. Au contraire, comme le montrent les valeurs de EPSO, minPSO et WCPSO, l'observation donne beaucoup d'information à l'attaquant, et donc corrompt la sécurité d'autant.

Le système $\mathscr{A}_{10}$ montre quant à lui les limites de la minPSO. En effet, le déséquilibres au sein de l'une des classes d'observation est atténué par l'équilibre au sein des autres classes d'observation, au point d'obtenir, dans le cas présent, la même vulnérabilité pour la variable aléatoire φ et la variable φ conditionnée par $\mathscr{O}$ (ici $\frac{5}{8}$). Le système $\mathscr{A}_{10}$ est cependant non opaque : en effet, la RPO et la RPSO sont toutes deux nulles tandis que la LPO et la LPSO ne le sont pas. Remarquons aussi que la EPSO n'est pas nulle, ne permettant donc pas de détecter la non opacité du système.

Dans les cas des systèmes non opaques, les mesures faibles deviennent pertinentes afin de mesurer la quantité de non opacité. Par exemple, $\mathscr{A}_{11}$ a un plus haut score pour la LPSO (donc une moins bonne sécurité) que $\mathscr{A}_{10}$. Cependant, le score de la LPO est inchangé : la classe où φ n'est jamais vérifiée n'est pas prise en compte. La WCPSO, qui ne détecte pas la non opacité de $\mathscr{A}_{10}$, détecte cependant que ni φ ni $\overline{\varphi}$ ne sont opaque sur $\mathscr{A}_{11}$. La EPSO et la minPSO sont aussi affectées par la grande proportion de classes très déséquilibrées dans $\mathscr{A}_{11}$, sans toutefois atteindre 0.

Ces mesures ont cependant l'avantage de pouvoir évaluer plus finement la sécurité, même lorsque le système est non opaque. Ainsi, la LPO et la LPSO ne discriminent pas entre $\mathscr{A}_{11}$ et $\mathscr{A}_{12}$: les classes dont l'information sur φ est totalement dévoilée restent identiques, seulement dans $\mathscr{A}_{12}$, les autres classes sont elles aussi plus déséquilibrées. Ceci est donc capturé par la EPSO et la minPSO. En revanche la WCPSO souffre des même limites que les mesures faibles de l'opacité : hormis les classes extrêmes du point de vue de la probabilité de φ, rien n'est mesuré.

La WCPSO ne différencie pas non plus fondamentalement le cas de $\mathscr{A}_{13}$ (où φ est opaque mais pas $\overline{\varphi}$) du cas de $\mathscr{A}_{10}$ (où $\overline{\varphi}$ est opaque mais pas φ). Notons que le système $\mathscr{A}_{13}$ est opaque mais pas symétriquement opaque : les mesures pertinentes sont donc la LPSO dans le cas symétrique et la RPO dans le cas asymétrique.

4.5.2 Des exemples plus réels

On étudie maintenant des exemples modélisant des systèmes plus réels, afin d'évaluer la façon dont les mesures se comportent en pratique.

Fuite d'information dans des programmes à affectation

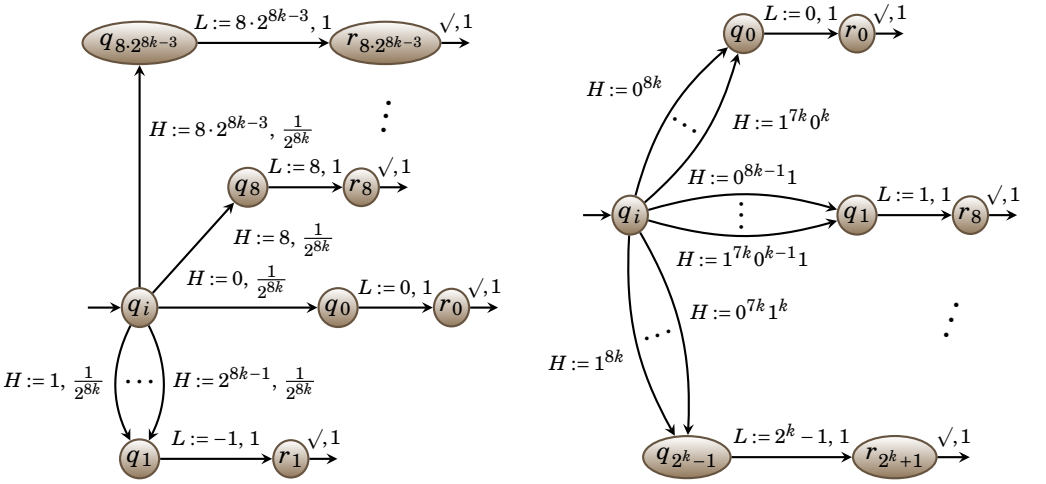
On considère les programmes P_1 et P_2 de la table 4.4, inspirés de [Smi09]. L'entier k est un paramètre connu du programme. La fonction `random` renvoie un entier compris entre ses deux arguments (inclus), en tirant selon une distribution uniforme. L'opérateur & représente le *et* bit à bit. La variable H est une variable privée, dont le contenu doit rester secret, tandis que L est une variable publique.

La valeur de H est un entier sur $8k$ bits déterminée par un tir aléatoire uniforme. Elle ne peut être connue qu'à travers l'observation de L. Intuitivement, le programme P_1 divulgue toute l'information sur H une fois sur huit (c'est-à-dire avec probabilité $\frac{1}{8}$). Le programme P_2, quant à lui, donne à chaque exécution un huitième de l'information sur H (en donnant les k bits de poids faible).

P_1 : $H := \texttt{random}(0, 2^{8k}-1)$;
if $H \mod 8 = 0$ then
  $L := H$
else
  $L := -1$
fi

P_2 : $H := \texttt{random}(0, 2^{8k}-1)$;
$L := H \,\&\, 0^{7k}1^k$

TABLE 4.4: Programmes P_1 et P_2.



(a) Automate probabiliste $\mathscr{A}_{P_1}$.

(b) Automate probabiliste $\mathscr{A}_{P_2}$. Chaque transition sortant de q_i a une probabilité de $\frac{1}{2^{8k}}$.

FIGURE 4.14: Automates probabilistes modélisant les programmes P_1 et P_2.

	LPO	LPSO	RPO	RPSO	EPSO	minPSO	WCPSO
P_1	$\frac{1}{8}$	1	0	0	$\frac{7}{8}\log 7 - 2 \simeq 0.46$	$\log 7 - 2 \simeq 0.81$	0
P_2	0	0	$1 - \frac{1}{2^{7k}}$	$\frac{1}{7k}$	1	1	1

TABLE 4.5: Mesures d'opacité pour les programmes P_1 and P_2.

Ces programmes peuvent être modélisés par les automates probabilistes $\mathscr{A}_{P_1}$ et $\mathscr{A}_{P_2}$ de la figure 4.14. La fonction d'observation est la projection de l'exécution sur les actions de la forme « $L := \ldots$ ». Le secret étant un prédicat binaire et non la *valeur* de H (sur $8k$ bits), on choisit le prédicat $\varphi_=$ qui est vrai lorsque L vaut exactement H :

$$\varphi_= \;=\; tr^{-1}\left(\left\{(H := x)(L := x) \,\middle|\, x \in \left\{0, \ldots, 2^{8k}-1\right\}\right\}\right)$$

Les valeurs de l'opacité pour les différentes mesures sont rassemblées sur la table 4.5.

On peut tout d'abord remarquer que $\varphi_=$ n'est pas opaque sur $\mathscr{A}_{P_1}$ (que ce soit pour l'opacité asymétrique ou symétrique). Ainsi les mesures RPO et RPSO sont nulles. De plus, puisque la fuite d'information est totale dans certains cas, la WCPSO est elle aussi nulle. À l'inverse, $\varphi_=$ est opaque sur $\mathscr{A}_{P_2}$, donc LPO et LPSO sont toutes deux

nulles. D'autre part, l'observation n'apporte pas de connaissance sur la véracité de $\varphi_=$. En effet, que l'on connaisse ou non les k derniers bits de H, on ne peut savoir si les $7k$ premiers bits sont tous nuls (seuls cas pour lesquels on ait $H = L$). Ainsi les mesures basées sur l'information ne détectent pas de fuite (et valent donc 1).

La mesure de la faille de sécurité de P_1 par la LPSO montre que l'observation permet de déduire exactement la valeur de φ : lorsque $L = -1$, on sait que $L \neq H$ et donc que $\varphi_=$ est faux. Cependant, lorsque φ est faux, l'observateur ne connaît pas la valeur de H. En conséquence, la LPO permet de mesurer seulement les cas où $H = L$ et donc que le secret est divulgué ; ceux-ci se produisent avec probabilité $\frac{1}{8}$.

Remarquons que seules les mesures de robustesse de la sécurité de P_2 dépendent de la valeur du paramètre k. En effet, seulement dans ce cas $\varphi_=$ est vrai avec une probabilité plus faible si k augmente. Donc $\varphi_=$ est moins facilement détecté par l'observation, faisant augmenter la RPO. Cependant, puisque $\varphi_=$ est de plus en plus souvent faux, la RPSO, quant à elle, décroît avec k.

Ainsi les mesures d'opacité donnent une indication de la sécurité du système. Cependant elles ne peuvent porter que sur un prédicat, et on ne peut mesurer finement la connaissance de l'observateur sur le système qu'en utilisant plusieurs prédicats. Par exemple, ici, un prédicat pour chaque bit de H permettrait de mesurer complètement la connaissance d'un observateur, mais cela ne fournit pas une mesure unifiée de la sécurité du système.

Le protocole *Crowds*

Le protocole *Crowds* est un protocole d'anonymat introduit par Reiter et Rubin [RR98]. Une version cryptée de ce protocole, *The Onion Router* (TOR) a connu un grand succès avec l'essor d'internet, et des besoins de préserver la vie privée des internautes. Le protocole crowds est aussi couramment utilisé dans l'étude des propriétés d'anonymat dans les systèmes probabilistes [CPP08, APvRS10].

Principe de fonctionnement. Lorsqu'un utilisateur veut envoyer un message (par exemple une requête HTTP) à un serveur sans que celui-ci puisse connaître l'identité de l'émetteur, l'utilisateur fait passer le message à travers une foule de n utilisateurs. À cet effet, le message est d'abord envoyé vers un utilisateur de la foule (potentiellement l'émetteur lui même) choisi aléatoirement et de manière uniforme. Cet utilisateur peut alors soit envoyer le message au serveur (avec probabilité $1 - q$) soit le renvoyer à un utilisateur de la foule (avec probabilité q) qui réitèrera cette procédure jusqu'à ce que le message soit envoyé au serveur. L'éventuel message retour (par exemple une page HTML) prendra le même chemin en sens inverse, depuis le serveur jusqu'à l'émetteur initial.

Sous ces hypothèses, le système fournit de bonnes propriétés d'anonymat : du point de vue du serveur, aucun utilisateur n'a une plus grande probabilité d'être l'émetteur. Le calcul des mesures d'opacité robustes [11], dans la suite de cette section, montre en effet qu'il est souvent proche de 1. On peut cependant supposer que c utilisateurs de la foule sont en réalité corrompus et donnent au serveur l'identité de celui qui leur a envoyé un message. Du point de vue de l'anonymat, ils agissent comme une

11. Le système étant opaque sauf dans certains cas extrêmes, on ne calcule pas ici les valeurs de la LPO ou de la LPSO.

copie du serveur. Les mesures d'opacité vont donc chercher à mesurer l'anonymat en fonction du nombre d'utilisateur corrompus (c) et du nombre d'utilisateurs au total dans la foule (n).

On suppose dans la suite que $n \geq c+1$, c'est-à dire qu'il y a au moins un utilisateur honnête, et que $n > 1$, c'est-à dire que la foule n'est pas réduite à un seul utilisateur. Dans ce cas trivial, il n'y a évidemment pas d'anonymat, et le protocole se réduit à une communication directe avec le serveur.

Ce protocole peut être modélisé par l'automate probabiliste de la figure 4.15. Chaque utilisateur est identifié par un numéro : les utilisateurs honnêtes étant numérotés de 1 à $n-c$ et les utilisateurs corrompus de $n-c+1$ à n. On suppose que les utilisateurs corrompus n'initient pas de communication. L'utilisateur qui initie la communication est donc choisi uniformément parmi les $n-c$ utilisateurs honnêtes.

L'identité de l'initiateur étant le secret à protéger, on considère $n-c$ prédicats tels que φ_i est vrai lorsque i a initié la communication. Bien que la mesure de l'opacité de plusieurs prédicats ne donne *a priori* pas une mesure globale de la sécurité du système, dans ce cas chaque utilisateur est semblable, et tous les φ_i ont donc la même opacité. L'observation est ce que connaît le serveur, c'est-à dire l'identité de l'utilisateur honnête ayant envoyé le message soit au serveur soit à un utilisateur corrompu. Ainsi, φ_i est vrai si le deuxième état de l'exécution est i', tandis que la fonction d'observation $\mathcal{O}_{\text{serveur}}$ retourne l'avant-dernier état de l'exécution.

Calcul des probabilités. On note $[i \rightsquigarrow]$ l'évènement « l'uitilisateur i initie la communication », qui est équivalent à $[\varphi_i = 1]$. De même, $[\rightsquigarrow i]$ est l'évènement « l'utilisateur i est détecté par le serveur », c'est-à dire $[\mathcal{O}_{\text{serveur}} = i]$. On note $[\neg i \rightsquigarrow]$ (respectivement $[\neg \rightsquigarrow i]$) l'évènement « un utilisateur autre que i initie la communication (respectivement est détecté) ». On agrège les occurrences multiple du symbole $\rightsquigarrow$; ainsi $[i \rightsquigarrow] \wedge [\rightsquigarrow j]$ devient $[i \rightsquigarrow j]$. On utilise dans la suite le symbole de Kronecker $\delta_{i,j} = 1$ si $i = j$ et 0 sinon.

Toutes les probabilités $\mathbf{P}(i \rightsquigarrow j)$ peuvent être calculées par la méthode présentée en section 4.4. Par exemple, $\mathbf{P}(1 \rightsquigarrow (n-c))$, la probabilité que le premier utilisateur initie une communication alors que le dernier utilisateur honnête est détecté, peut être calculée à partir de l'automate sous-stochastique $\mathscr{A}^{n,c}_{\text{crowds}} \,||\, \mathscr{A}_{1 \rightsquigarrow (n-c)}$, représenté figure 4.16.

Le système linéaire associé est représenté table 4.6, où la ligne X_S correspond à l'état *Serveur*. Ce système peut se simplifier en

$$\begin{cases} X_0 &= \frac{1}{q(n-c)} \cdot X_1 \\ X_1 &= q\left(\frac{n-c-1}{n} \cdot X_1 + \frac{1}{n} \cdot X_{n-c}\right) \\ X_{n-c} &= 1-q+X_1+\frac{q \cdot c}{n} \;=\; 1-\frac{q(n-c)}{n}+X_1 \end{cases}$$

Ce qui donne, pour X_1 :

$$\begin{aligned} X_1 &= \frac{q}{n}(n-c)\cdot X_1 + \frac{q}{n}\left(1-\frac{q(n-c)}{n}\right) \\ \frac{n}{q}\cdot X_1 &= (n-c)\cdot X_1 + 1 - \frac{q(n-c)}{n} \\ X_1\left(\frac{n}{q}-(n-c)\right) &= \frac{n-q(n-c)}{n} \\ X_1 &= \frac{q}{n} \end{aligned}$$

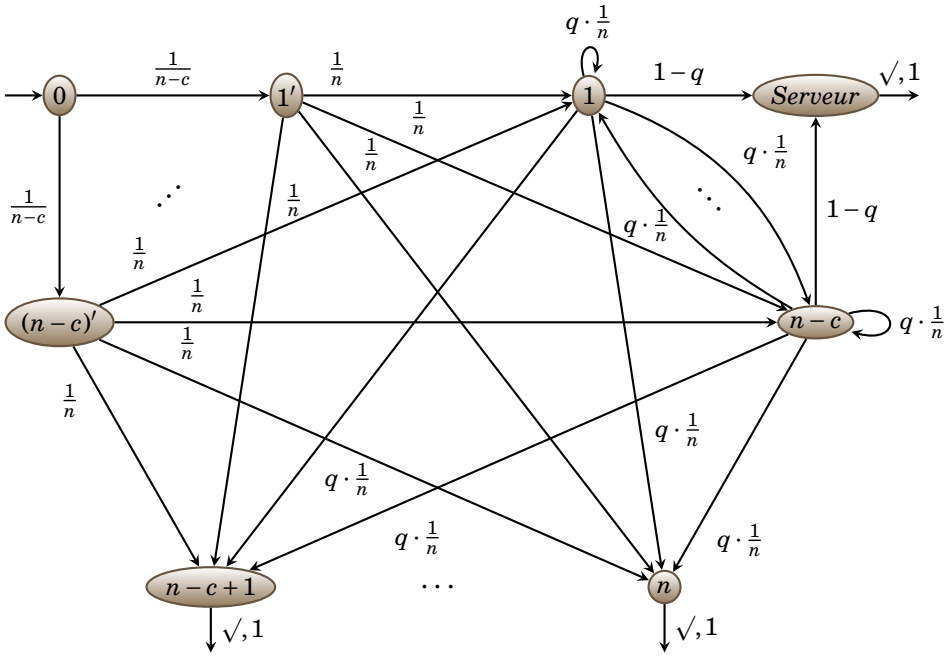


FIGURE 4.15: Automate probabiliste $\mathscr{A}^{n,c}_{\text{crowds}}$ modélisant le protocole crowds avec n utilisateurs, dont c sont corrompus.

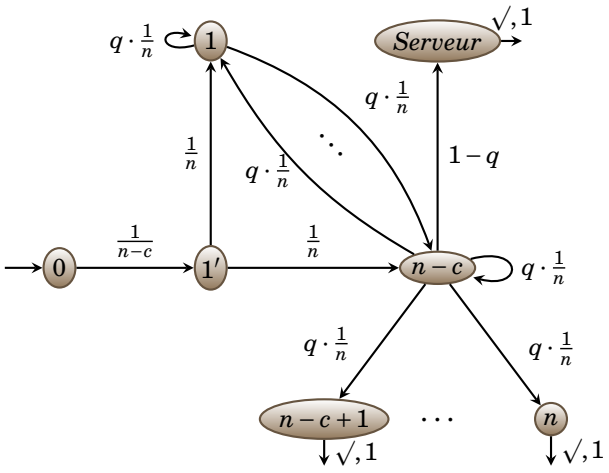


FIGURE 4.16: Automate sous-stochastique $\mathscr{A}^{n,c}_{\text{crowds}} \,||\, \mathscr{A}_{1 \rightsquigarrow (n-c)}$ correspondant aux exécutions de $\mathscr{A}^{n,c}_{\text{crowds}}$ où 1 initie le protocole $(n-c)$ est détecté.

$$\begin{cases} X_0 &= \frac{1}{n-c}\cdot X_{1'} \\ X_{1'} &= \sum_{i=1}^{n-c}\frac{1}{n}\cdot X_i \\ X_1 &= \sum_{i=1}^{n-c}\frac{q}{n}\cdot X_i \\ &\vdots \\ X_{n-c-1} &= \sum_{i=1}^{n-c}\frac{q}{n}\cdot X_i \\ X_{n-c} &= (1-q)\cdot X_S + \sum_{i=1}^{n}\frac{q}{n}\cdot X_i \\ X_{n-c+1} &= 1 \\ &\vdots \\ X_n &= 1 \\ X_S &= 1 \end{cases}$$

TABLE 4.6: Système linéaire associé à $\mathscr{A}^{n,c}_{\text{crowds}} \,||\, \mathscr{A}_{1\rightsquigarrow(n-c)}$ (figure 4.16).

On obtient donc $X_i = \frac{q}{n}$ pour $i \in \{1,\dots n-c-1\}$, $X_{n-c} = 1 - \frac{q\cdot(n-c-1)}{n}$, $X_{1'} = \frac{1}{n}$, et $X_0 = \frac{1}{(n-c)\cdot n}$. On a donc $\mathbf{P}(1 \rightsquigarrow (n-c)) = \frac{1}{(n-c)\cdot n}$.

Dans le cas de ce protocole, un raisonnement sur les symétries du système permet de calculer les autres probabilités $\mathbf{P}(i \rightsquigarrow j)$. On remarque que la probabilité qu'un message aille directement de l'initiateur au serveur ou à un utilisateur corrompu est de $\frac{c}{n}$: en effet ce cas ne se produit que lorsque le message va directement de l'initiateur au serveur. Si un utilisateur honnête est choisi par l'initiateur, avec probabilité $\frac{n-c}{n}$, alors le chemin vers le serveur sera plus long. De plu,s par symétrie, chaque utilisateur honnête a la même probabilité d'être initiateur, et d'être détecté, donc $\mathbf{P}(i \rightsquigarrow) = \mathbf{P}(\rightsquigarrow i) = \frac{1}{n-c}$.

L'évènement $i \rightsquigarrow j$ se produit lorsque i est l'initiateur (probabilité $\frac{1}{n-c}$) et soit (1) si $i = j$ et i choisit un utilisateur corrompu pour transmettre son message, ou (2) si un utilisateur honête est choisi et j envoie le message au serveur ou à un utilisateur corrompu (le chemin entre la foule des utilisateurs honnêtes avant j n'a pas d'importance). Donc

$$\mathbf{P}(i \rightsquigarrow j) \;=\; \frac{1}{n-c}\cdot\left(\delta_{ij}\cdot\frac{c}{n} + \frac{1}{n-c}\cdot\frac{n-c}{n}\right) \;=\; \frac{1}{n-c}\cdot\left(\delta_{ij}\cdot\frac{c}{n} + \frac{1}{n}\right).$$

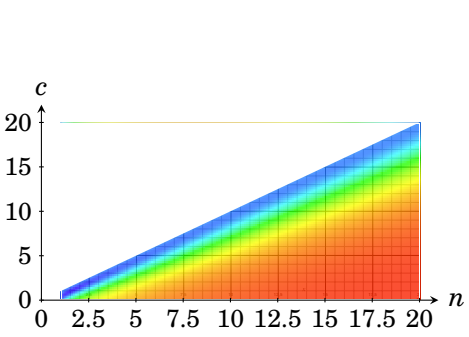
Le cas où i n'est pas l'initiateur découle de ces probabilités :

$$\begin{aligned} \mathbf{P}(\neg i \rightsquigarrow j) &= \sum_{\substack{k=1\\k\neq i}}^{n-c} \mathbf{P}(k \rightsquigarrow j) \\ \mathbf{P}(\neg i \rightsquigarrow j) &= \frac{1}{n-c}\cdot\left((1-\delta_{ij})\cdot\frac{c}{n} + \frac{n-c-1}{n}\right) \end{aligned}$$

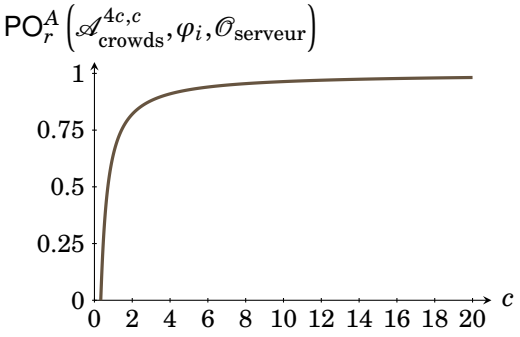
Les probabilités conditionelles sont donc :

$$\begin{aligned} \mathbf{P}(i \rightsquigarrow \,|\, \rightsquigarrow j) &= \frac{\mathbf{P}(i \rightsquigarrow j)}{\mathbf{P}(\rightsquigarrow j)} \;=\; \delta_{ij}\cdot\frac{c}{n} + \frac{1}{n} \\ \mathbf{P}(\neg i \rightsquigarrow \,|\, \rightsquigarrow j) &= \frac{\mathbf{P}(\neg i \rightsquigarrow j)}{\mathbf{P}(\rightsquigarrow j)} \;=\; (1-\delta_{ij})\cdot\frac{c}{n} + \frac{n-c-1}{n} \end{aligned}$$

Curieusement, ces probabilités ne dépendent pas de q. Cela provient du fait que l'attaquant du modèle original était soit le serveur, soit les utilisateurs corrompus, mais pas les deux.



(a) Évolution de $\mathsf{PO}_r^A\left(\mathscr{A}_{\text{crowds}}^{n,c}, \varphi_i, \mathscr{O}_{\text{serveur}}\right)$ en fonction de n et de c. Le rouge indique une valeur proche de 1 tandis que le bleu indique une valeur proche de 0.



(b) Évolution de $\mathsf{PO}_r^A\left(\mathscr{A}_{\text{crowds}}^{n,c}, \varphi_i, \mathscr{O}_{\text{serveur}}\right)$ lorsque $n = 4c$, en fonction de c.

FIGURE 4.17: Évolution de la RPO pour le protocole crowds.

Calcul de la RPO. À partir des probabilités calculées ci-dessus, on obtient une valeur pour la RPO en fonction de n et c.

$$\begin{aligned}
\frac{1}{\mathsf{PO}_r^A\left(\mathscr{A}_{\text{crowds}}^{n,c}, \varphi_i, \mathscr{O}_{\text{serveur}}\right)} &= \sum_{j=1}^{n-c} \mathbf{P}(\rightsquigarrow j) \cdot \frac{1}{\mathbf{P}(\neg i \rightsquigarrow \mid \rightsquigarrow j)} \\
&= (n-c-1) \cdot \frac{1}{n-c} \cdot \frac{n}{n-1} + \frac{1}{n-c} \cdot \frac{n}{n-c-1} \\
&= \frac{n}{n-c}\left(\frac{n-c-1}{n-1} + \frac{1}{n-c-1}\right) \\
\frac{1}{\mathsf{PO}_r^A\left(\mathscr{A}_{\text{crowds}}^{n,c}, \varphi_i, \mathscr{O}_{\text{serveur}}\right)} &= \frac{n \cdot (n^2 + c^2 - 2nc - n + 2c)}{(n-c) \cdot (n-1) \cdot (n-c-1)}
\end{aligned}$$

D'où

$$\mathsf{PO}_r^A\left(\mathscr{A}_{\text{crowds}}^{n,c}, \varphi_i, \mathscr{O}_{\text{serveur}}\right) = \frac{(n-c) \cdot (n-1) \cdot (n-c-1)}{n \cdot (n^2 + c^2 - 2nc - n + 2c)}.$$

Cette valeur tend vers 1 lorsque n tend vers $+\infty$ (à c fixé). Cela confirme l'intuition que plus la foule est grande, moins l'initiateur est détectable. L'évolution de la RPO est représentée figure 4.17(a).

Cette intuition reste vraie même lorsque le nombre d'utilisateur corrompus croît, pour peu que leur proportion dans la foule reste constante. Par exemple, si l'on fixe $n = 4c$, on a

$$\mathsf{PO}_r^A\left(\mathscr{A}_{\text{crowds}}^{4c,c}, \varphi_i, \mathscr{O}_{\text{serveur}}\right) = \frac{(4c-1) \cdot (9c-3)}{4c \cdot (9c-2)}$$

qui tend aussi vers 1 lorsque n tend vers $+\infty$ (voir le graphe figure 4.17(b)).

Lorsqu'il n'y a aucun utilisateur corrompu, alors

$$\mathsf{PO}_r^A\left(\mathscr{A}_{\text{crowds}}^{n,c}, \varphi_i, \mathscr{O}_{\text{serveur}}\right) = \frac{n-1}{n},$$

qui a beau être proche de 1, ne l'atteint jamais, car φ_i n'est jamais le prédicat toujours faux. En effet, même si un utilisateur a peu de chance d'être détecté en étant l'initiateur, cette chance n'est pas nulle. Ce phénomène est déjà présent dans le modèle de [RR98], qui distingue le cas où l'utilisateur est « hors de tout soupçon » (pas plus

suceptible qu'un autre d'être l'initiateur) et le cas de l'« anonymat absolu » (le serveur ne sait même pas s'il y a eu communication).

Calcul de la RPSO. Les probabilités obtenues permettent le calcul des vulnérabilités conditionnelles $V(\varphi_i|\rightsquigarrow j)$ et donc de la RPSO.

$$\text{Si } i \neq j,\ V(\varphi_i|\rightsquigarrow j) \ = \ \max\left(\frac{1}{n}, \frac{n-1}{n}\right) \ = \ \frac{1}{n}\cdot\max(1, n-1).$$

Puisque $n > 1$, on a $V(i \rightsquigarrow | \rightsquigarrow j) = \frac{n-1}{n}$.

$$\text{Si } i = j,\ V(\varphi_i|\rightsquigarrow i) \ = \ \max\left(\frac{c+1}{n}, \frac{n-c-1}{n}\right) \ = \ \frac{1}{n}\cdot\max(c+1, n-c-1).$$

La vulnérabilité de la classe d'observation où l'initiateur i est détecté dépend de la proportion d'utilisateurs corrompus dans la foule. On a en effet $V(i \rightsquigarrow | \rightsquigarrow i) = \frac{c+1}{n}$ si et seulement si $n \leq 2(c+1)$. On distingue alors les deux cas.

Si $n \leq 2(c+1)$. Le message a initialement plus de chance d'être envoyé soit à un utilisateur corrompu, soit à l'initiateur, qu'à quelqu'autre utilisateur honnête :

$$\begin{aligned}\sum_{j=1}^{n-c} \mathbf{P}(\rightsquigarrow j)\cdot\log(1-V(\varphi_i|\rightsquigarrow j)) &= \frac{1}{n-c}\cdot\left((n-c-1)\cdot\log\left(\frac{1}{n}\right)+\log\left(\frac{n-c-1}{n}\right)\right)\\ &= \frac{1}{n-c}\cdot(\log(n-c-1)-(n-c)\cdot\log(n))\\ &= \frac{\log(n-c-1)}{n-c}-\log(n)\end{aligned}$$

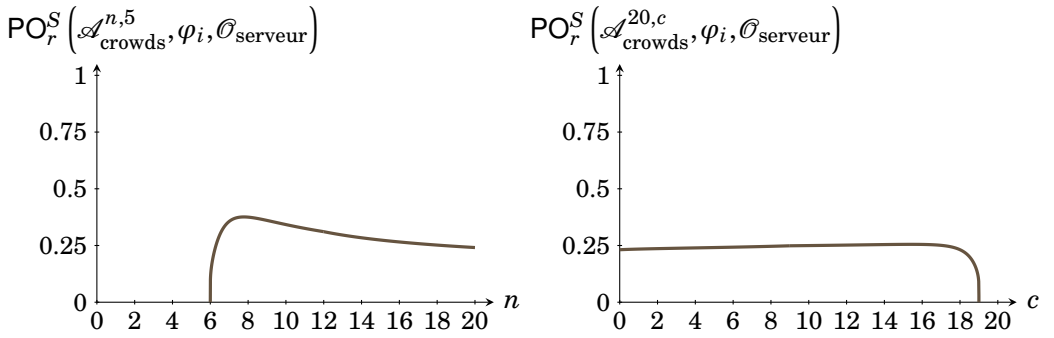
$$\text{D'où}\quad \mathsf{PO}_r^S\left(\mathscr{A}^{n,c}_{\text{crowds}}, \varphi_i, \mathscr{O}_{\text{serveur}}\right) \ = \ \frac{1}{\log(n)-\frac{\log(n-c-1)}{n-c}}$$

Si $n \geq 2(c+1)$. Le message a cette fois plus de chance d'être envoyé à un utilisateur honnête n'étant pas l'initiateur :

$$\begin{aligned}\sum_{j=1}^{n-c} \mathbf{P}(\rightsquigarrow j)\cdot\log(1-V(\varphi_i|\rightsquigarrow j)) &= \frac{1}{n-c}\cdot\left((n-c-1)\cdot\log\left(\frac{1}{n}\right)+\log\left(\frac{c+1}{n}\right)\right)\\ &= \frac{1}{n-c}\cdot(\log(c+1)-(n-c)\cdot\log(n))\\ &= \frac{\log(c+1)}{n-c}-\log(n)\end{aligned}$$

$$\text{D'où}\quad \mathsf{PO}_r^S\left(\mathscr{A}^{n,c}_{\text{crowds}}, \varphi_i, \mathscr{O}_{\text{serveur}}\right) \ = \ \frac{1}{\log(n)-\frac{\log(c+1)}{n-c}}$$

L'évolution de la RPSO est représentée sur la figure 4.18 : en fonction de n pour $c = 5$ (figure 4.18(a)) et en fonction de c pour $n = 20$ (figure 4.18(b)). On peut remarquer que la RPSO *décroît* lorsque n croît. En effet, lorsque n croît, le prédicat φ_i est très rarement vrai, et ce au sein de chaque classe d'observation. Ainsi l'attaquant sait quasi-sûrement que i n'est *pas* l'initiateur. La faible opacité symétrique trahit donc ici plus le fait que i est « hors de tout soupçon », plutôt qu'une potentielle faille de sécurité. Cela montre aussi que le choix du prédicat et du type d'opacité que l'on cherche à mesurer (asymétrique ou symétrique) est important. L'opacité symétrique mesure une fuite d'information seulement lorsque le prédicat et sa négation sont des secrets à protéger de l'observateur.



(a) Évolution de $\mathsf{PO}_r^S\left(\mathscr{A}_{\text{crowds}}^{n,c},\varphi_i,\mathscr{O}_{\text{serveur}}\right)$ en fonction de n lorsque $c=5$.

(b) Évolution de $\mathsf{PO}_r^S\left(\mathscr{A}_{\text{crowds}}^{n,c},\varphi_i,\mathscr{O}_{\text{serveur}}\right)$ en fonction de c lorsque $n=20$.

FIGURE 4.18: Évolution de la RPSO pour le protocole crowds.

Calcul de la EPSO. Le calcul des entropies et entropies conditionnelles découle de celui des probabilités :

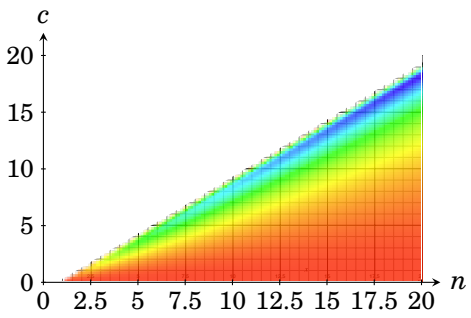
$$\begin{aligned}
-H(\varphi_i|\mathscr{O}_{\text{serveur}}) &= \sum_{j=1}^{n-c}\left(\mathbf{P}(i\rightsquigarrow j)\cdot\log(\mathbf{P}(i\rightsquigarrow|\rightsquigarrow j))+\mathbf{P}(\neg i\rightsquigarrow j)\cdot\log(\mathbf{P}(\neg i\rightsquigarrow|\rightsquigarrow j))\right)\\
&= \mathbf{P}(i\rightsquigarrow i)\cdot\log(\mathbf{P}(i\rightsquigarrow|\rightsquigarrow i))+\mathbf{P}(\neg i\rightsquigarrow i)\cdot\log(\mathbf{P}(\neg i\rightsquigarrow|\rightsquigarrow i))\\
&\quad+\sum_{\substack{j=1\\ j\neq i}}^{n-c}\left(\mathbf{P}(i\rightsquigarrow j)\cdot\log(\mathbf{P}(i\rightsquigarrow|\rightsquigarrow j))+\mathbf{P}(\neg i\rightsquigarrow j)\cdot\log(\mathbf{P}(\neg i\rightsquigarrow|\rightsquigarrow j))\right)\\
&= \frac{1}{n-c}\cdot\left(\frac{c+1}{n}\cdot\log(\frac{c+1}{n})+\frac{n-c-1}{n}\cdot\log\left(\frac{n-c-1}{n}\right)\right.\\
&\qquad\left.+\frac{n-c-1}{n}\cdot\log\left(\frac{1}{n}\right)+\frac{(n-c-1)\cdot(n-1)}{n}\cdot\log\left(\frac{n-1}{n}\right)\right)\\
-H(\varphi_i|\mathscr{O}_{\text{serveur}}) &= \frac{1}{n-c}\cdot\left(\frac{(n-c-1)\cdot(n-1)}{n}\cdot\log(n-1)\right.\\
&\qquad\left.+\frac{n-c-1}{n}\cdot\log(n-c-1)+\frac{c+1}{n}\cdot\log(c+1)\right)-\log(n)
\end{aligned}$$

Par ailleurs

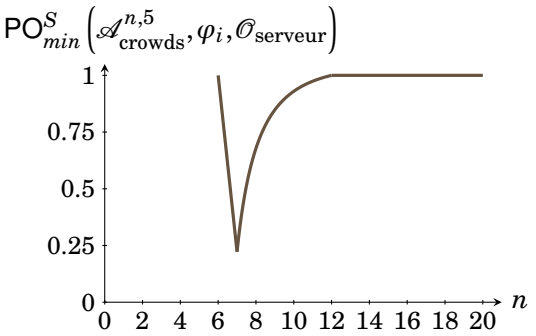
$$\begin{aligned}
H(\varphi_i) &= \mathbf{P}(i\rightsquigarrow)\cdot\log(\mathbf{P}(i\rightsquigarrow))+\mathbf{P}(\neg i\rightsquigarrow)\cdot\log(\mathbf{P}(\neg i\rightsquigarrow))\\
&= -\frac{1}{n-c}\cdot\log\left(\frac{1}{n-c}\right)-\frac{n-c-1}{n-c}\cdot\log\left(\frac{n-c-1}{n-c}\right)\\
H(\varphi_i) &= \log(n-c)-\frac{n-c-1}{n-c}\cdot\log(n-c-1)
\end{aligned}$$

D'où

$$\begin{aligned}
\mathsf{PO}_{en}^S\left(\mathscr{A}_{\text{crowds}}^{n,c},\varphi_i,\mathscr{O}_{\text{serveur}}\right) &= 1+\log(n)-\log(n-c)+\frac{n-c-1}{n-c}\cdot\log(n-c-1)\\
&\quad-\frac{1}{n-c}\cdot\left(\frac{(n-c-1)\cdot(n-1)}{n}\cdot\log(n-1)\right.\\
&\qquad\left.+\frac{n-c-1}{n}\cdot\log(n-c-1)+\frac{c+1}{n}\cdot\log(c+1)\right)
\end{aligned}$$



(a) Évolution de $\mathsf{PO}^{S}_{en}\left(\mathscr{A}^{n,c}_{\text{crowds}},\varphi_i,\mathscr{O}_{\text{serveur}}\right)$ en fonction de n et de c. Le rouge indique une valeur proche de 1 tandis que le bleu indique une valeur proche de 0.



(b) Évolution de $\mathsf{PO}^{S}_{min}\left(\mathscr{A}^{n,c}_{\text{crowds}},\varphi_i,\mathscr{O}_{\text{serveur}}\right)$ en fonction de n lorsque $c=5$.

FIGURE 4.19: Évolution de la EPSO et de la minPSO pour le protocole crowds.

Les variations de la EPSO en fonction de n et de c sont représentées figure 4.19(a). Ici, dans le cas où $c=0$, c'est-à dire lorsqu'aucun utilisateur n'est corrompu, la EPSO vaut 1. En effet, chaque utilisateur a la même probabilité d'être l'initiateur, quelque soit l'observation. Ceci confirme donc l'intuition que ce protocole fournit de l'anonymat car l'attaquant n'*apprend* rien de son observation.

Calcul de la minPSO. Les vulnérabilités calculées pour la RPSO permettent de calculer la minPSO pour le protocole crowds. Tout d'abord, si $n > c+1$

$$\begin{aligned} V(\varphi_i) &= \frac{n-c-1}{n-c} \\ \text{D'où}\quad H_\infty(\varphi_i) &= \log(n-c)-\log(n-c-1). \end{aligned}$$

D'autre part,

$$\begin{aligned} V(\varphi_i|\mathscr{O}_{\text{serveur}}) &= \sum_{j=1}^{n-c}\mathbf{P}(\rightsquigarrow j)\cdot V(\varphi_i|\rightsquigarrow j) \\ &= \mathbf{P}(\rightsquigarrow i)\cdot V(\varphi_i|\rightsquigarrow i)+\sum_{\substack{j=1\\ j\neq i}}^{n-c}\mathbf{P}(\rightsquigarrow j)\cdot V(\varphi_i|\rightsquigarrow j) \\ &= \frac{1}{n-c}\cdot\frac{1}{n}\cdot\max(c+1,n-c-1)+\frac{n-c-1}{n-c}\cdot\frac{n-1}{n} \end{aligned}$$

(car $n>1$). On a encore une fois une disjonction de cas selon la proportion d'utilisateurs corrompus dans la foule.
Si $n \leq 2(c+1)$. On a dans ce cas

$$\begin{aligned} V(\varphi_i|\mathscr{O}_{\text{serveur}}) &= \frac{c+1+(n-c-1)(n-1)}{n\cdot(n-c)} \\ \text{D'où}\quad H_\infty(\varphi_i|\mathscr{O}_{\text{serveur}}) &= \log(n)+\log(n-c)-\log\left(n^2-nc-2n+2c+1\right). \end{aligned}$$

Donc

$$\mathsf{PO}^{S}_{min}\left(\mathscr{A}^{n,c}_{\text{crowds}},\varphi_i,\mathscr{O}_{\text{serveur}}\right) = 1+\log(n-c-1)+\log(n)-\log\left(n^2-nc-2n+2c+1\right)$$

Si $n \geq 2(c+1)$. On a alors

$$\begin{aligned} V(\varphi_i|\mathcal{O}_{\text{serveur}}) &= \frac{n-c-1}{n-c} \\ V(\varphi_i|\mathcal{O}_{\text{serveur}}) &= V(\varphi_i). \end{aligned}$$

Donc

$$\mathsf{PO}^S_{min}\left(\mathcal{A}^{n,c}_{\text{crowds}}, \varphi_i, \mathcal{O}_{\text{serveur}}\right) = 1.$$

Comme dans le cas de la RPSO, la vulnérabilité est, lorsque n est suffisament grand, la probabilité que i ne soit *pas* l'initiateur, même lorsque celui-ci est détecté par le serveur.

Le cas où $n = c+1$ doit être traité à part. En effet, dans ce cas, $V(\varphi_i) = \frac{1}{n-c}$ et

$$V(\varphi_i|\mathcal{O}_{\text{serveur}}) = V(\varphi_i| \rightsquigarrow i) = \frac{1}{n-c}$$

donc $\mathsf{PO}^S_{min}\left(\mathcal{A}^{n,n-1}_{\text{crowds}}, \varphi_i, \mathcal{O}_{\text{serveur}}\right) = 1$. Cette valeur particulière s'explique par le fait que puisqu'il y a un seul utilisateur honnête, c'est nécessairement lui qui initie la communication et l'observation n'apporte rien : l'attaquant connaît déjà tout. L'ensemble des valeurs prises par la minPSO dans le cas où $c = 5$ est représenté figure 4.19(b).

Calcul de la WCPSO. Une fois encore, on doit distinguer le cas où il y a un seul utilisateur honnête du cas où il y en a plusieurs. Si $n > c+1$:

$$\begin{aligned} \mathsf{PO}^S_{wc}\left(\mathcal{A}^{n,c}_{\text{crowds}}, \varphi_i, \mathcal{O}_{\text{serveur}}\right) &= 1 - \max_{j,k} \left|\mathbf{P}(\varphi_i| \rightsquigarrow j) - \mathbf{P}(\varphi_i| \rightsquigarrow k)\right| \\ &= 1 - \mathbf{P}(\varphi_i| \rightsquigarrow i) - \mathbf{P}(\varphi_i| \rightsquigarrow j) \text{ pour n'importe quel } j \neq i \\ &= 1 - \left(\frac{c+1}{n} - \frac{1}{n}\right) \\ \mathsf{PO}^S_{wc}\left(\mathcal{A}^{n,c}_{\text{crowds}}, \varphi_i, \mathcal{O}_{\text{serveur}}\right) &= \frac{n-c}{n} \end{aligned}$$

Donc la WCPSO est (dans ce cas) la proportion d'utilisateurs honnêtes dans la foule.

Si $n = c+1$, alors

$$\mathsf{PO}^S_{wc}\left(\mathcal{A}^{n,c}_{\text{crowds}}, \varphi_i, \mathcal{O}_{\text{serveur}}\right) = 1 - \max_{j,k} \left|\mathbf{P}(\varphi_i| \rightsquigarrow j) - \mathbf{P}(\varphi_i| \rightsquigarrow k)\right| = 1$$

car il n'y a pas non plus d'information transmise par l'observation, tout étant déjà connu.

4.5.3 Discussion

On peut tirer des cas d'études précédents que ces mesures permettent en général de donner une bonne idée de la sécurité. Il faut cependant utiliser chaque métrique pour ce qu'elle est, afin que la mesure ait réellement du sens. De plus, le calcul des valeurs des mesures en fonction des paramètres du modèle permet de connaître l'évolution de la sécurité avec ceux-ci. Une autre utilisation possible de ces mesures est d'évaluer l'évolution de la sécurité lors du raffinement du modèle [MMM10]. Les liens entre l'opacité et la diagnosticabilité [Lin11] sont aussi une piste d'application de ces mesures.

Tout d'abord, seules les mesures LPO et RPO (et leurs versions symétriques LPSO et RPSO) ont un lien fort avec l'opacité classique. Les mesures basées sur la théorie de

l'information (EPSO, minPSO, WCPSO) évaluent quant à elles le pouvoir de l'observation vis-à-vis du prédicat. Ainsi ces dernières mesures permettent d'évaluer ce qui est *appris* et non ce qui est *su* par l'attaquant. Un prédicat non-opaque car souvent vrai n'est donc pas détecté comme une fuite d'information, car l'insécurité ne provient pas de l'observation, mais du prédicat.

Par ailleurs, le choix des prédicats influe beaucoup sur la pertinence de la mesure. En particulier, les mesures symétriques (RPSO et LPSO) ne représentent une réelle faiblesse de l'opacité que dans le cas où le secret et sa négation doivent rester secrets.

Enfin, les modèles où la sécurité dépend en réalité de l'opacité de plusieurs prédicats (comme dans le cas des programmes à affectation, où un prédicat par bit de H serait à considérer), marquent souvent les limites de l'opacité elle même, et donc des mesures définies ici qui la généralisent.

4.6 Introduction du non déterminisme

Le modèle utilisé jusqu'ici dans ce chapitre est celui d'un attaquant passif, qui se contente d'observer le système. Cette modélisation ne permet donc pas de rendre compte de certaines faiblesses du système. En particulier, les choix internes au système sont abstraits dans une distribution de probabilité. Mais si ceux-ci modélisent une action quelconque de l'environnement, il faut envisager le pire cas du point de vue de la sécurité de l'information, et non un cas moyen.

Un exemple d'un tel système est celui d'un cheval de Troie dans un ordinateur. Celui-ci peut faciliter la fuite d'informations confidentielles qui n'auraient pas eu lieu autrement. Ainsi, modéliser explicitement le non déterminisme du modèle permet de mesurer l'effet sur la sécurité d'un canal de communication entre le cheval de Troie à l'intérieur du système et l'observateur passif à l'extérieur.

Plus généralement, l'entité prenant les décisions au moment d'un choix non déterministe est appelé *ordonnanceur*. L'attaquant se divisant alors en deux agents, l'ordonnanceur à l'intérieur du système et l'observateur à l'extérieur (voir figure 4.20), à la manière de l'encodeur et du décodeur du chapitre 3. Dans ce cas, l'ordonnanceur influence le système en pondérant les choix non déterministes qui lui sont proposés à l'aide d'une distribution. Cette distribution peut très bien dépendre de tout le passé de l'exécution en cours.

4.6.1 Le modèle non déterministe

Le modèle utilisé élargit le modèle purement probabiliste en y introduisant du non déterminisme. Pour chaque état de l'automate, on n'associe plus une distribution, mais un ensemble de distributions [Seg95].

Définition 4.9 (Automate probabiliste non déterministe)**.** *Un* automate probabiliste non déterministe *(NPA) est un quadruplet* $\mathscr{A} = \langle S, Lab, \Delta, s_I \rangle$ *où :*

- S *est un ensemble fini d'états ;*
- Lab *est un ensemble fini d'étiquettes ;*
- $\Delta : S \to \mathscr{P}(\mathbb{D}((Lab \times S) \uplus \{\checkmark\}))$ *est une relation de transition non déterministe telle que pour* $s \in S$, $\Delta(s)$ *est un ensemble fini ;*
- s_I *est l'état initial.*

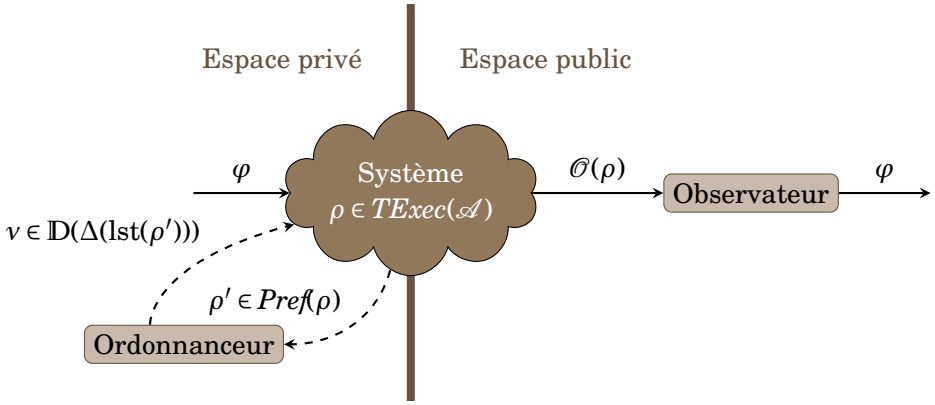


FIGURE 4.20: Canal dans un automate probabiliste non déterministe.

Contrairement au cas du non déterminisme dans les automates finis, ce n'est pas une seule distribution qui est choisie par l'ordonnanceur, mais plusieurs, en leur donnant un poids. Cette fonction de poids est donnée sous la forme d'une distribution sur les distributions possibles pour un état. Ainsi, si $\{\mu_1,\dots,\mu_n\}$ sont les distributions possibles, le choix de l'ordonnanceur est une distribution ν sur $\{1,\dots,n\}$.

Définition 4.10 (Ordonnanceur)**.** *Un* ordonnanceur *sur un NPA* $\mathscr{A} = \langle S, Lab, \Delta, s_I\rangle$ *est une fonction*

$$\sigma : Exec(\mathscr{A}) \to \mathbb{D}(\mathbb{D}((Lab \times S) \uplus \{\checkmark\}))$$

telle que pour toute exécution $\rho \in Exec(\mathscr{A})$, $\sigma(\rho)(\nu) > 0 \Rightarrow \nu \in \Delta(lst(\rho))$.

L'ensemble de tous les ordonnanceurs sur $\mathscr{A}$ est noté $Ord_{\mathscr{A}}$ ou simplement Ord lorsqu'aucune ambiguïté n'est possible sur l'automate considéré.

On peut remarquer que dans cette définition le choix de l'ordonnanceur peut dépendre de l'historique arbitrairement long de l'exécution. On peut se limiter aux ordonnanceurs *sans mémoire*, qui ne prennent en compte que l'état courant de l'automate : un ordonnanceur σ est dit sans mémoire s'il existe une fonction

$$\sigma' : S \to \mathbb{D}(\mathbb{D}((Lab \times S) \uplus \{\checkmark\}))$$

telle que pour toute exécution $\rho \in Exec(\mathscr{A})$, $\sigma(\rho) = \sigma'(\mathrm{lst}(\rho))$.

Comme il a été dit plus haut, l'ordonnanceur donne des poids aux distributions en sortie. Ainsi, un automate probabiliste non déterministe ordonnancé est un automate (purement) probabiliste, mais infini.

Définition 4.11 (NPA ordonnancé)**.** *L'automate probabiliste non déterministe* $\mathscr{A} = \langle S, Lab, \Delta, s_I\rangle$ *ordonnancé par* σ *est l'automate probabiliste* $\mathscr{A}_{/\sigma} = \langle Exec(\mathscr{A}), Lab, \Delta', \varepsilon\rangle$ *(dont l'ensemble d'états est infini) avec, pour* $\rho, \rho' \in Exec(\mathscr{A})$ *et* $a \in Lab$,

$$\Delta'(\rho)(a,\rho') = \sum_{\mu \in \Delta(lst(\rho))} \sigma(\rho)(\mu)\cdot\mu(a,q) \quad si\ \rho' = \rho \xrightarrow{a} q \quad et \quad \Delta'(\rho)(a,\rho') = 0 \quad autrement.$$

Remarquons cependant que pour un ordonnanceur sans mémoire, l'automate ordonnancé est isomorphe à un automate probabiliste fini.

Ainsi, on obtient une définition de toutes les mesures d'opacité sur ces systèmes non déterministes. En effet, la sécurité du système non déterministe est celle du système lorsqu'il est ordonnancé par le pire ordonnanceur possible.

Définition 4.12 (Opacité probabiliste non déterministe)**.** *Soit $\mathscr{A}$ un automate probabiliste non déterministe, $\mathscr{O} : TExec(\mathscr{A}) \to Obs$ une fonction d'observation et $\varphi \subseteq TExec(\mathscr{A})$ un prédicat. L'*opacité probabiliste non déterministe *est définie par :*

$$\widehat{PO^*}(\mathscr{A},\varphi,\mathscr{O}) = \max_{\sigma\in Ord} PO^*(\mathscr{A}_{/\sigma},\varphi,\mathscr{O}) \text{ pour } PO^* \in \left\{PO^A_\ell, PO^S_\ell\right\};$$

$$\widehat{PO^*}(\mathscr{A},\varphi,\mathscr{O}) = \min_{\sigma\in Ord} PO^*(\mathscr{A}_{/\sigma},\varphi,\mathscr{O}) \text{ pour } PO^* \in \left\{PO^A_r, PO^S_r, PO^S_{en}, PO^S_{min}, PO^S_{wc}\right\}.$$

4.6.2 Le pouvoir des ordonnanceurs

Bien que les ordonnanceurs aient *a priori* le pouvoir de prendre leurs décisions à partir de tout le début de l'exécution, cela produit des automates de taille infinie, et de plus rend impossible en général le calcul de l'ordonnanceur optimal (ici minimisant la sécurité). De plus, cet excès de pouvoir permet à l'attaquant des fuites d'information irréalistes [Kir10, HM10].

Certaines classes d'ordonnanceurs permettant le calcul des probabilités extrêmales avaient été définies par Giro et D'Argenio [GD09]. Lorsque cela est possible, les ordonnanceurs sont donnés comme les solutions à un problème d'optimisation linéaire. Cependant, les mesures d'opacité ne sont pas toujours linéaires ; par exemple la RPO se base sur une moyenne harmonique, la RPSO sur le logarithme de la vulnérabilité et la EPSO sur l'information mutuelle (bien qu'il soit parfois possible d'optimiser celle-ci [BCP08, AAP10, APvRS10]).

On montre tout d'abord que dans le cas de l'opacité probabiliste, on ne peut se restreindre aux ordonnanceurs sans mémoire, et ce même dans le cas de prédicats et de fonction d'observation « simples ». En effet, l'optimum de la mesure d'opacité n'est pas nécessairement atteint sur un ordonnanceur de ce type. On prouve ce résultat dans le cas de la RPO, mais il est adaptable à d'autres mesures d'opacité.

Théorème 4.9. *Il existe*
- *un automate probabiliste non déterministe $\mathscr{A}$,*
- *une fonction d'observation $\mathscr{O} : TExec(\mathscr{A}) \to Obs$ régulière* [12]*,*
- *un prédicat régulier $\varphi \subseteq TExec(\mathscr{A})$,*

tels que pour tout ordonnanceur sans mémoire *σ,*

$$\widehat{PO^A_r}(\mathscr{A},\varphi,\mathscr{O}) < PO^A_r(\mathscr{A}_{/\sigma},\varphi,\mathscr{O}).$$

Démonstration. On considère l'automate probabiliste non déterministe $\mathscr{A}_{14}$ de la figure 4.21. Les transition sur a et b vers q_1 (ainsi que le symbole ✓ vers la gauche) appartiennent à la même distribution, comme indiqué par l'arc les liant. De même, les transitions vers q_2 et le symbole ✓ vers la droite appartiennent à la même distribution.

Soit φ_{alt} le prédicat régulier qui est vrai lorsque l'exécution contient une alternance de a et de b (en commençant par a et à l'exclusion du mot vide), c'est-à-dire

$$\varphi_{\text{alt}} = tr^{-1}\left(\pi^{-1}_{\{a,b\}}\left((ab)^+ + (ab)^*a\right)\right).$$

La fonction d'observation $\mathscr{O}_{\text{last}}$ renvoie la dernière lettre de l'exécution avant ✓, c'est-à-dire o_1, o_2, ou ε si l'exécution est vide. Cette fonction, clairement régulière,

12. C'est-à-dire partitionnant $TExec(\mathscr{A})$ en un nombre fini d'ensembles réguliers, voir section 4.4.

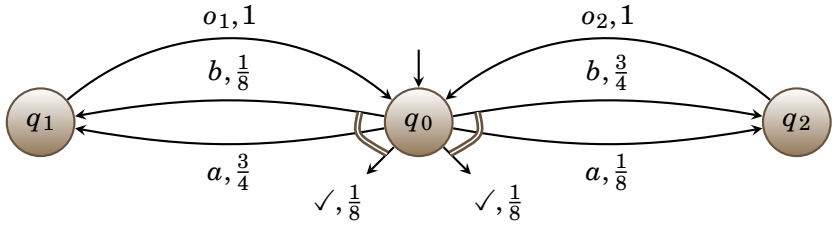


FIGURE 4.21: Automate probabiliste non déterministe $\mathscr{A}_{14}$. Les transitions appartenant à la même distribution sont liées.

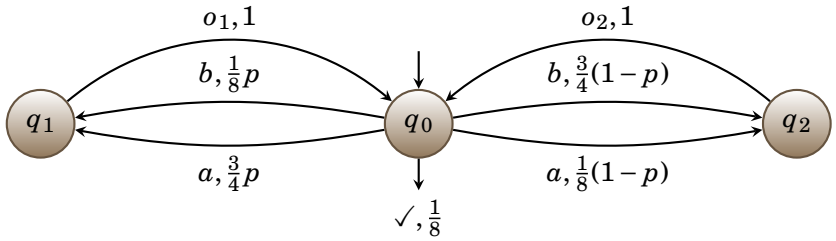


FIGURE 4.22: Automate probabiliste $\mathscr{A}_{14/\sigma_p}$, ordonnancement de $\mathscr{A}_{14}$ par σ_p.

permet à l'observateur de connaître le dernier des états q_1 ou q_2 visité lors de l'exécution.

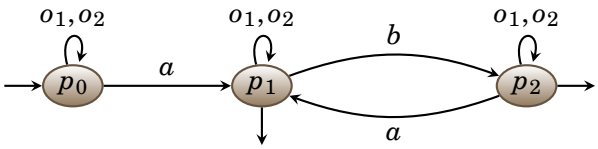
Puisque lorsque l'on va vers q_1, la lettre a est lue avec plus forte probabilité que la lettre b, et l'inverse dans le cas de q_2, les poids sur les deux distributions vont influencer la lettre suivante. Ainsi, avec de la mémoire, l'ordonnanceur peut faciliter l'alternance de a et de b, et donc φ_{alt}, ce que ne peut pas faire un ordonnanceur sans mémoire.

Soit un ordonnanceur sans mémoire σ_p, paramétré par le poids p donné à la distribution allant vers q_1. L'automate ordonnancé $\mathscr{A}_{14/\sigma_p}$ est représenté figure 4.22. Puisque le prédicat et la fonction d'observation sont réguliers, les différentes probabilités peuvent être calculées avec la méthode présentée en section 4.4. On abrège dans la suite par $[o]$ l'évènement $[\mathscr{O}_{\text{last}} = o]$, pour $o \in \{\varepsilon, o_1, o_2\}$. On obtient aisément :

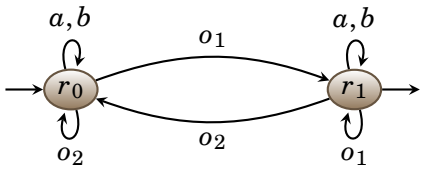
$$\mathbf{P}(\varepsilon) = \frac{1}{8} \qquad \mathbf{P}(o_1) = \frac{7}{8} \cdot p \qquad \mathbf{P}(o_2) = \frac{7}{8} \cdot (1-p) \qquad \mathbf{P}(\varphi_{\text{alt}}, \varepsilon) = 0$$

Le calcul de la probabilité $\mathbf{P}(\varphi_{\text{alt}}, o_1)$ sur l'automate $\mathscr{A}_{14/\sigma_p}$ s'obtient de la manière suivante. On écrit $\overline{p} = 1 - p$ pour plus de concision. On construit le produit synchronisé $\mathscr{A}_{14/\sigma_p} || \mathscr{A}_{\varphi_{\text{alt}}} || \mathscr{A}_{o_1}$, représenté figure 4.24. Le système linéaire de la table 4.7 est construit à partir de cet automate. Ce système est réduit en enlevant les informations redondantes, et parce que seule la valeur de $x_{000} = \mathbf{P}(\varphi_{\text{alt}}, o_1)$ est recherchée :

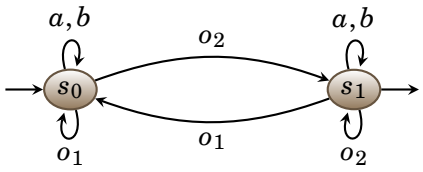
$$\left\{\begin{array}{lcl} x_{000} &=& \frac{1}{8}\overline{p}\ x_{010} + \frac{3}{4}p\ x_{011} \\ x_{010} &=& \frac{1}{8}p\ x_{021} + \frac{3}{4}\overline{p}\ x_{020} \\ x_{011} &=& \frac{1}{8} + \frac{3}{4}\overline{p}\ x_{020} + \frac{1}{8}p\ x_{021} \\ x_{020} &=& \frac{1}{8}\overline{p}\ x_{010} + \frac{3}{4}p\ x_{011} \\ x_{021} &=& \frac{1}{8} + \frac{1}{8}\overline{p}\ x_{010} + \frac{3}{4}p\ x_{011} \end{array}\right. \iff \left\{\begin{array}{lcl} x_{000} &=& \frac{1}{8}\overline{p}\ x_{010} + \frac{3}{4}p\ x_{011} \\ x_{010} &=& \frac{1}{8}p\ x_{021} + \frac{3}{4}\overline{p}\ x_{020} \\ x_{011} &=& \frac{1}{8} + x_{010} \\ x_{020} &=& x_{000} \\ x_{021} &=& \frac{1}{8} + x_{000} \end{array}\right.$$



(a) Automate fini déterministe $\mathscr{A}_{\varphi_{\text{alt}}}$



(b) Automate fini déterministe $\mathscr{A}_{o_1}$



(c) Automate fini déterministe $\mathscr{A}_{o_2}$

FIGURE 4.23: Automates pour le calcul de $\mathbf{P}(\varphi_{\text{alt}}, o_1)$ et $\mathbf{P}(\varphi_{\text{alt}}, o_2)$.

$$
\begin{cases}
x_{000} &= \frac{1}{8}\overline{p}\ x_{210} + \frac{3}{4}p\ x_{110} \\
x_{210} &= x_{010} \\
x_{110} &= x_{011} \\
x_{010} &= \frac{1}{8}p\ x_{120} + \frac{3}{4}\overline{p}\ x_{220} \\
x_{011} &= \frac{1}{8} + \frac{3}{4}\overline{p}\ x_{221} + \frac{1}{8}p\ x_{121} \\
x_{120} &= x_{021} \\
x_{220} &= x_{020} \\
x_{221} &= x_{020} \\
x_{121} &= x_{021} \\
x_{020} &= \frac{1}{8}\overline{p}\ x_{210} + \frac{3}{4}p\ x_{110} \\
x_{021} &= \frac{1}{8} + \frac{1}{8}\overline{p}\ x_{211} + \frac{3}{4}p\ x_{111} \\
x_{211} &= x_{010} \\
x_{111} &= x_{011}
\end{cases}
$$

TABLE 4.7: Système linéaire associé à l'automate $\mathscr{A}_{14/\sigma_p} || \mathscr{A}_{\varphi} \varphi_{\text{alt}} || \mathscr{A}_{o_1}$ de la figure 4.24. Le nom de chaque variable correspond à l'état de l'automate auquel elle correspond ; par exemple x_{210} correspond à l'état (q_2, p_1, r_0).

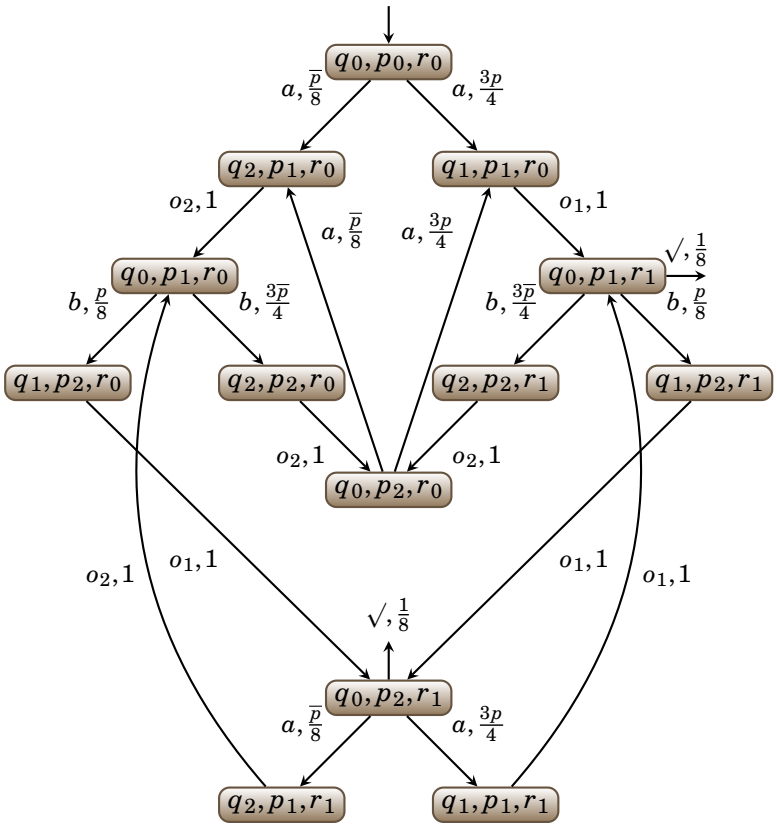


FIGURE 4.24: Automate sous-stochastique $\mathscr{A}_{14/\sigma_p}||\mathscr{A}_{\varphi_{\text{alt}}}||\mathscr{A}_{o_1}$.

On obtient donc :

$$\begin{cases} x_{000} &= \frac{1}{8}\overline{p}\ x_{010} + \frac{3}{4}p\left(\frac{1}{8} + x_{010}\right) \\ x_{010} &= \frac{1}{8}p\left(\frac{1}{8} + x_{000}\right) + \frac{3}{4}\overline{p}\ x_{000} \end{cases} \text{ donc } \begin{cases} x_{000} &= \frac{1}{8}\overline{p}\ x_{010} + \frac{3}{4}p\left(\frac{1}{8} + x_{010}\right) \\ x_{010} &= \frac{1}{64}p + \frac{1}{8}p\ x_{000} + \frac{3}{4}\overline{p}\ x_{000} \end{cases}$$

D'où

$$x_{000} = \frac{1}{8}\overline{p}\left(\frac{1}{64}p + \frac{1}{8}p\ x_{000} + \frac{3}{4}\overline{p}\ x_{000}\right) + \frac{3}{4}p\left(\frac{1}{8} + \frac{1}{64}p + \frac{1}{8}p\ x_{000} + \frac{3}{4}\overline{p}\ x_{000}\right)$$

Pour faciliter la lecture, x_{000} est remplacée par x dans la suite des calculs.

$$\begin{aligned} & x = \frac{1}{8}\overline{p}\left(\frac{1}{64}p + \frac{1}{8}p\ x + \frac{3}{4}\overline{p}\ x\right) + \frac{3}{4}p\left(\frac{1}{8} + \frac{1}{64}p + \frac{1}{8}p\ x + \frac{3}{4}\overline{p}\ x\right) \\ \iff & x = \frac{1}{512}p\overline{p} + \frac{1}{64}p\overline{p}x + \frac{3}{32}\overline{p}^2x + \frac{3}{32}p + \frac{3}{256}p^2 + \frac{3}{32}p^2x + \frac{9}{16}p\overline{p}x \\ \iff & x\left(1 - \frac{1}{64}p\overline{p} - \frac{3}{32}\overline{p}^2 - \frac{3}{32}p^2 - \frac{9}{16}p\overline{p}\right) = \frac{1}{512}p\overline{p} + \frac{3}{32}p + \frac{3}{256}p^2 \end{aligned}$$

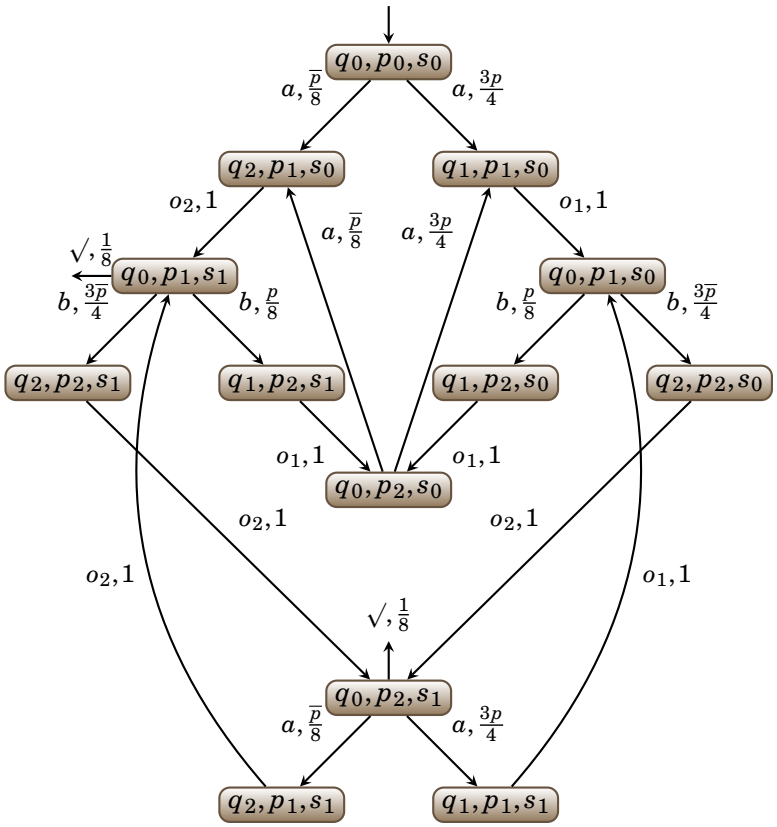


FIGURE 4.25: Automate sous-stochastique $\mathscr{A}_{14/\sigma_p}||\mathscr{A}_{\varphi_{\text{alt}}}||\mathscr{A}_{o_2}$.

$$
\begin{aligned}
\iff \quad x &= \frac{\frac{1}{8}p\overline{p}+6p+\frac{3}{4}p^2}{64-p\overline{p}-6\overline{p}^2-6p^2-36p\overline{p}} \\
\iff \quad x &= \frac{\frac{1}{8}p(1-p)+6p+\frac{3}{4}p^2}{64-37p(1-p)-6(p^2-2p+1)-6p^2} \\
\iff \quad x &= \frac{\frac{1}{8}p-\frac{1}{8}p^2+6p+\frac{3}{4}p^2}{64-37p+37p^2-6p^2+12p-6-6p^2} \\
\iff \quad x &= \frac{1}{8}\cdot p\cdot\frac{5p+49}{25p^2-25p+58}
\end{aligned}
$$

La même technique est employée pour le calcul de $\mathbf{P}(\varphi_{\text{alt}},o_2)$ sur $\mathscr{A}_{14/\sigma_p}$. Le produit synchronisé est représenté figure 4.25 et le système linéaire obtenu est (après élimination des équations redondantes) :

$$
\begin{cases}
x_{000} &= \frac{1}{8}\overline{p}\left(\frac{1}{8}+x_{010}\right)+\frac{3}{4}p\ x_{010} \\
x_{010} &= \frac{1}{8}p\ x_{000}+\frac{3}{4}\overline{p}\left(\frac{1}{8}+x_{000}\right)
\end{cases}
$$

Comme au dessus, x_{000} est remplacée par x pour faciliter la lecture ; on résout :

$$
\begin{aligned}
x &= \frac{1}{8}\overline{p}\left(\frac{1}{8}+\frac{1}{8}px+\frac{3}{4}\overline{p}\left(\frac{1}{8}+x\right)\right)+\frac{3}{4}p\ \left(\frac{1}{8}px+\frac{3}{4}\overline{p}\left(\frac{1}{8}+x\right)\right)\\
x &= \frac{1}{64}\overline{p}+\frac{1}{64}p\overline{p}x+\frac{3}{256}\overline{p}^2+\frac{3}{32}\overline{p}^2x+\frac{3}{32}p^2x+\frac{9}{128}p\overline{p}+\frac{9}{16}p\overline{p}x\\
x &= \frac{\frac{1}{64}\overline{p}+\frac{3}{256}\overline{p}^2+\frac{9}{128}p\overline{p}}{1-\frac{1}{64}p\overline{p}-\frac{3}{32}\overline{p}^2-\frac{3}{32}p^2-\frac{9}{16}p\overline{p}}\\
x &= \frac{4\overline{p}+3\overline{p}^2+18p\overline{p}}{256-24\overline{p}^2-24p^2-148p\overline{p}}\\
x &= \frac{(1-p)(4+3-3p+18p)}{256-24p^2+48p-24-24p^2-148p+148p^2}\\
x &= \frac{(1-p)(15p+7)}{232-100p+100p^2}\\
x &= \frac{(1-p)(15p+7)}{4(25p^2-25p+58)}
\end{aligned}
$$

On a donc calculé :

$$\mathbf{P}(\varphi_{\text{alt}},o_1)=\frac{p}{25p^2-25p+58}\cdot\frac{5p+49}{8}\qquad \mathbf{P}(\varphi_{\text{alt}},o_2)=\frac{1-p}{25p^2-25p+58}\cdot\frac{15p+7}{4}.$$

On calcule donc les probabilités conditionnelles, en notant $f(p)=25p^2-25p+58$. On a $\mathbf{P}(\overline{\varphi_{\text{alt}}}|\varepsilon)=1$ et

$$
\begin{aligned}
\mathbf{P}\left(\overline{\varphi_{\text{alt}}}|o_1\right) &= 1-\frac{\mathbf{P}\left(\varphi_{\text{alt}},o_1\right)}{\mathbf{P}(o_1)}\\
\mathbf{P}\left(\overline{\varphi_{\text{alt}}}|o_1\right) &= 1-\frac{1}{8}\cdot p\cdot\frac{5p+49}{25p^2-25p+58}\cdot\frac{8}{7p}\\
\mathbf{P}\left(\overline{\varphi_{\text{alt}}}|o_1\right) &= \frac{7f(p)-5p-49}{7f(p)}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{P}\left(\overline{\varphi_{\text{alt}}}|o_2\right) &= 1-\frac{\mathbf{P}(\varphi_{\text{alt}},o_2)}{\mathbf{P}(o_2)}\\
\mathbf{P}\left(\overline{\varphi_{\text{alt}}}|o_2\right) &= 1-\frac{(1-p)(15p+7)}{4(25p^2-25p+58)}\cdot\frac{8}{7(1-p)}\\
\mathbf{P}\left(\overline{\varphi_{\text{alt}}}|o_2\right) &= \frac{7f(p)-30p-14}{7f(p)}
\end{aligned}
$$

$$
\begin{aligned}
\frac{1}{\mathsf{PO}_r^A\left(\mathscr{A}_{14/\sigma_p},\varphi_{\text{alt}},\mathscr{O}_{\text{last}}\right)} &= \mathbf{P}(\varepsilon)+\mathbf{P}(o_1)\cdot\frac{1}{\mathbf{P}\left(\overline{\varphi_{\text{alt}}}|o_1\right)}+\mathbf{P}(o_2)\cdot\frac{1}{\mathbf{P}\left(\overline{\varphi_{\text{alt}}}|o_2\right)}\\
\frac{1}{\mathsf{PO}_r^A\left(\mathscr{A}_{14/\sigma_p},\varphi_{\text{alt}},\mathscr{O}_{\text{last}}\right)} &= \frac{1}{8}+\frac{7}{8}\cdot p\cdot\frac{7f(p)}{7f(p)-5p-49}+\frac{7}{8}\cdot(1-p)\cdot\frac{7f(p)}{7f(p)-30p-14}\\
\frac{1}{\mathsf{PO}_r^A\left(\mathscr{A}_{14/\sigma_p},\varphi_{\text{alt}},\mathscr{O}_{\text{last}}\right)} &= \frac{1}{8}+\frac{49f(p)}{8}\left(\frac{p}{7f(p)-5p-49}+\frac{1-p}{7f(p)-30p-14}\right)
\end{aligned}
$$

On peut montrer que la fonction

$$g:\left|\begin{array}{ccc}[0,1] & \to & [0,1]\\ p & \mapsto & \mathsf{PO}_r^A\left(\mathscr{A}_{14/\sigma_p},\varphi_{\text{alt}},\mathscr{O}_{\text{last}}\right)\end{array}\right.$$

a pour valeur minimale 0.88.

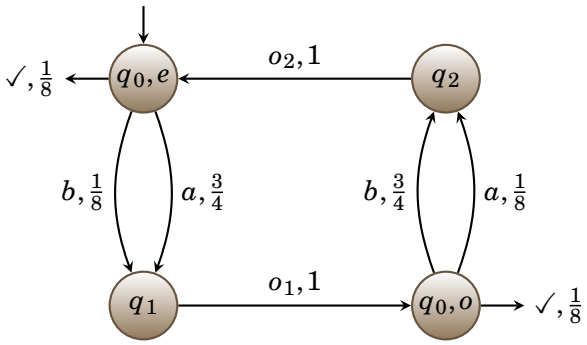


FIGURE 4.26: Automate probabiliste $\mathscr{A}_{14/\sigma_m}$, ordonnancement de $\mathscr{A}_{14}$ par σ_m.

On considère à présent l'ordonnanceur σ_m disposant d'un bit de mémoire [13]. Cet ordonnanceur essaye de maximiser la véracité de φ_{alt} en sélectionnant alternativement q_1 puis q_2 comme état cible. Ainsi, lorsqu'un nombre pair de lettres a ou b a été lu, le poids de la distribution vers q_1 est de 1 et celui de q_2 de 0, tandis que lorsqu'un nombre impair de lettres a été lu, le poids est 0 pour q_1 et 1 pour q_2. L'automate ordonnancé $\mathscr{A}_{14/\sigma_m}$ est représenté figure 4.26. De la même manière, le calcul des probabilités donne

$$\mathbf{P}(\varepsilon) = \frac{1}{8} \qquad \mathbf{P}(o_1) = \frac{7}{15} \qquad \mathbf{P}(o_2) = \frac{7}{8}\cdot\frac{7}{15}$$

$$\mathbf{P}(\varphi_{\text{alt}},\varepsilon) = 0 \qquad \mathbf{P}(\varphi_{\text{alt}},o_1) = \frac{3}{14} \qquad \mathbf{P}(\varphi_{\text{alt}},o_2) = \frac{3}{4}\cdot\frac{3}{14}.$$

La probabilité $\mathbf{P}(o_1)$ a été obtenue en observant que l'exécution doit s'arrêter après qu'un nombre impair de a ou de b a été lu. Or la probabilité d'arrêt après exactement n lettres a ou b est $\frac{1}{8}\cdot\left(\frac{7}{8}\right)^n$. D'où

$$\mathbf{P}(o_1) \;=\; \frac{1}{8}\cdot\sum_{i\geq 0}\left(\frac{7}{8}\right)^{2i+1} \;=\; \frac{1}{8}\cdot\frac{7}{8}\cdot\frac{1}{1-\frac{49}{64}} \;=\; \frac{1}{8}\cdot\frac{7}{8}\cdot\frac{64}{15} \;=\; \frac{7}{15}.$$

Des raisonnements similaires donnent les autres probabilités. Le calcul des probabilités conditionnelles et de la RPO s'obtient donc :

$$\mathbf{P}(\overline{\varphi_{\text{alt}}}|\varepsilon) = 1 \qquad \mathbf{P}(\overline{\varphi_{\text{alt}}}|o_1) = 1-\frac{3}{14}\cdot\frac{15}{7} = \frac{53}{98} \qquad \mathbf{P}(\overline{\varphi_{\text{alt}}}|o_2) = 1-\frac{3}{4}\cdot\frac{3}{14}\cdot\frac{15}{7}\cdot\frac{8}{7} = \frac{208}{343}$$

Ainsi :

$$\begin{aligned}
\frac{1}{\mathsf{PO}_r^A\left(\mathscr{A}_{14/\sigma_m},\varphi_{\text{alt}},\mathscr{O}_{\text{last}}\right)} &= \frac{1}{8}+\frac{7}{15}\cdot\frac{98}{53}+\frac{7}{8}\cdot\frac{7}{15}\cdot\frac{343}{208}\\
\frac{1}{\mathsf{PO}_r^A\left(\mathscr{A}_{14/\sigma_m},\varphi_{\text{alt}},\mathscr{O}_{\text{last}}\right)} &= \frac{146509}{88192}\\
\mathsf{PO}_r^A\left(\mathscr{A}_{14/\sigma_m},\varphi_{\text{alt}},\mathscr{O}_{\text{last}}\right) &= \frac{88192}{146509}\\
\mathsf{PO}_r^A\left(\mathscr{A}_{14/\sigma_m},\varphi_{\text{alt}},\mathscr{O}_{\text{last}}\right) &\simeq 0.60
\end{aligned}$$

On a donc un ordonnanceur avec mémoire qui rend le système moins robuste que tous les ordonnanceurs sans mémoire, ce qui conclut la preuve. □

13. Contrairement à p, m n'est ici pas un paramètre, mais dénote le fait que l'ordonnanceur n'est pas sans mémoire.

4.6.3 Ordonnanceurs restreints

Dans la preuve du théorème 4.9, l'ordonnanceur avec mémoire σ_m dispose par rapport à σ_p d'une information importante : il sait dans quel état se trouve l'automate $\mathscr{A}_{\varphi_{\text{alt}}}$ qui reconnaît les exécutions de φ_{alt} et ce qui sera observé grâce aux automates de chaque classe d'observation.

Ainsi, on définit la classe des ordonnanceurs bénéficiant exactement de cette information, que l'on conjecture suffisante pour atteindre les valeurs extrêmales des mesures d'opacité.

On considère un automate probabiliste non déterministe $\mathscr{A} = \langle S, Lab, \Delta, s_I \rangle$. Soit $\mathscr{O} : TExec(\mathscr{A}) \to \{o_1, \ldots, o_n\}$ une fonction d'observation régulière où pour $1 \leq i \leq n$, l'ensemble $\mathscr{O}^{-1}(o_i)$ est accepté par l'automate fini déterministe et complet $\mathscr{A}_{o_i}$. Soit $\varphi \subseteq TExec(\mathscr{A})$ un prédicat régulier accepté par l'automate déterministe et complet $\mathscr{A}_\varphi$. Soit $\mathscr{A}_{\varphi,\mathscr{O}}$ le produit synchronisé de tous ces automates :

$$\mathscr{A}_{\varphi,\mathscr{O}} \;=\; \mathscr{A}_\varphi \,||\, \mathscr{A}_{o_1} \,||\, \cdots \,||\, \mathscr{A}_{o_n}$$

qui est aussi déterministe et complet. On note $S_{\varphi,\mathscr{O}}$ son ensemble d'états et pour $\rho \in Exec(\mathscr{A})$, $\mathscr{A}_{\varphi,\mathscr{O}}(\rho)$ est l'état atteint après l'exécution de ρ.

Définition 4.13 (Ordonnanceur $(\varphi,\mathscr{O})$-restreint)**.** *Un ordonnanceur pour $\mathscr{A}$ est dit* $(\varphi,\mathscr{O})$-restreint *s'il existe une fonction*

$$\sigma' : \left(S \times S_{\varphi,\mathscr{O}}\right) \to \mathbb{D}(\mathbb{D}((Lab \times S) \uplus \{\checkmark\}))$$

telle que pour toute exécution $\rho \in Exec(\mathscr{A})$, $\sigma(\rho) = \sigma'\left(lst(\rho), \mathscr{A}_{\varphi,\mathscr{O}}(\rho)\right)$.

L'ensemble des ordonnanceurs $(\varphi,\mathscr{O})$-restreints est noté $Ord_{\varphi,\mathscr{O}}$. Les ordonnanceurs sans mémoire sont un cas particulier d'ordonnanceurs $(\varphi,\mathscr{O})$-restreints. Ces derniers ont par contre eux aussi la propriété de générer un automate probabiliste fini lorsqu'ils ordonnancent le système.

Proposition 4.10. *Si $\sigma \in Ord_{\varphi,\mathscr{O}}$ est un ordonnanceur $(\varphi,\mathscr{O})$-restreint sur $\mathscr{A}$, alors $\mathscr{A}_{/\sigma}$ est isomorphe à un automate probabiliste fini.*

Démonstration. On peut montrer que si σ est $(\varphi,\mathscr{O})$-restreint, alors :
- σ est sans mémoire sur le produit $\mathscr{A} \,||\, \mathscr{A}_{\varphi,\mathscr{O}}$
- et $\left(\mathscr{A} \,||\, \mathscr{A}_{\varphi,\mathscr{O}}\right)_{/\sigma} = \mathscr{A}_{/\sigma}$. □

Puisque les ordonnanceurs $(\varphi,\mathscr{O})$-restreints détiennent toute l'information à la fois sur le prédicat et sur l'observation, on conjecture qu'ils sont suffisamment puissants pour atteindre le pire cas du point de vue de la sécurité :

Conjecture 4.11. *Soit $\mathscr{A} = \langle S, Lab, \Delta, s_I \rangle$ un automate probabiliste non déterministe. Soit $\mathscr{O} : TExec(\mathscr{A}) \to \{o_1, \ldots, o_n\}$ une fonction d'observation régulière et $\varphi \subseteq TExec(\mathscr{A})$ un prédicat régulier. Pour toute mesure d'opacité*

$$PO^* \in \left\{PO^A_\ell, PO^S_\ell, PO^A_r, PO^S_r, PO^S_{en}, PO^S_{min}, PO^S_{wc}\right\},$$

il existe un ordonnanceur $(\varphi,\mathscr{O})$-restreint $\sigma \in Ord_{\varphi,\mathscr{O}}$ tel que

$$PO^*(\mathscr{A}_{/\sigma}, \varphi, \mathscr{O}) \;=\; \widehat{PO^*}(\mathscr{A}, \varphi, \mathscr{O}).$$

Si cette conjecture est vraie, le problème de mesure d'opacité dans les systèmes non déterministe (pour des cas simples puisque l'on suppose la régulalité du prédicat et de la fonction d'observation) reviendrait à un problème d'optimisation. Cependant, ce problème d'optimisation (*i*) comporterait de nombreuses variables (de l'ordre de $O\left(\left|S \times S_{\varphi,\mathscr{O}}\right| \times |\Delta|\right)$) et (*ii*) ne serait pas nécessairement linéaire.

Ainsi de nombreuses perspectives sont ouvertes sur les méthodes de calcul de l'opacité probabiliste, que ce soit dans le cas non déterministes ou dans le cas purement probabiliste. Et bien que ces calculs ne soient pas toujours possibles automatiquement, certaines classes de prédicats et de fonctions d'observation simplifient le problème.

HIÉRARCHIE DE L'INFORMATION DE TEMPS DANS LES AUTOMATES TEMPORISÉS À INTERRUPTIONS

CHAPITRE 5

Mais ses visions ne durent qu'un temps,
Et le temps lui-même disparaît.
Les heures se courbent dans l'espace.

Hubert-Félix THIÉFAINE, *Les ombres du soir*,
Suppléments de mensonge, 2011.

Outre les probabilités régissant le fonctionnement d'un système, une autre dimension quantitative des systèmes d'information est l'écoulement du temps. En effet, celui-ci peut être utilisé par un attaquant pour déduire de l'information cachée.

Dans ce chapitre, nous présentons le modèle des *automates temporisés à interruptions*, dans lesquels les informations sur le temps sont syntaxiquement séparées en *niveaux*.

Nous nous intéressons aux propriétés de ce modèle vis-à-vis des problèmes de *model-checking* de logiques temporelles (temporisées) et de l'expressivité des langages temporisés définis.

Plus précisément, nous montrons que le langage non temporisé d'un ITA est régulier, *via* une abstraction finie du temps. Cette construction fournit aussi une procédure (quoique peu efficace) d'accessibilité d'un état et de *model-checking* de propriétés non temporisées.

Nous définissons le modèle des ITA$_-$, syntaxiquement restreint mais équivalent aux ITA vis-à-vis des langages temporisés acceptés. Dans un ITA$_-$, il est possible de borner la taille d'un chemin minimal *via* des arguments de comptage. Cette propriété est utilisée dans la suite dans des procédures de *model-checking* de propriétés temporelles ou pour montrer l'incomparabilité entre la classe des langages définies par les ITA et la classe des langages définis par les automates temporisés.

Concernant le *model-checking*, nous montrons l'indécidabilité du problème pour la logique *State Clock Logic* (SCL), un fragment de MITL, une extension temporisée de LTL. Pour les logiques arborescentes, nous donnons deux fragments pour lesquels le problème de *model-checking* est décidable.

Outre l'expressivité incomparable avec le modèle des automates temporisés, nous montrons que la classe des langages définis par les ITA n'est pas close par intersection ni par complémentation.

Nous montrons enfin que si la hiérarchie entre les horloges est brisée par l'intermédiaire d'un observateur extérieur (sous la forme d'un automate temporisé), ce modèle ne dispose plus de la décidabilité du problème d'accessibilité.

5.1 Le modèle des automates temporisés à interruptions

Dans le modèle des automates temporisés à interruptions (ITA) [BH09], les états du système sont séparés en n niveaux, et une horloge unique est attribuée à chaque niveau. Lorsque le système se trouve dans un niveau, seule l'horloge associée évolue. Celles des niveaux inférieurs gardent leur valeur et peuvent ensuite être redémarrées, à la manière des chronomètres [CL00]. Un exemple d'évolution des valeurs d'horloges est représenté figure 5.1.

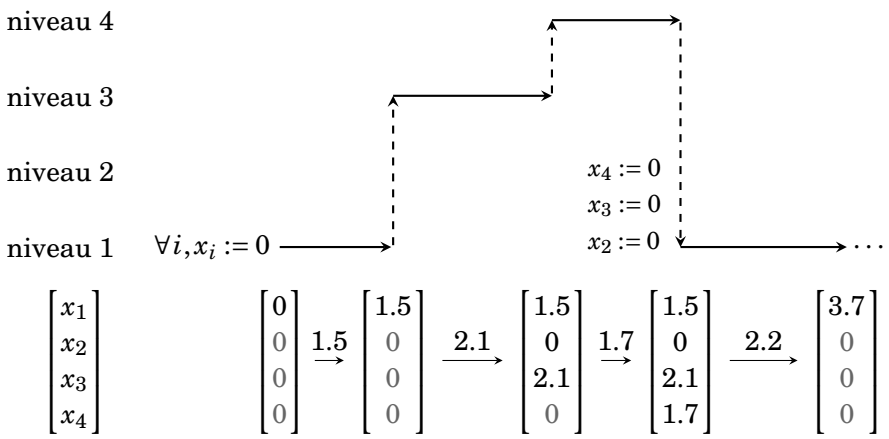


FIGURE 5.1: Niveaux et horloges d'interruption dans un ITA.

Les horloges de niveau supérieur ne peuvent pas être utilisées dans des gardes ou des mises à jour, tandis que les valeurs des horloges de niveau inférieur peuvent servir dans les gardes et les mises à jour. De plus, les horloges de niveau inférieur peuvent elle-même être mises à jour (en n'utilisant là encore que des valeurs d'horloges de niveau plus bas). Puisque les horloges de niveau supérieur ne sont pas utilisables, leur valeur est par défaut à 0.

Cette séparation en niveaux peut modéliser un système dans lequel à chaque niveau s'exécute une tâche qui peut être interrompue par une tâche de niveau supérieur. Les niveaux sont donc à la fois des niveaux de priorité, puisqu'une tâche interrompt une tâche inférieure, et de sécurité, puisqu'une tâche ne lit que les valeurs d'horloges de niveau inférieur.

On ajoute à chaque état une *politique* qui peut être soit *urgente*, soit *paresseuse*, soit *retardée*. Les états urgents ne peuvent pas laisser du temps s'écouler, tandis que les états retardés doivent laisser passer du temps. Les états paresseux n'ont pas de telle contrainte. Ces politiques peuvent, dans le cas des automates temporisés, être implémentées par des horloges additionnelles qui ne sont pas autorisées par le modèle des ITA. Par extension, une transition qui prend sa source dans un état urgent (respectivement paresseux ou retardé) sera dite urgente (respectivement paresseuse ou retardée). De même, le niveau d'une transition est le niveau de l'état source.

De plus, en vue de vérifier des formules de logique temporelle sur les automates temporisés à interruptions, chaque état dispose d'un ensemble de propositions atomiques qui sont vraies dans cet état.

Pour un ensemble X d'horloges, une mise à jour de $x \in X$ est une affectation de la

forme $x := C$, où C est une expression linéaire sur X :

$$C = \sum_{y \in X} a_y y + b$$

où les $\{a_y\}$ et b sont des constantes de $\mathbb{Q}$. Une mise à jour sur X est une conjonction d'affectations pour les horloges de X :

$$\bigwedge_{x \in X} x := C_x.$$

L'ensemble des mises à jour sur X est noté $\mathscr{U}(X)$. Pour une expression linéaire C, l'application d'une mise à jour u de la forme $\bigwedge_{x \in X} x := C_x$ est l'expression $C[u]$ où chaque variable x est remplacée par l'expression C_x. Remarquons que ces mises à jour sont atomiques. Par exemple, considérons l'expression $C = x_2 - 2x_1 + 3$ sur $\{x_1, x_2\}$ et la mise à jour $u = x_1 := 1 \wedge x_2 := 2x_1 + 1$ (les affectations triviales de la forme $x := x$ sont omises). On a donc $C[u] = 2x_1 + 2$.

Une condition sur X est une conjonction de contraintes atomiques de la forme $C \bowtie 0$ où C est une expression linéaire sur X et $\bowtie \in \{<, \leq, =, \geq, >\}$ est un opérateur de comparaison. L'ensemble des conditions est noté $\mathscr{C}(X)$.

Par extension, une mise à jour peut s'appliquer sur une condition (en s'appliquant sur l'expression linéaire de chaque contrainte atomique) et même sur une autre mise à jour. Par exemple pour la garde $g = x_3 - x_2 + 2x_1 - 3 > 0$ et la mise à jour $u = x_1 := 1 \wedge x_2 := 2x_1 + 1$, on obtient la garde $g[u] = x_3 - 2x_1 - 2 > 0$.

Définition 5.1 (Automate temporisé à interruptions)**.** *Un automate temporisé à interruptions (ITA) est un décuplet $\mathscr{A} = \langle S, Lab, X, \lambda, pol, \Delta, s_I, F, AP, prop \rangle$ où*

- *S est un ensemble fini d'états ;*
- *Lab est un ensemble fini d'étiquettes ;*
- *$X = \{x_1, \ldots, x_n\}$ est un ensemble fini d'horloges d'interruption ;*
- *la fonction $\lambda : S \to \{1, \ldots, n\}$ associe à chaque état son niveau ;*
- *la fonction $pol : S \to \{U, L, D\}$ associe à chaque état sa politique : urgente (U), paresseuse (L) ou retardée (D) ;*
- *la relation de transition $\Delta \subseteq S \times \mathscr{C}(X) \times (Lab \uplus \{\varepsilon\}) \times \mathscr{U}(X) \times S$ est telle que si $s \xrightarrow{g,a,u} s' \in \Delta$ avec $\lambda(s) = k$ et $\lambda(s') = k'$*
  - *g ne fait apparaître que les horloges $\{x_1, \ldots x_k\}$:*

$$g = \bigwedge \left(\sum_{i=1}^{k} a_i x_i + b \bowtie 0 \right)$$

  - *u est une conjonction d'affectations $x_i := C_i$ où*
    - *pour $i \in \{1, \ldots, \min(k, k')\}$, soit $C_i = \sum_{j=1}^{i-1} a_j x_j + b$ (mise à jour en utilisant les valeurs d'horloges de niveau strictement plus bas), soit $C_i = x_i$ (pas de mise à jour)*
    - *pour $i \in \{\min(k, k') + 1, \ldots, \max(k, k')\}$, $C_i = 0$: les horloges des niveaux traversés (en montant [1] ou en descendant) sont remises à zéro ;*
- *$s_I \in S$ est l'état initial ;*
- *$F \subseteq S$ est l'ensemble des états finaux.*
- *AP est un ensemble de propositions atomiques ;*
- *la fonction $prop : S \to 2^{AP}$ associe à chaque état l'ensemble des propositions atomiques qui y sont vraies.*

1. Ces horloges étant déjà à 0 ces mises à jour sont purement syntaxiques.

Même si la définition formelle des ITA n'autorise qu'un seul état initial pour faciliter les notations, on peut simuler la présence de plusieurs états initiaux en prenant un unique état initial urgent relié par des transitions sans garde (c'est-à-dire de garde $\top$), étiquetée par ε et sans mise à jour (c'est à dire avec la mise à jour triviale $\bigwedge_i x_i := x_i$) vers les divers états originellement initiaux.

Une configuration d'un ITA $\mathscr{A}$ est un triplet (s, v, β) où s est un état de $\mathscr{A}$, v une valuation qui à chaque horloge associe une valeur dans $\mathbb{R}$, et β un booléen dans $\mathbb{B} = \{\bot, \top\}$ mémorisant si du temps s'est écoulé depuis le dernier franchissement de transition. Ainsi on interdit les pas de temps de durée non nulle si la politique de l'état courant est urgente et l'on interdit le franchissement de transitions depuis des états de politique retardée si l'on n'a pas encore laissé passer de temps ($\beta = \bot$).

Les valuations sont étendues aux expressions linéaires et sont modifiées par mise à jour de la même manière que pour les automates temporisés. Ainsi pour l'expression $C = \sum_{x \in X} a_x x + b$, $v(C) = \sum_{x \in X} a_x v(x) + b$ et pour une mise à jour $u = \bigwedge_{x \in X} x := C_x$, la valuation $v[u]$ est donnée par $v[u](x) = v(C_x)$ pour toute horloge $x \in X$.

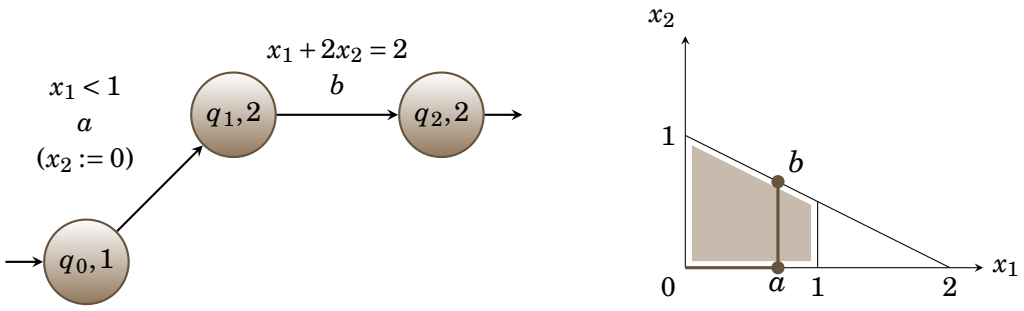
Lorsque $v(C) \bowtie 0$, on écrit $v \vDash C \bowtie 0$; cette notation s'étend aux conditions (en tant que conjonctions de telles contraintes).

Définition 5.2 (Sémantique des ITA)**.** *La sémantique d'un automate temporisé à interruptions est le système de transition temporisé* $\mathscr{T}_{\mathscr{A}} = \langle S_{\mathscr{T}}, Lab \uplus \{\varepsilon\} \uplus \mathbb{R}_{\geq 0}, \Delta_{\mathscr{T}}, I_{\mathscr{T}} \rangle$ *défini par :*

- $S_{\mathscr{T}} = S \times \mathbb{R}^X \times \mathbb{B}$ *est l'ensemble des* configurations *de* $\mathscr{A}$ *;*
- $I_{\mathscr{T}} = \{(s_I, \mathbf{0}, \bot)\}$ *contient la configuration initiale de* $\mathscr{A}$ *;*
- *pour une configuration* (s, v, β) *telle que* $pol(s) \neq U$ *et pour une durée* $d > 0$*,* $(s, v, \beta) \xrightarrow{d} (s, v', \top) \in \Delta_{\mathscr{T}}$ *avec* $v'\left(x_{\lambda(s)}\right) = v\left(x_{\lambda(s)}\right) + d$ *et* $v'(x) = v(x)$ *pour les autres horloges (un pas de temps de durée non nulle) ;*
- $(s, v, \beta) \xrightarrow{0} (s, v, \beta) \in \Delta_{\mathscr{T}}$ *(un pas de temps nul ne change pas la configuration) ;*
- *pour une configuration* (s, v, β) *et une transition* $e = \left(s \xrightarrow{g,a,u} s'\right) \in \Delta$ *telle que* $v \vDash g$*, si* $pol(s) \neq D$ *ou* $\beta \neq \bot$*, alors* $(s, v, \beta) \xrightarrow{a} (s, v[u], \bot) \in \Delta_{\mathscr{T}}$ *(un pas discret). Afin de conserver dans la sémantique une trace de la transition de l'ITA, le pas discret peut aussi bien être étiqueté par la transition* e *au lieu de la lettre* a*.*

Les exécutions, les mots temporisés, le langage des ITA, le notions de durée et de longueur d'exécutions finies sont définis de la même manière que pour les automates temporisés. L'ensemble des langages temporisés définis par des ITA est noté ITL.

Un exemple d'ITA contenant deux niveaux (et donc deux horloges) est représenté sur la figure 5.2(a). Tous les états ont une politique paresseuse (qui est la politique par défaut, et n'est souvent pas indiquée dans les états). Une exécution acceptée par $\mathscr{A}_{15}$ est représentée de manière géométrique sur la figure 5.2(b). Tout d'abord seule la valeur de x_1 évolue dans l'état q_0 (ligne horizontale). Après le franchissement de a, la valeur de x_1 ne change plus dans l'état q_1, et seule x_2 augmente (ligne verticale), jusqu'à atteindre la droite d'équation $x_2 = -\frac{1}{2}x_1 + \frac{1}{2}$, qui correspond à la satisfaction de la garde pour b. La zone colorée, définie par $(0 < x_1 < 1, 0 < x_2 < -\frac{1}{2}x_1 + \frac{1}{2})$, correspond à des valuations que l'on peut rencontrer dans q_1. Le langage accepté par l'ITA $\mathscr{A}_{15}$ est $L_{15} = \mathscr{L}(\mathscr{A}_{15}) = \left\{(a, \tau)(b, 1 + \frac{\tau}{2}) \,\middle|\, 0 \leq \tau < 1\right\}$.



(a) Un ITA $\mathscr{A}_{15}$ à deux niveaux d'interruption.

(b) Une exécution possible de $\mathscr{A}_{15}$.

FIGURE 5.2: Un exemple d'ITA avec une exécution possible.

Sémantique pour le *model-checking*. Dans le contexte du *model-checking*, on ne considère plus les exécutions acceptées, mais les exécutions totales, notées $TExec(\mathscr{A})$. Ce sont des exécutions soit infinies, soit telles qu'aucun pas discret n'est plus possible. Ces exécutions totales n'excluent pas les exécutions dites *Zénon* où le temps ne diverge pas. De plus, la fonction *prop* est étendue aux configurations : $prop(s, v, \beta) = prop(s)$.

Le long d'une exécution, on utilise la notion de *positions* totalement ordonnées qui permettent de distinguer plusieurs actions simultanées [HNSY94]. Pour une exécution $\rho \in TExec(\mathscr{A})$, l'ordre strict entre les positions est noté $<_\rho$. Pour une position π le long de ρ, le préfixe jusqu'à π (inclus) est une exécution finie notée $\rho_{\leq\pi}$. De même, le suffixe de ρ depuis π est noté $\rho_{\geq\pi}$. On utilise aussi la notation $\rho_{<\pi}$ (respectivement $\rho_{>\pi}$) pour le préfixe (respectivement le suffixe) de ρ jusqu'à (respectivement depuis) π exclus. Pour deux positions $\pi \leq_\rho \pi'$, la partie de ρ entre π et π' est notée $\rho_{[\pi,\pi']}$, de durée $Dur\left(\rho_{[\pi,\pi']}\right) = Dur\left(\rho_{\leq\pi'}\right) - Dur\left(\rho_{\leq\pi}\right)$.

5.2 Régularité du langage non temporisé d'un ITA

Nous montrons dans cette section que le langage non temporisé d'un ITA $\mathscr{A}$ est régulier. Comme dans le cas des automates temporisés, cette preuve repose sur la construction d'un système de transition fini $\mathscr{G}_\mathscr{A}$ bisimilaire par abstraction du temps au système de transition infini $\mathscr{T}_\mathscr{A}$. Ce système, appelé *graphe des classes*, fournit entre autre une preuve que le problème d'accessibilité est décidable pour les ITA.

Théorème 5.1 (Régularité du langage non temporisé d'un ITA)**.** *Soit $\mathscr{A}$ un automate temporisé à interruptions. Le langage $Untimed(\mathscr{L}(\mathscr{A}))$ est régulier.*

Dans la suite on considère l'ITA $\mathscr{A} = \langle S, Lab, X, \lambda, pol, \Delta, s_I, F, AP, prop\rangle$ avec n niveaux. On note $T = |\Delta|$ la taille de l'ensemble des transitions de $\mathscr{A}$; cette taille tient compte autant du nombre des transitions que de la taille des gardes et des mises à jour y apparaissant.

5.2.1 Le graphe des classes

La construction du graphe des classes est plus complexe que celle de l'automate des régions [AD94]. Plus précisément, elle se base sur les expressions linéaires qui apparaissent dans les gardes et les mises à jour des transitions du système, et non simplement sur la constante maximale apparaissant dans celles-ci.

On associe à chaque état s un ensemble d'expressions $Exp(s)$ en fonction de son niveau. Pour cela on définit une famille d'ensembles d'expressions $\{E_k\}_{1\leq k\leq n}$ et l'on note $Exp(s) = \bigcup_{k=1}^{\lambda(s)} E_k$.

Lors de l'exécution de l'automate dans un état s, et donc dans un niveau $\lambda(s)$, les valeurs à laquelle $x_{\lambda(s)}$ doit se comparer dans les gardes sont des expressions sur les horloges $x_1,\ldots,x_{\lambda(s)-1}$. Or au niveau $\lambda(s)$, les valeurs de ces horloges, et donc des expressions, n'évoluent pas avec l'écoulement du temps. Il suffit donc de comparer $x_{\lambda(s)}$ avec ces expressions pour déterminer si une transition est franchissable ou non. D'autre part, connaitre l'ordre des expressions sur $x_1,\ldots,x_{\lambda(s)-1}$ entre elles permet de savoir dans quel ordre $x_{\lambda(s)}$ rencontre ces expressions lors de l'écoulement du temps. Puisque deux expressions peuvent avoir la même valeur, on les range au sein de chaque E_k selon un *préordre total*, c'est-à-dire une relation réflexive ($x \preceq x$) transitive (si $x \preceq y$ et $y \preceq z$ alors $x \preceq z$) totale (si $x \not\preceq y$ alors $y \preceq x$). Les valuations qui ordonnent les expressions de $Exp(s)$ de la même manière constituent donc une *classe*.

Nous décrivons dans la suite tout d'abord comment sont calculées ces expressions. Puis nous donnons la construction du graphe des classes $\mathscr{G}_{\mathscr{A}}$ et nous montrons que ce système est bisimilaire à $\mathscr{T}_{\mathscr{A}}$ par abstraction du temps. Plus précisément, si (s,v,β) est dans la classe [2] R et $(s,v,\beta) \xrightarrow{a} (s',v',\beta')$ (avec $a \in Lab \uplus \{\varepsilon\}$) alors $R \xrightarrow{a} R'$ où (s',v',β') est dans la classe R' (et dualement si $R \xrightarrow{a} R'$). De plus, si $(s,v,\beta) \xrightarrow{d} (s',v',\beta')$ (avec $d \in \mathbb{R}_{\geq 0}$) alors $R \xrightarrow{succ}{}^* R'$ où $\xrightarrow{succ}{}^*$ signifie zéro ou plusieurs transitions de successeur par le temps $\xrightarrow{succ}$ et (s',v',β') est dans la classe R' (et dualement si $R \xrightarrow{succ} R'$).

Calcul des expressions

On introduit tout d'abord un opérateur de *normalisation* d'une expression linéaire par rapport à un niveau. La normalisation sert à comparer les valeurs des différentes expressions pour un certain niveau, comme expliqué dans la suite.

Définition 5.3 (Normalisation)**.** *Pour une expression linéaire $C = \sum_{i=1}^{k} a_i x_i + b$ sur $\{x_1,\ldots,x_k\}$, la k-normalisation de C est définie par*

$$Norm(C,k) = \begin{cases} C & si\ a_k = 0 \\ x_k + \frac{1}{a_k}\left(\sum_{i=1}^{k-1} a_i x_i + b\right) & sinon \end{cases}$$

Puisque toutes les gardes ne contiennent que des conditions $C \bowtie 0$ où C est une expression linéaire à constantes rationnelles [3], on peut supposer que C est sous la forme $x_k + \sum_{i=1}^{k-1} a_i x_i + b$ (par normalisation et éventuellement en changeant l'opérateur de comparaison) ou sous la forme $\sum_{i=1}^{k-1} a_i x_i + b$. On l'écrit donc sous la forme $\alpha x_k + \sum_{i=1}^{k-1} a_i x_i + b$, avec $\alpha \in \{0,1\}$.

2. Puisqu'il s'agit ici d'une partition, il est plus aisé de dire qu'« une configuration est dans une classe » plutôt qu'« une configuration est bisimilaire à une classe ».

3. On ne peut donc pas ici supposer, contrairement au cas des automates temporisés, que les constantes apparaissant dans un ITA sont entières.

On initialise chaque E_k à $\{x_k, 0\}$. Le calcul des ensembles d'expressions $\{E_k\}_{1\leq k\leq n}$ procède de haut en bas – de E_n jusqu'à E_1. En effet, lors du traitement d'un niveau k, la procédure ajoute des expressions aux ensembles correspondant à des niveaux inférieurs. Intuitivement, lors du calcul de E_k, on ajoute aux niveaux inférieurs les expressions nécessaires au calcul de l'ordre entre les expressions de E_k. On détaille à présent le traitement des expressions de niveau k.

(1) Tout d'abord, pour chaque expression $\alpha x_k + \sum_{i=1}^{k-1} a_i x_i + b$ (avec $\alpha \in \{0,1\}$) apparaissant dans la garde d'une transition dont l'état source est de niveau k, on ajoute $-\sum_{i=1}^{k-1} a_i x_i - b$ à E_k.

(2) Ensuite, on itère les opérations suivantes jusqu'à ce qu'aucune nouvelle expression ne soit ajoutée aux $\{E_i\}_{1\leq i\leq k}$.

(2.1) Soit $s \xrightarrow{g,a,u} s' \in \Delta$ avec $\lambda(s) \geq k$ et $\lambda(s') \geq k$. Pour $C \in E_k$, on ajoute $C[u]$ à E_k (rappelons que $C[u]$ est l'expression C à laquelle u a été appliquée).

(2.2) Soit $s \xrightarrow{g,a,u} s' \in \Delta$ avec $\lambda(s) < k$ et $\lambda(s') \geq k$. Soient C et C' deux expression distinctes [4] de E_k. Soit $C'' = Norm\big(C[u] - C'[u], \lambda(s)\big)$. Notons que les conditions sur les mises à jour dans les ITA assurent que pour $i \in \{\lambda(s)+1, \ldots, n\}$, x_i n'apparaît ni dans $C[u]$ ni dans $C'[u]$; la normalisation peut donc s'appliquer. L'expression C'' peut être réécrite $\alpha x_{\lambda(s)} + \sum_{i=1}^{\lambda(s)-1} a_i x_i + b$ (avec $\alpha \in \{0,1\}$). On ajoute alors l'expression $-\sum_{i=1}^{\lambda(s)-1} a_i x_i - b$ à $E_{\lambda(s)}$.

Remarquons que les expressions de E_i ne portent que sur $\{x_1, \ldots, x_i\}$. Plus précisément, hormis x_i, toutes les expressions de E_i ne portent que sur $\{x_1, \ldots, x_{i-1}\}$.

Exemple. On illustre la construction des expressions sur l'ITA $\mathscr{A}_{15}$ représenté figure 5.2(a) (page 97).

On démarre avec les ensembles $E_1 = \{0, x_1\}$ et $E_2 = \{0, x_2\}$. Pour le niveau 2, l'étape (1) ajoute l'expression $-\frac{1}{2}x_1 + 1$, issue de la normalisation de la contrainte $x_1 + 2x_2 = 2$, garde de la transition étiquetée par b. L'étape (2.1) ne s'applique pas ici car la seule transition de niveau 2 n'a pas de mise à jour (non triviale). L'étape (2.2) s'applique avec la transition étiquetée par a et la mise à jour $x_2 := 0$. Il faut donc ajouter à E_1 toutes les différences (normalisées) des expressions de E_2 après application de la mise à zéro de x_2. Cela donne donc seulement l'expression $-\frac{1}{2}x_1 + 1 - 0$, qui se normalise en $x_1 - 2$, et donc donne l'expression 2 à ajouter à E_1.

Au niveau 1, seule l'étape (1) ajoute des expressions à E_1, à savoir l'expression 1.

On obtient donc les ensembles d'expressions $E_1 = \{x_1, 0, 1, 2\}$ et $E_2 = \{x_2, 0, -\frac{1}{2}x_1 + 1\}$.

Complexité.

Lemme 5.2. *La procédure de construction des $\{E_k\}_{1\leq k\leq n}$ termine et la taille de chaque E_k est bornée par $(T+2)^{2^{n(n-k+1)}+1}$ où T est la taille des transitions de l'ITA.*

Démonstration. Pour un niveau k, on prove la terminaison de l'étape relative à k. L'étape (1) termine trivialement car T est fini. Remarquons que l'étape (2.2) n'ajoute de nouvelles expressions qu'aux ensembles d'expressions de niveau strictement inférieur à k. On peut donc considérer que l'étape (2) sature d'abord E_k par l'étape (2.1), puis sature les $\{E_i\}_{1\leq i<k}$ par l'étape (2.2).

4. On peut choisir un ordre entre C et C' afin d'éviter les doublons, mais cela ne change pas le reste de la construction.

On pose $B = T + 2$ pour faciliter la lecture de la suite de la preuve.

Montrons tout d'abord la terminaison de l'étape (2.1). Notons E_k^0 l'ensemble E_k au début de cette étape. À chaque ajout d'une expression $C[u]$ on peut associer son *père* C. Ainsi E_k est une forêt qui croît au cours de cette étape. Cette forêt est à branchement fini, car T est fini, donc un nombre fini de mises à jour apparaissent dans $\mathscr{A}$. De plus, E_k a au plus une racine par élément de E_k^0.

Supposant que l'étape (2.1) ne termine pas. On a donc une forêt infinie à branchement fini. Par le lemme de König, E_k contient alors une branche infinie $C_0, C_1, \ldots$ où pour $i \in \mathbb{N}$, $C_{i+1} = C_i[u_i]$ pour une certaine mise à jour u_i telle que $C_{i+1} \neq C_i$. Remarquons que lors d'une mise à jour, une variable x_j ne peut pas être remplacée par une expression contenant une variable $x_{j'}$ si $j' > j$. En particulier, si x_k est remplacée par une expression sur $\{x_1, \ldots, x_{k-1}\}$ par une mise à jour u_{i_0}, alors x_k ne peut plus apparaître dans C_i pour $i > i_0$. Ainsi la branche infinie contient au plus une seule mise à jour de x_k. Un raisonnement similaire montre que x_{k-1} n'est mis à jour au plus qu'une seule fois avant i_0 et une seule fois après[5] i_0 (ou une seule fois le long de la branche si x_k n'est pas mis à jour). En itérant ce raisonnement, on obtient qu'il ne peut se produire que $2^k - 1$ mises à jour le long de cette branche, et donc que la longueur de la branche est bornée par 2^k. Ceci est en contradiction avec l'hypothèse que la branche est infinie. Le branchement de chaque arbre de la forêt est borné par T donc par B. Ainsi la taille de E_k après l'étape (2.1) est au plus $\left|E_k^0\right| \times B^{2^k}$.

L'étape (2.2) ajoute au plus $B \times \frac{|E_k| \times (|E_k|-1)}{2}$ expressions dans les $\{E_i\}_{1 \leq i < k}$, ce qui conclut la preuve de terminaison.

On montre maintenant par induction (descendante) que si $n \geq 2$, $|E_k| \leq B^{2^{n(n-k+1)}+1}$. Notons $p_k = |E_k|$.

Cas de base : $k = n$. L'ensemble E_n ne croît que par les étapes (1) et (2.1). On a donc $p_n \leq p_n^0 \times B^{2^n}$ où p_n^0 est le nombre de conditions dans les gardes de niveau n. D'où

$$p_n \;\leq\; B \times B^{2^n} \;=\; B^{2^n+1} \;=\; B^{2^{n(n-n+1)}+1}$$

qui est la borne énoncée.

Induction. Supposons que pour $j \in \{k+1, \ldots, n\}$ on ait $p_j \leq B^{2^{n(n-j+1)}+1}$. Notons p_k^0 le nombre de transitions ajoutées à E_k par les étapes (2.2) aux niveaux supérieurs. On a :

$$\begin{aligned}
p_k^0 &\leq B + B \times \left(\frac{p_{k+1} \times (p_{k+1}-1)}{2} + \cdots + \frac{p_n \times (p_n - 1)}{2} \right) \\
&\leq B + B \times \left(p_{k+1}^2 + \cdots + p_n^2 \right) \\
&\leq B + B \times \left(B^{2^{n(n-k)+1}+2} + \cdots + B^{2^{n+1}+2} \right) \\
&\leq B \times (n-k+1) \times B^{2^{n(n-k)+1}+2} \\
&\leq B \times B^n \times B^{2^{n(n-k)+1}+2} \quad \text{(on utilise ici le fait que } B \geq 2\text{)} \\
p_k^0 &\leq B^{2^{n(n-k)+1}+n+3}
\end{aligned}$$

5. En effet, la mise à jour de x_k peut faire (ré)apparaître x_{k-1}.

En prenant en compte les étapes (1) et (2.1) au niveau k, on obtient :

$$p_k \leq B^{2^{n(n-k)+1}+2^k+n+3}.$$

Posons $\delta = 2^{n(n-k+1)} + 1 - \left(2^{n(n-k)+1} + 2^k + n + 3\right)$. Puisque $k < n$, on a :

$$\begin{aligned}
\delta &\geq (2^{n-1}-1) \times 2^{n(n-k)+1} - (2^k + n + 2) \\
&\geq (2^{n-1}-1) \times 2^{n(n-k)+1} - (2^{n-1} + 2^n) \\
&\geq (2^{n-1}-1) \times 2^{n(n-k)+1} - 2^{n+1} \\
\delta &\geq 0
\end{aligned}$$

Donc

$$p_k \leq B^{2^{n(n-k)+1}+2^k+n+3} \leq B^{2^{n(n-k+1)}+1} = (T+2)^{2^{n(n-k+1)}+1}$$

qui est la borne énoncée. □

Construction du graphe des classes

On construit tout d'abord le graphe des classes en supposant que la politique de tous les états est paresseuse. Le cas des états urgents et retardés est traité ensuite. Comme souvent, ce graphe a pour états des *classes* correspondant à des ensembles de configurations équivalentes du point de vue du passage du temps et des conditions sur les horloges. Les transitions sont de deux types : soit elle correspondent à des pas discrets du système de transition temporisé sous-jacent, soit elles constituent une abstraction des pas de temps.

Les classes. Les classes sont une représentation d'un sous-ensemble de configurations de l'ITA. Une classe est une paire $R = \left(s, \{\preceq_k\}_{1 \leq k \leq \lambda(s)}\right)$ où $s \in S$ est un état de $\mathscr{A}$ et pour $k \in \{1, \ldots, \lambda(s)\}$, $\preceq_k$ est un préordre total sur E_k. La classe R décrit l'ensemble de configurations

$$[\![R]\!] = \left\{(s, v, \beta) \,\middle|\, \beta \in \mathbb{B},\ \forall k \in \{1, \ldots, \lambda(s)\},\ \forall C, C' \in E_k,\ v(C) \leq v(C') \text{ ssi } C \preceq_k C'\right\}.$$

Le nombre de classes pour un état donné est borné par le nombre de préordres possibles sur les expressions des niveaux inférieurs. On pose $m = (T+2)^{2^{n^2}+1}$ qui est le nombre maximal d'expressions à un niveau quelconque. Le nombre de préordres sur m expressions ne peut excéder $m! \times 2^{m-1}$, puisqu'il existe $m!$ ordres stricts $C_{i_1} < \cdots < C_{i_m}$ et que chacune des $m-1$ inégalités « $C_{i_j} < C_{i_{j+1}}$ » peut en fait être une égalité « $C_{i_j} = C_{i_{j+1}}$ ».

Pour un état de niveau k, on doit choisir un préordre parmi les $m! \times 2^{m-1}$ pour chaque niveau inférieur. On a donc au plus $\left(m! \times 2^{m-1}\right)^n$ ensembles de préordres possibles pour chaque état. Or

$$\begin{aligned}
m! \times 2^{m-1} &\leq m^m \times 2^m \\
&\leq 2^{m(\log(m)+1)} \\
&\leq 2^{m^2} \\
&\leq 2^{\left((T+2)^{2^{n^2}+1}\right)^2} \\
m! \times 2^{m-1} &\leq 2^{(T+2)^{2^{n^2}+1}+2}
\end{aligned}$$

donc le nombre total de classes dans $\mathscr{G}_{\mathscr{A}}$ est borné par

$$|S| \times 2^{n \times (T+2)^{2^{n^2+1}+2}}.$$

La classe initiale de $\mathscr{G}_{\mathscr{A}}$ est celle qui contient la configuration initiale $(s_0, \mathbf{0}, \bot)$; elle se détermine en ordonnant les expressions de $Exp(s_0)$ pour la valuation nulle. Les classes finales sont toutes les classes $R = \left(s, \{\preceq_k\}_{1 \le k \le \lambda(s)}\right)$ où $s \in F$.

Les transitions discrètes. Les transitions discrètes entre deux classes R et R' doivent satisfaire :

- pour $(s,v,\beta) \in [\![R]\!]$ et $(s',v',\beta') \in [\![R']\!]$, si $(s,v,\beta) \xrightarrow{e} (s',v',\beta')$ dans $\mathscr{T}_{\mathscr{A}}$, alors on a une transition $R \xrightarrow{e} R'$ dans le graphe des classes ;
- et si il y a une transition $R \xrightarrow{e} R'$ dans le graphe des classes, pour toute configuration $(s,v,\beta) \in [\![R]\!]$ il y a une configuration $(s',v',\beta') \in [\![R']\!]$ telle que $(s,v,\beta) \xrightarrow{e} (s',v',\beta')$ dans $\mathscr{T}_{\mathscr{A}}$.

Pour $R = \left(s, \{\preceq_i\}_{1 \le i \le \lambda(s)}\right)$ et $e = \left(s \xrightarrow{g,a,u} s'\right) \in \Delta$ on détermine si e est franchissable depuis R et on calcule l'éventuelle classe successeur R' de la façon suivante.

Pour plus de lisibilité, on pose $k = \lambda(s)$.

Condition de franchissement. Une garde peut s'écrire $g = \bigwedge_{j \in J} C_j \bowtie_j 0$ où l'expression linéaire C_j est sous la forme $C_j = \alpha x_k + \sum_{i=1}^{k-1} a_i x_i + b$ avec $\alpha \in \{0,1\}$. Par construction, l'expression $C'_j = -\sum_{i=1}^{k-1} a_i x_i - b$ est dans l'ensemble E_k (car ajoutée à l'étape (1)). Pour chaque contrainte $C_j \bowtie_j 0$ on définit une (conjonction de) contrainte sur $\preceq_k$ dépendant de $\bowtie_j$. Par exemple, pour une contrainte $C_j \le 0$, on demande que $\alpha x_k \preceq_k C'_j$; les différentes contraintes sont rassemblées table 5.1. Rappelons que x_k et 0 sont toujours des expressions de E_k.

Contrainte de la garde	Contrainte dans le graphe des classes
$C < 0$	$\alpha x_k \preceq_k C' \ \wedge\ C' \not\preceq_k \alpha x_k$
$C \le 0$	$\alpha x_k \preceq_k C'$
$C = 0$	$\alpha x_k \preceq_k C' \ \wedge\ C' \preceq_k \alpha x_k$
$C \ge 0$	$C' \preceq_k \alpha x_k$
$C > 0$	$\alpha x_k \not\preceq_k C' \ \wedge\ C' \preceq_k \alpha x_k$

TABLE 5.1: Conditions sur $\preceq_k$ en fonction des opérateurs de comparaison des gardes.

Calcul du successeur. Si les conditions de franchissement sont vérifiées, on définit les préordres de $R' = \left(s', \{\preceq'_j\}_{1 \le j \le \lambda(s')}\right)$ comme suit. Soit $j \in \{1, \ldots, \lambda(s')\}$ et C_1, C_2 deux expressions de E_j.

Si $j \le k$**.** Par construction, $C_1[u]$ et $C_2[u]$ sont des expressions de E_j (étape (2.1)), et on prend $C_1 \preceq'_j C_2$ si et seulement si $C_1[u] \preceq_j C_2[u]$.

Si $j > k$**.** On pose $C_0 = C_1[u] - C_2[u] = \sum_{i=1}^{k} a_i x_i + b$ et $C = Norm(C_0, k)$. On peut écrire $C = \alpha x_k + \sum_{i=1}^{k-1} a'_i x_i + b'$ (avec $\alpha \in \{0,1\}$). Par construction des ensembles d'expressions, $C' = -\sum_{i=1}^{k-1} a'_i x_i - b'$ est une expression de E_k (étape (2.2) au niveau j).

Lorsque $a_k \ge 0$, on prend $C_1 \preceq'_j C_2$ si et seulement si $\alpha x_k \preceq_k C'$.

Lorsque $a_k < 0$, on prend $C_1 \preceq'_j C_2$ si et seulement si $C' \preceq_k \alpha x_k$.

On montre à présent que la construction ci-dessus est correcte et complète vis-à-vis des pas discrets du système de transition $\mathscr{T}_{\mathscr{A}}$.

Lemme 5.3 (Correction et complétude des transitions discrètes)**.** *Soit une configuration* $(s,v,\beta) \in [\![R]\!]$.

1. *La transition e est franchissable depuis* (s,v,β) *si et seulement si, dans* $\mathscr{G}_A$*, e vérifie les conditions de franchissement depuis la classe R ;*
2. *Si* $(s,v,\beta) \xrightarrow{e} (s',v',\beta')$ *dans* $\mathscr{T}_{\mathscr{A}}$*, alors* $R \xrightarrow{e} R'$ *dans* $\mathscr{G}_{\mathscr{A}}$ *et* $(s',v',\beta') \in [\![R']\!]$ *;*
3. *Si* $R \xrightarrow{e} R'$ *dans* $\mathscr{G}_{\mathscr{A}}$*, il existe une configuration* $(s',v',\beta') \in [\![R']\!]$ *telle que* $(s,v,\beta) \xrightarrow{e} (s',v',\beta')$ *dans* $\mathscr{T}_{\mathscr{A}}$.

Démonstration.

1. Puisque $e = \left(s \xrightarrow{g,a,u} s'\right)$ est franchissable dans $\mathscr{T}_{\mathscr{A}}$, on a $v \vDash g$. Donc pour chaque contrainte de g on a bien $v(C) \bowtie 0$, ou encore $v(x_k) \bowtie v(C')$, où C' est l'opposé de la partie de C ne contenant pas x_k. La définition de $[\![\cdot]\!]$ permet de conclure que les conditions de franchissement du graphe des classes sont satisfaites dans R. La réciproque provient de jeux de réécriture analogues.
2. Prenons deux expressions C_1 et C_2 d'un E_j pour $j \in \{1,\dots,\lambda(s')\}$. Supposons, sans perte de généralité, que $v'(C_1) \leq v'(C_2)$. Il suffit de montrer que $C_1 \preceq'_j C_2$.

   Si $j \leq k$**.** On a alors les équivalences :

$$\begin{array}{rcll} v'(C_1) & \leq & v'(C_2) & \iff \\ v[u](C_1) & \leq & v[u](C_2) & \iff \\ v(C_1[u]) & \leq & v(C_2[u]) & \iff \\ C_1[u] & \preceq_j & C_2[u] & \iff \\ C_1 & \preceq'_j & C_2. & \end{array}$$

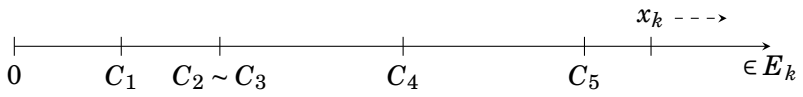
   Si $j > k$**.** On a :

$$\begin{array}{rcll} v'(C_1) = v[u](C_1) = v(C_1[u]) & \leq & v'(C_2) = v[u](C_2) = v(C_2[u]) & \iff \\ v(C_1[u]) - v(C_2[u]) & \leq & 0 & \iff \\ v(C_1[u] - C_2[u]) & \leq & 0 & \end{array}$$

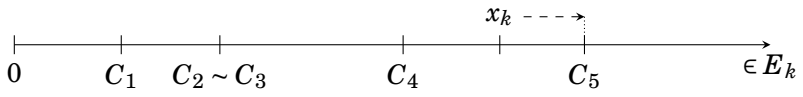
   La normalisation de $C_0 = C_1[u] - C_2[u]$ peut changer le signe du coefficient de x_k, mais ce changement de signe est rattrapé lors de la définition de $\preceq'_j$. On suppose donc sans perte de généralité que x_k est à coefficient positif dans C_0. Par normalisation au niveau k, on obtient l'expression C où x_k a le coefficient $\alpha \in \{0,1\}$ et on note C' l'opposé de la partie de C ne contenant pas x_k. On a donc $v(\alpha x_k) \leq v(C')$, c'est-à-dire $\alpha x_k \preceq_k$ d'où $C_1 \preceq'_j C_2$.
3. Si $R \xrightarrow{e} R'$, c'est en particulier que e est franchissable depuis R, donc pour toute configuration de $(s,v,\beta) \in R$. Le successeur par e de (s,v,β) dans $\mathscr{T}_{\mathscr{A}}$ existe et est bien dans $[\![R']\!]$ ce qui conclut la preuve. □

Les transitions de temps. Lors du passage du temps, seule l'horloge du niveau de l'état courant s est active. Ainsi, toutes les expressions de $Exp(s)$ hormis $x_{\lambda(s)}$ sont constantes. On pose $k = \lambda(s)$. Pour une région $R = (s, \{\preceq_i\}_{1\leq i\leq k})$, le graphe des classes contient la transition $R \xrightarrow{succ} Post(R)$ où la classe $Post(R) = \left(s, \{\preceq'_i\}_{1\leq i\leq k}\right)$ est définie de la façon suivante.

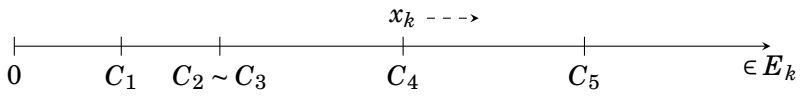
Pour $i \in \{1,\dots,k-1\}$, on prend $\preceq'_i = \preceq_i$: l'ordre entre les expressions des niveaux inférieurs, dont les valeurs sont constantes, n'évolue pas.



(a) x_k est l'expression maximale.



(b) x_k est entre deux expressions.



(c) x_k est égal à d'autres expressions.

FIGURE 5.3: Passage du temps au niveau k pour les expressions.

Soit $\sim$ la relation d'équivalence $\preceq_k \cap \preceq_k^{-1}$ induite par $\preceq_k$ sur E_k. Sur les classes d'équivalence de $\sim$, le préordre (total) $\preceq_k$ devient un ordre (total). On note V la classe d'équivalence qui contient x_k.

- Si $V = \{x_k\}$ et V est la classe maximale pour l'ordre $\preceq_k$ (figure 5.3(a)). On prend alors $\preceq'_i = \preceq_i$ (donc $Post(R) = R$).
- Si $V = \{x_k\}$ mais V n'est pas la classe maximale pour l'ordre $\preceq_k$ (figure 5.3(b)). Soit V' la classe d'équivalence suivante. On fusionne alors V et V' et on préserve $\preceq_k$ ailleurs. On prend donc, pour $C, C' \in E_k$:
  - si $C, C' \in V \cup V'$, $C \preceq'_k C'$ et $C' \preceq'_k C$;
  - si $C \notin V \cup V'$ ou $C' \notin V \cup V'$, $C \preceq'_k C'$ si et seulement si $C \preceq_k C'$.
- Si V n'est pas un singleton (figure 5.3(c)), on sépare V en $V \setminus \{x_k\}$ et $\{x_k\}$ avec $V \setminus \{x_k\} \preceq'_k \{x_k\}$. Ainsi pour $C \in V$ et $C' \notin V$ (ou $C \notin V$ et $C' \in V$), $C \preceq'_k C'$ si et seulement si $C \preceq_k C'$; pour $C \in V \setminus \{x_k\}$, $C \preceq'_k x_k$.

On montre que la définition des transitions de temps de l'automate des classes est correcte vis-à-vis des pas de temps de $\mathscr{T}_{\mathscr{A}}$.

Lemme 5.4 (Correction des transitions de temps)**.** *Soit* $(s, v, \beta) \in [\![R]\!]$. *Il existe* $d > 0$ *et* $\beta' \in \mathbb{B}$ *tels que* $(s, v + d, \beta') \in \left[\!\left[Post(R')\right]\!\right]$ *et pour tout* $d' \in [0, d']$, *il existe* $\beta'' \in \mathbb{B}$ *tel que* $(s, v + d', \beta'') \in R \cup Post(R)$.

Démonstration.

- Si $V = \{x_k\}$ et V est la classe maximale pour l'ordre $\preceq_k$, alors $Post(R) = R$ et n'importe quelle valeur de d convient. Il est alors clair que les valeurs de délais inférieures à d conservent les ordres et donc la classe.
- Si $V = \{x_k\}$ mais V n'est pas la classe maximale pour l'ordre $\preceq_k$, on prend C une expression de la classe d'équivalence suivante V'. On choisit $d = v(C) - v(x_k)$. On a $(v + d)(C) = v(C)$ et $(v + d)(x_k) = v(x_k) + d$, donc $(v + d)(C) = (v + d)(x_k)$. Pour $C' \notin V \cup V'$, on a aussi $(v + d)(C') = v(C')$. Donc $(v + d)(C') \bowtie (v + d)(x_k)$ si et seulement si $v(C') \bowtie v(x_k)$ ($\bowtie$ est un opérateur de comparaison, que l'on peut supposer strict ici car x_k est seule dans sa classe d'équivalence). Et pour

$C', C'' \neq x_k$, les valeurs ne changent pas donc les ordres entre elles non plus. Ainsi $(s, v+d, \beta') \in Post(R)$ (pour un β' quelconque).

De plus, pour $d' \in [0, d[$, on a toujours $(v+d')(C) = v(C)$ et $(v+d')(x_k) = v(x_k)+d'$, donc $(v+d')(C) > (v+d')(x_k)$. Et pour $C' \notin V \cup V'$, l'ordre entre $(v+d)(C')$ et $(v+d)(x_k)$ est toujours le même que celui entre $v(C')$ et $v(x_k)$: soit $(v+d)(C') = v(C') > v(C) = (v+d')(C) > (v+d')(x_k)$, soit $(v+d)(C') = v(C') < v(x_k) \leq (v+d)(x_k)$. Donc $(s, v+d', \beta'') \in [\![R]\!]$.

– Si V n'est pas un singleton, on prend C une expression de la classe d'équivalence suivante V'. On choisit $d = (v(C) - v(x_k))/2$. Pour $C'' \in V \setminus \{x_k\}$, on a bien $v(C') = (v+d)(C') < (v+d)(x_k) < (v+d)(C) = v(C)$. Similairement au cas précédent, les ordres entre les valuations des autres expressions et la valuation de x_k (et des autres expressions entre elles) ne sont pas changés, on a donc $(s, v+d, \beta') \in Post(R)$. On a aussi, pour $d' \in]0, d]$, $(s, v+d', \beta'') \in Post(R)$ qui se montre de la même manière. □

Lemme 5.5 (Complétude des transitions de temps)**.** *Soit $(s, v, \beta) \in [\![R]\!]$ et $d \in \mathbb{R}_{\geq 0}$ tel que $(s, v, \beta) \xrightarrow{d} (s, v+d, \beta')$ est une transition de $\mathcal{T}_{\mathcal{A}}$. Alors $(s, v+d, \beta') \in [\![R']\!]$ où pour un certain $m \in \mathbb{N}$, $R' = Post^m(R)$.*

Démonstration. On ne s'occupe que du niveau k ; les expressions aux niveaux inférieurs et les valuations d'icelles ne variant ni par passage du temps dans $\mathcal{T}_{\mathcal{A}}$ ni par la transition $\xrightarrow{succ}$ dans $\mathcal{G}_{\mathcal{A}}$.

Soit $C \in E_k \setminus \{x_k\}$; cette expression ne porte que sur les variables $\{x_1, \ldots, x_{k-1}\}$. Donc $(v+d)(C) = v(C)$. Ainsi l'ordre entre les expressions autres que x_k est inchangé. De plus, si $v(x_k) \geq v(C)$ alors $(v+d)(x_k) \geq (v+d)(C)$. Or les successeurs de R par $\xrightarrow{succ}$ sont les exactement les classes pour lesquelles le préordre entre les expressions autres que x_k est inchangé et x_k peut « dépasser » les autres expressions. □

La gestion des politiques. On a jusque là supposé que les politiques des états étaient toutes paresseuses. Ainsi la valeur de β dans une configuration (s, v, β) n'entrait pas en jeu. Nous montrons maintenant comment modifier le graphe $\mathcal{G}_{\mathcal{A}}$ afin de prendre en compte ces politiques.

Cas des politiques urgentes. Le cas d'une classe $R = \left(s, \{\preceq_k\}_{1 \leq k \leq \lambda(s)}\right)$ où $pol(s) = U$ est simplement traité en supprimant toutes les transitions $R \xrightarrow{succ} Post(R)$ du graphe des classes. De plus, les configurations $(s, v, \bot)$ ne sont dans ce cas jamais atteintes. La classe R représente donc l'ensemble de configurations

$$[\![R]\!] = \left\{(s, v, \bot) \,\middle|\, \forall k \in \{1, \ldots, \lambda(s)\},\ \forall C, C' \in E_k,\ v(C) \leq v(C') \text{ ssi } C \preceq_k C'\right\}.$$

Cas des politiques retardées. On considère différement les classes selon qu'elles sont *ouvertes* ou *fermées* pour le temps. Soit $R = \left(s, \{\preceq_k\}_{1 \leq k \leq \lambda(s)}\right)$ où $pol(s) = D$. La classe R est dite *ouverte* si $x_{\lambda(s)}$ est seul dans la classe d'équivalence induite par $\sim$ (avec $\sim = \preceq_{\lambda(s)} \cap \preceq_{\lambda(s)}^{-1}$ comme dans la construction de la classe $Post(R)$ ci-dessus). Dans une classe ouverte, un écoulement du temps est possible (parfois court) sans changer de classe. La classe R est dite *fermée* si $x_{\lambda(s)}$ n'est pas seul dans la classe d'équivalence induite par $\sim$. Dans une classe fermée, tout écoulement de temps fait changer de classe.

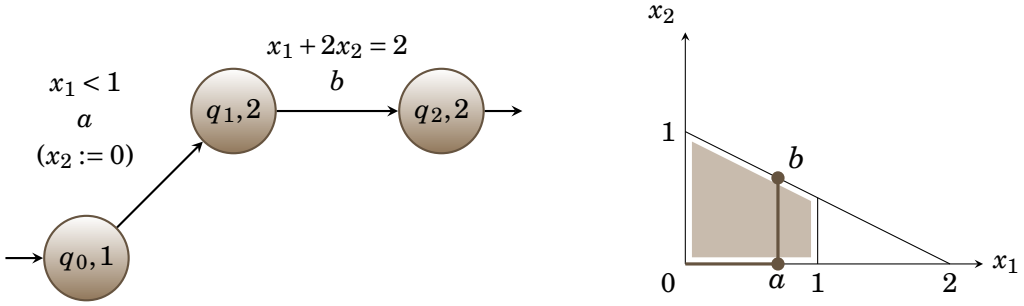
Puisque les classes ouvertes autorisent toujours l'écoulement de temps, celles-ci sont conservées telles quelles : avant de franchir une transition discrète depuis $(s,v,\bot) \in [\![R]\!]$, on fait s'écouler du temps, ce qui correspond à un pas $(s,v,\bot) \to (s,v+d,\top)$ dans $\mathscr{T}_{\mathscr{A}}$.

Si R est fermée pour le temps, on la sépare en deux copies R^- et R^+, qui gardent en mémoire si du temps s'est écoulé depuis le dernier franchissement d'une transition discrète. Ainsi

$$[\![R^-]\!] = \big\{(s,v,\bot) \,\big|\, \forall k \in \{1,\dots,\lambda(s)\},\ \forall C,C' \in E_k,\ v(C) \le v(C') \text{ ssi } C \preceq_k C'\big\},$$
$$[\![R^+]\!] = \big\{(s,v,\top) \,\big|\, \forall k \in \{1,\dots,\lambda(s)\},\ \forall C,C' \in E_k,\ v(C) \le v(C') \text{ ssi } C \preceq_k C'\big\}.$$

Une transition de temps qui entrait dans R est donc redirigée vers R^+ tandis qu'une transition discrète entrant dans R est redirigée ver R^-. Les transitions de temps $R \xrightarrow{succ} R'$ sont remplacées par deux transitions de temps $R^- \xrightarrow{succ} R'$ et $R^+ \xrightarrow{succ} R'$. Les transitions discrètes $R \xrightarrow{e} R'$ sont remplacées par la seule transition $R^+ \xrightarrow{e} R'$. Ces remplacements sont fait successivement sur toutes les classes dont la politique est retardée. Puisque chaque classe a au plus une seule transition de temps sortante, ces remplacements ne font, au plus, que doubler la taille du graphe des classes.

Exemple de graphe des classes. On illustre la construction du graphe des classes sur l'ITA $\mathscr{A}_{15}$ reproduit figure 5.4(a). Rappelons que les ensembles d'expressions obtenus pour cet automate sont $E_1 = \{x_1, 0, 1, 2\}$ et $E_2 = \{x_2, 0, -\frac{1}{2}x_1 + 1\}$.



(a) L'ITA $\mathscr{A}_{15}$ à deux niveaux d'interruption. (b) Une exécution possible de $\mathscr{A}_{15}$.

FIGURE 5.4: Un exemple d'ITA avec une exécution possible.

Dans l'état q_0, seule l'horloge x_1 est pertinente. Les préordres possibles sur E_1 sont :

$$Z_0 : x_1 = 0 < 1 < 2 \qquad Z_1 : 0 < x_1 < 1 < 2 \qquad Z_2 : 0 < x_1 = 1 < 2$$
$$Z_3 : 0 < 1 < x_1 < 2 \qquad Z_4 : 0 < 1 < x_1 = 2 \qquad Z_5 : 0 < 1 < 2 < x_1.$$

On note R_0^i la classe (q_0, Z_i), pour $i \in \{0,\dots,5\}$. La classe initiale est donc R_0^0, et R_0^{i+1} est le successeur par une transition de temps de R_0^i pour $i \in \{0,\dots,4\}$, R_0^5 étant son propre successeur par transition de temps.

La transition a vers l'état q_1 peut être franchie depuis les classes R_0^0 et R_0^1, mais pas depuis $R_0^2,\dots,R_0^5$, car la contrainte $x_1 < 1$ n'y est pas vérifiée.

On note R_1^0 (respectivement R_1^1) le successeur de R_0^0 (respectivement R_0^1) par a. Dans R_1^0 comme dans R_1^1, on a $x_2 = 0$. Il reste à calculer le préordre entre $0 = x_2$ et

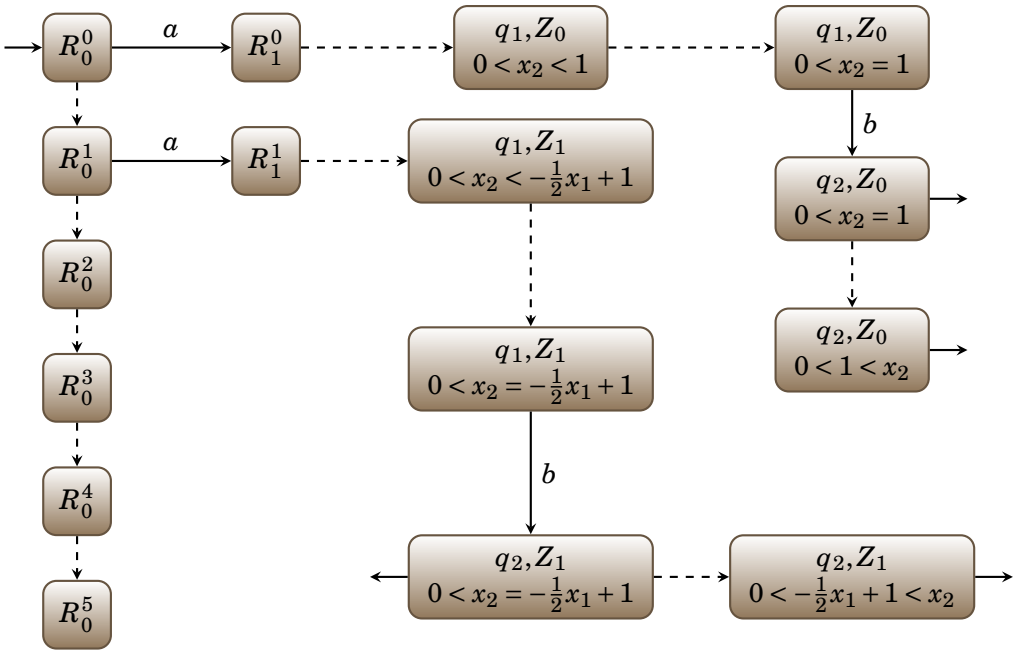


FIGURE 5.5: Graphe des classes $\mathscr{G}_{\mathscr{A}_{15}}$.

$-\frac{1}{2}x_1 + 1$. On calcule pour cela la différence entre les (mises à jour des) expressions $0 = x_2$ et $-\frac{1}{2}x_1 + 1$, et en particulier son signe selon le préordre sur E_1 dans la classe source. Ainsi, pour R_0^0, le signe de $0 - (-\frac{1}{2}x_1 + 1) = \frac{1}{2}x_1 - 1$ est négatif car $x_1 = 0$ (l'expression $-\frac{1}{2}x_1 + 1$ vaut donc ici 1). De même, dans R_0^1, le signe de $\frac{1}{2}x_1 - 1$ est négatif car $x_1 < 2$. Dans les deux cas, on a donc l'ordre $0 = x_2 < -\frac{1}{2}x_1 + 1$ sur E_2.

La transition b n'est franchissable que depuis leurs successeurs par transitions de temps dans lesquels $x_2 = -\frac{1}{2} + 1$, ce qui correspond à la satisfaction de la garde.

Le graphe des classes $\mathscr{G}_{\mathscr{A}_{15}}$, abstraction de $\mathscr{A}_{15}$, est représenté figure 5.5. Les transitions de temps y sont représentées en pointillés.

Sur la représentation géométrique de la figure 5.4(b), la trajectoire correspond au chemin suivant dans le graphe des classes :

$$R_0^0 \to R_0^1 \xrightarrow{a} R_1^1 \to \left(q_1, Z_1, 0 < x_2 < -\frac{1}{2}x_1 + 1\right) \to$$

$$\left(q_1, Z_1, 0 < x_2 = -\frac{1}{2}x_1 + 1\right) \xrightarrow{b} \left(q_2, Z_1, 0 < x_2 = -\frac{1}{2}x_1 + 1\right)$$

Le problème d'accessibilité

Le calcul du graphe des classes fournit une preuve de la régularité du langage non-temporisé de $\mathscr{A}$. Il permet aussi de calculer des propriétés d'accessibilité d'un état de contrôle.

Pour savoir si un état s_f est atteignable dans $\mathscr{A}$, on devine un chemin dans le graphe des classes $\mathscr{G}_{\mathscr{A}}$ (sans le construire explicitement). On ne garde en mémoire que la classe courante, dont la taille est majorée par le nombre d'expressions qu'elle contient, c'est-à-dire en $O\left((T+2)^{2^{n^2}}\right)$. Cette procédure de recherche a donc une com-

plexité en 2-EXPSPACE. Cependant, lorsque l'on fixe le nombre d'horloges, le nombre d'expressions, et donc la complexité de cet algorithme pour le test de l'accessibilité, devient polynomial. On a donc, lorsque le nombre d'horloges est fixé, une complexité PSPACE.

Proposition 5.6 (Accessibilité dans les ITA)**.** *Le problème d'accessibilité dans un ITA est décidable en 2-EXPSPACE et en PSPACE lorsque le nombre d'horloges est fixé.*

Ces deux bornes de complexité sont améliorées dans la section 5.3.

5.3 ITA restreints

Si l'on considère que chaque niveau d'interruption est un niveau de sécurité, le modèle des ITA autorise un niveau à lire et à écrire des valeurs (d'horloge) dans les niveaux inférieurs. C'est par exemple le cas dans l'automate $\mathscr{A}_{16}$ de la figure 5.6 où x_1 est remis à jour au niveaux 2 et 3 et x_2 est remise à jour au niveau 3.

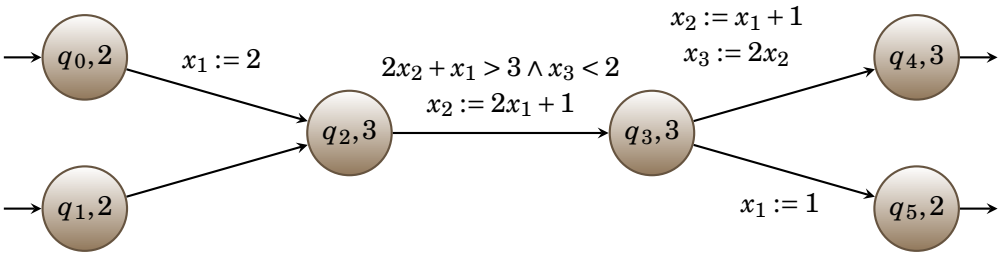


FIGURE 5.6: ITA $\mathscr{A}_{16}$ contenant des mises à jour d'horloges inactives.

Cependant, si l'on considère qu'à chaque niveau correspond une tâche interrompue par un niveau supérieur, il est plus conforme à la réalité de n'autoriser que l'horloge active à subir des mises à jour. Les horloges interrompues ne peuvent pas changer de valeur par l'intermédiaire d'une mise à jour. On définit donc dans cette section le modèle des *automates temporisés à interruptions restreints* (ITA$_-$), où les valeurs des horloges de niveaux inférieurs peuvent être lues, mais pas mises à jour.

La restriction que l'on impose sur les ITA$_-$ comporte aussi des avantages d'un point de vue théorique. On montre qu'ils sont équivalents du point de vue des langages temporisés aux ITA, bien que moins succincts (la traduction est doublement exponentielle). De plus, une procédure d'accessibilité dédiée aux ITA$_-$, en temps exponentiel, permet d'atteindre une borne supérieure doublement exponentielle pour la complexité dans les ITA.

5.3.1 Le modèles des ITA restreints

Les mises à jour « complexes » (dans $\mathscr{U}(X) \setminus \mathscr{U}_0(X)$) d'un ITA$_-$ ne se produisent donc que pour le niveau actif, et uniquement lorsque le niveau ne décroît pas.

Définition 5.4 (ITA restreint)**.** *La sous-classe des* automates temporisés à interruptions restreints *(ITA$_-$) est définie par les restrictions suivantes sur les mises à jour. Un ITA $\mathscr{A} = \langle S, Lab, X, \lambda, pol, \Delta, s_I, F, AP, prop \rangle$ à n horloges est un ITA$_-$ si pour toute*

transition $s \xrightarrow{g,a,u} s' \in \Delta$ *(avec* $k = \lambda(s)$ *et* $k' = \lambda(s')$*), la mise à jour* u *est de la forme* $\bigwedge_{i=1}^{n} x_i := C_i$ *où les* C_i *sont tels que :*

- *si* $k > k'$ *(le niveau décroît), pour* $i \in \{1,\dots,k'\}$ $C_i = x_i$ *(pas de mise à jour) et pour* $i \in \{k'+1,\dots,n\}$, $C_i = 0$ *(remise à zéro des niveaux intermédiaires* [6]*) ;*
- *si* $k \leq k'$*, pour* $i \in \{1,\dots,k-1\}$ $C_i = x_i$ *(pas de mise à jour),* $C_k = \sum_{i=1}^{k-1} a_i x_i + b$ *ou* $C_k = x_k$ *(potentielle mise à jour de l'horloge de niveau d'origine) et pour* $i \in \{k+1,\dots,n\}$, $C_i = 0$ *(remise à zéro des niveaux intermédiaires).*

Dans la suite, nous donnons une traduction des ITA vers les ITA$_-$ (section 5.3.2) et une procédure d'accessibilité efficace sur les ITA$_-$ (section 5.3.3). La combinaison de ces résultats dans la preuve du théorème suivant (section 5.3.4) nous permet d'obtenir une meilleurs complexité pour le problème d'accessibilité dans les ITA :

Théorème 5.7 (Accessibilité dans les ITA)**.** *Le problème d'accessibilité dans les ITA peut se résoudre en 2-NEXPTIME et en NP lorsque le nombre d'horloges est fixé.*

Remarquons que le problème d'accessibilité dans les automates temporisés est PSPACE-complet, et donc plus facile que pour les ITA. Cependant, pour les automates temporisés, cette complexité reste la même lorsque l'on fixe le nombre d'horloges (dès qu'elles sont au moins trois) [CY92]. Dans les ITA, la complexité à nombre d'horloges fixé est beaucoup plus petite (NP). Ainsi, dans les ITA la source de complexité est le nombre d'horloges, tandis que dans les automates temporisés, la source de complexité est l'encodage binaire des constantes.

5.3.2 Traduction des ITA en ITA$_-$

On montre maintenant que le modèle des ITA$_-$ est aussi expressif que celui des ITA, malgré une explosion doublement exponentielle de la taille du modèle.

Proposition 5.8 (Équivalence des ITA et des ITA$_-$)**.** *Étant donné un ITA* $\mathscr{A}$ *à* n *horloges et des transitions de taille* T*, on peut construire un ITA*$_-$ $\mathscr{A}'$ *à* n *horloges et dont les transitions sont de taille* T' *tel que :*

- $\mathscr{A}$ *et* $\mathscr{A}'$ *acceptent le même langage temporisé :* $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{A}')$*;*
- *la taille de* $\mathscr{A}'$ *est doublement exponentielle par rapport à celle de* $\mathscr{A}$ *: si* $T \geq 2$*,* $T' \leq T^{2^{(2n+1)(\log(n)+1)}}$ *et si* $|S| \geq 2$*,* $|S'| \leq |S|^{2^{(2n+1)(\log(n)+1)}}$.

Construction de $\mathscr{A}'$

La transformation de $\mathscr{A}$ en $\mathscr{A}'$ consiste à ne pas appliquer les mises à jour interdites par le modèle des ITA$_-$, mais à les garder en mémoire dans les états, sous la forme d'une expression par horloge de niveau inférieur. Lors d'une descente de niveau, les mises à jours en attente sont alors appliquées au nouveau niveau courant.

Calcul des expressions. Afin de mémoriser les mises à jour des niveaux plus bas que le niveau courant, on définit les ensembles d'expressions $\{F_{i,j}\}_{1 \leq j \leq i \leq n}$. Intuitivement, l'ensemble $F_{i,j}$ contient toutes les mises à jour possibles de l'horloge x_j qui

6. Pour les niveaux supérieurs à k, les horloges étant déjà à 0 une remise à zéro est équivalente à une absence de mise à jour.

$i \setminus j$	1	2	3
1	$\{x_1\}$		
2	$\{x_1, 2\}$	$\{x_2\}$	
3	$\{x_1, 2, 1\}$	$\{x_2, \underbrace{2x_1+1,\ 5,\ 3}_{q_2 \xrightarrow{x_2:=2x_1+1} q_3},\ \underbrace{x_1+1,\ 3,\ 2}_{q_3 \xrightarrow{x_2:=x_1+1} q_4}\}$	$\{x_3\}$

TABLE 5.2: Ensembles d'expressions $F_{i,j}$ pour la transformation de $\mathscr{A}_{16}$ en $\mathscr{A}'_{16}$.

a pu être faite au niveau i, mais en prenant en compte les possibles mises à jour antérieures.

Pour $i \in \{1,\dots,n\}$ et $j \in \{1,\dots,i\}$, on définit l'ensemble $F_{i,j}$ inductivement par :
- $F_{i,i} = \{x_i\}$;
- si $j < i$,

$$F_{i,j} \;=\; F_{i-1,j} \cup \left\{ C\left[\bigwedge_{k=1}^{j-1} x_k := C_k\right] \;\middle|\; \begin{array}{l} \exists s \xrightarrow{g,a,u} s' \in \Delta,\ \lambda(s) = i, (x_j := C) \in u \\ \text{et pour } k \in \{1,\dots,j-1\}, C_k \in F_{i,k} \end{array} \right\}.$$

  Ainsi pour chaque expression apparaissant dans une mise à jour de x_j dans une garde de niveau i, la valeur d'une horloge x_k de niveau inférieur à j peut être remplacée par une mise à jour antérieure de x_k.

On note $F_j = F_{n,j} = \bigcup_{i=j}^{n} F_{i,j}$. L'ensemble F_j contient l'ensemble des expressions des mises à jour de x_j dans des niveaux supérieurs.

La construction précédente est illustrée sur l'ITA $\mathscr{A}_{16}$ de la figure 5.6. Les ensembles d'expressions $F_{i,j}$ pour $1 \le j \le i \le 3$ sont calculés sur la table 5.2 ; l'expression 3 apparaît deux fois dans $F_{3,2}$ car elle correspond à la fois à $(2x_1+1)[x_1 := 1]$ et à $(x_1+1)[x_1 := 2]$. Les expressions à mémoriser dans les états sont donc $F_1 = F_{3,1} = \{x_1, 1, 2\}$ pour x_1 et $F_2 = F_{3,2} = \{x_2, 2x_1+1, x_1+1, 2, 3, 5\}$ pour x_2. Aucune expression n'a besoin d'être mémorisée pour x_3 car on ne retarde jamais de mise à jour pour cette horloge.

La taille de ces ensembles d'expressions est doublement exponentielle dans le nombre d'horloges de $\mathscr{A}$ et polynomial dans son nombre de transitions. En effet, on montre que $|F_{i,j}| \le T^{2^{i+j} \times (j-1)!}$. La preuve procède par induction sur la somme $i+j$. Si $i+j = 2$, alors $i = j = 1$ et $|F_{1,1}| = |\{x_1\}| = 1 \le T^{2^2 \times 0!}$. Supposons que pour $i+j < \ell$, $F_{i,j} \le T^{2^{i+j} \times (j-1)!}$. Soit $i+j = \ell$:

$$\begin{array}{rcl} |F_{i,j}| & \le & |F_{i-1,j}| + T \times \prod_{k=1}^{j-1} |F_{i,k}| \\ & \le & T^{2^{i+j-1} \times (j-1)!} + T \times \prod_{k=1}^{j-1} T^{2^{i+k} \times (k-1)!} \\ & \le & T^{2^{i+j-1} \times (j-1)!} + T \times \left(T^{2^{i+j-1} \times (j-2)!}\right)^{j-1} \\ & \le & T^{2^{i+j-1} \times (j-1)!} + T \times T^{2^{i+j-1} \times (j-1)!} \\ & \le & T^{2^{i+j-1} \times (j-1)!} \times (T+1) \\ |F_{i,j}| & \le & T^{2^{i+j} \times (j-1)!} \end{array}$$

Définition de $\mathscr{A}'$. Pour $\mathscr{A} = \langle S, Lab, X, \lambda, pol, \Delta, s_I, F, AP, prop\rangle$, on définit l'ITA$_-$ $\mathscr{A}' = \langle S', Lab, X, \lambda', pol', \Delta', s'_I, F', AP, prop'\rangle$ de la façon suivante.

Les états. L'ensemble des états est

$$\begin{aligned} S' \;=\; & \left\{(s^+, f_1, \ldots, f_{i-1}) \,\middle|\, s \in S, \lambda(s) = i \text{ et } \forall j \in \{1,\ldots,i-1\}, f_j \in F_j\right\} \\ & \cup \left\{(s^-, f_1, \ldots, f_i) \,\middle|\, s \in S, \lambda(s) = i \text{ et } \forall j \in \{1,\ldots,i\}, f_j \in F_j\right\} \end{aligned}$$

avec $\lambda'(s^+, f_1, \ldots, f_{i-1}) = \lambda'(s^-, f_1, \ldots, f_i) = \lambda(s)$. De même :

$$prop'(s^+, f_1, \ldots, f_{i-1}) = prop'(s^-, f_1, \ldots, f_i) = prop(s).$$

On prend $pol'(s^+, f_1, \ldots, f_{i-1}) = pol(s)$ et $pol'(s^-, f_1, \ldots, f_i) = U$. Les états s^+ mémorisent les mises à jour pour les niveaux inférieurs. Les états s^- sont des états intermédiaires lors d'une baisse de niveau. En effet, les mises à jour de x_i ne peuvent être appliquées que *depuis* le niveau i, selon les restrictions imposées aux ITA$_-$. On garde donc celles-ci encore en mémoire dans s^-.

L'état initial de $\mathscr{A}'$ est $s'_I = (s_I^+, x_1, \ldots, x_{\lambda(s_I)-1})$. Les états finaux sont les états non intermédiaires dont la première composante correspond à un état final :

$$F' \;=\; \left\{(s^+, f_1, \ldots, f_{i-1}) \,\middle|\, s \in F, \lambda(s) = i \text{ et } \forall j \in \{1,\ldots,i-1\}, f_j \in F_j\right\}.$$

Pour un état $s' = (s^+, f_1, \ldots, f_{\lambda(s)-1})$ on note $u_0(s')$ la mise à jour

$$u_0(s') \;=\; \left(\bigwedge_{j=1}^{\lambda(s)-1} x_j := f_j\right) \wedge \left(\bigwedge_{j=\lambda(s)}^{n} x_j := x_j\right).$$

Les transitions. Soit $s_1 \xrightarrow{g,a,u} s_2 \in \Delta$ une transition de $\mathscr{A}$ avec $i = \lambda(s_1)$, $i' = \lambda(s_2)$. On écrit $u = \bigwedge_{j=1}^{n} x_j := C_j$; pour $j \in \{i+1, \ldots, n\}$, on peut supposer $C_j = x_j$.

- Si $i \le i'$. Alors pour chaque état $s' = (s_1^+, f_1, \ldots, f_{i-1})$ on a une transition

$$(s_1^+, f_1, \ldots, f_{i-1}) \xrightarrow{g',a,u'} (s_2^+, f'_1, \ldots, f'_{i'-1}) \in \Delta'$$

  où
  - $g' = g[u_0(s')]$ (chaque contrainte $C \bowtie 0$ est remplacée par $C[u_0(s')] \bowtie 0$) : les mises à jour retardées sont effectuées à la volée et intégrées dans la garde ;
  - $u' = x_i := C_i[u_0(s')] \wedge \bigwedge_{j \neq i} x_j := x_j$: seule la mise à jour de x_i reste, prenant en compte les mises à jour retardées des variables apparaissant dans C_i ;
  - pour $j \in \{1, \ldots, i-1\}$, $f'_j = C_j[u_0(s')]$: les mises à jours retardées sont mémorisées dans l'état cible, en prenant en compte les mises à jours antérieures ;
  - pour $j \in \{i, \ldots, i'-1\}$, $f'_j = x_j$: tant qu'il n'y a pas de mises à jour retardée pour x_j, l'expression associée est elle même.
- Si $i > i'$, c'est-à-dire lorsque le niveau courant diminue. Alors pour chaque état $s' = (s_1^+, e_1, \ldots, e_{i-1})$ on a une transition

$$(s_1^+, f_1, \ldots, f_{i-1}) \xrightarrow{g',a,u'} (s_2^-, f'_1, \ldots, f'_{i'}) \in \Delta'$$

  où
  - $g' = g[u_0(s')]$;
  - $u' = \bigwedge_{j=1}^{n} x_j := x_j$: on ne fait que des mises à jour triviales ;
  - pour $j \in \{1, \ldots, i'\}$, $f'_j = C_j[u_0(s')]$.

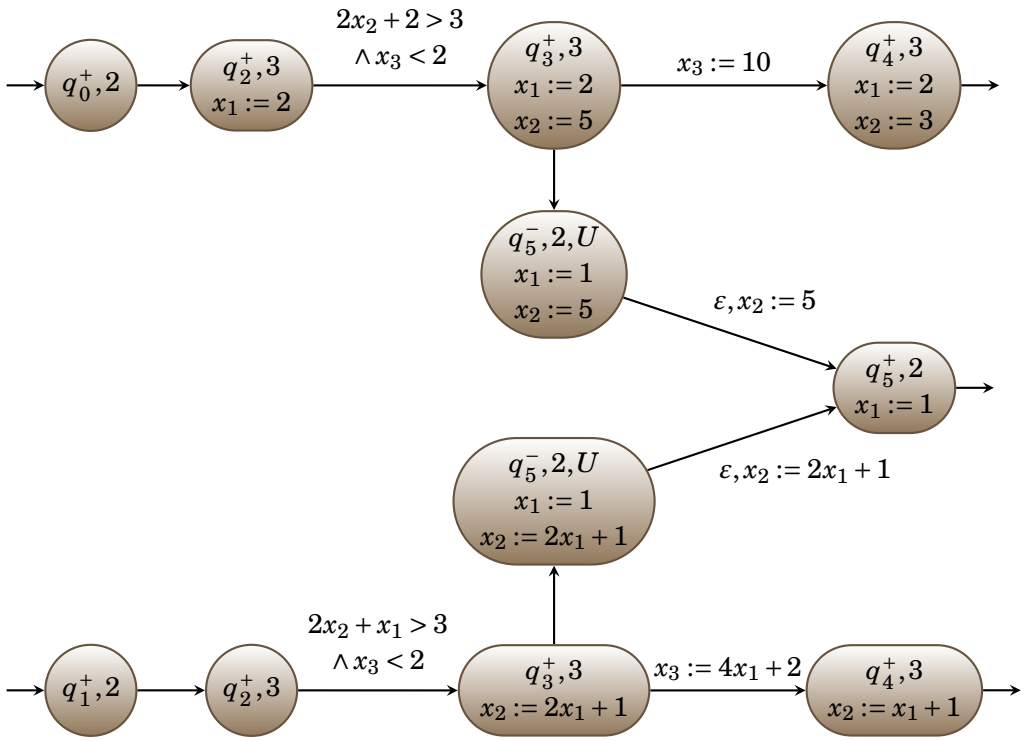


FIGURE 5.7: ITA$_-$ $\mathscr{A}'_{16}$ équivalent à $\mathscr{A}_{16}$.

– Pour chaque état $(s^-, f_1, \ldots, f_i)$ on a une transition

$$(s^-, f_1, \ldots, f_i) \xrightarrow{\top, \varepsilon, x_i := f_i} (s^+, f_1, \ldots, f_{i-1}) \in \Delta'$$

qui effectue immédiatement après une diminution de niveau (à cause de la politique urgente des états s^-) la mise à jour de x_i qui était retardée.

L'automate temporisé à interruptions restreint $\mathscr{A}'_{16}$ équivalent à $\mathscr{A}_{16}$ construit par la méthode ci-dessus est représenté figure 5.7.

Il est clair que cette construction conserve les langages temporisés. En effet, chaque transition n'est franchie que si les valeurs des horloges prenant en compte les mises à jour avaient satisfait la garde. Plus précisément, tous les chemins dans un ITA sont conservés par cette traduction. Soit ρ une exécution dans $\mathscr{A}$ et ρ' l'exécution correspondante dans $\mathscr{A}'$, c'est-à-dire l'exécution acceptant le même mot temporisé que ρ et traversant les mêmes états (quoique étendus par les expressions). La suite des états traversés par ρ' peut être projetée sur celle de ρ, en omettant les états de la forme $(s^-, -)$: à la sous-séquence

$$(s_0^+, -) \to \cdots \to (s_{m-1}^+, -) \to (s_m^-, -) \to (s_m^+, -)$$

dans ρ' correspond la sous-séquence $s_0 \to \cdots \to s_{m-1} \to s_m$ dans ρ.

Il faut cependant remarquer que le système de transition temporisé généré par $\mathscr{A}'$ n'est pas le même que celui de $\mathscr{A}$. En effet, les mises à jour retardées n'affectent pas les valuations de la même manière sur l'ITA$_-$ que sur l'ITA original.

Ainsi la traduction d'un ITA en ITA$_-$ préserve bien les exécutions mais pas les valuations des configurations. Dans le cas du *model-checking* de propriétés temporisés (voir section 5.4), il faut donc distinguer les logiques dont la sémantique dépend des exécutions de l'ITA et celles dont la sémantique dépend des valuations rencontrées. Si la sémantique porte sur les exécutions, la procédure de vérification peut se faire sur l'ITA$_-$ équivalent. Si la sémantique porte sur les valuations, l'évaluation d'une formule doit se faire directement sur l'ITA.

Complexité. Chaque transition de $\mathscr{A}$ est séparée en plusieurs transitions dans $\mathscr{A}'$. Le nombre de transitions de $\mathscr{A}'$ correspondant à $s_1 \xrightarrow{g,a,u} s_2 \in \Delta$, est borné par le nombre d'états de $\mathscr{A}'$ associés à s_1, c'est-à-dire le nombre d'expressions associés à chaque horloge x_j pour $j \in \{1,\dots,\lambda(s_1)\}$. De plus, dans le cas des transitions descendant de niveau, on double le nombre de transition correspondant dans $\mathscr{A}'$. Donc le nombre T' de transitions de $\mathscr{A}'$ est borné par :

$$\begin{aligned}
T' &\leq 2 \times T \times |F_n|^n \\
&\leq 2 \times T \times \left(T^{2^{2n} \times (n-1)!}\right)^n \\
&\leq 2 \times T \times T^{2^{2n} \times n^n} \\
&\leq T^{2^{2n+n\log(n)}+2} \quad (\text{si } T \geq 2) \\
&\leq T^{2^{2n+n\log(n)+1}} \\
T' &\leq T^{2^{n(\log(n)+3)}}
\end{aligned}$$

Le calcul du nombre d'états est similaire : chaque état de niveau i est séparé en au plus $2 \times |F_n|^i$ états dans $\mathscr{A}'$ et donc $|S'| \leq |S|^{2^{n(\log(n)+3)}}$ (en ayant supposé $|S| \geq 2$).

5.3.3 L'accessibilité sur les ITA$_-$

La procédure d'accessibilité sur les ITA$_-$ repose sur un argument de comptage qui ne dépend pas de la sémantique des ITA$_-$, hormis pour ce qui est des politiques.

Rappelons que le niveau (respectivement la politique) d'une transition est le niveau (respectivement la politique) de l'état source de la transition. Les fonctions λ et *pol* s'étendent donc aux transitions.

Lemme 5.9 (Lemme de comptage)**.** *Soit $\mathscr{A}$ un ITA$_-$ avec n horloges et T transitions. Pour une séquence de transitions $e_1 \cdots e_\ell$ dans $\mathscr{A}$ avec $\ell > (T+n)^{3n}$, il existe deux indices $i < j$ avec $e_i = e_j$ de niveau k tels que pour $m \in \{i,\dots,j\}$, $\lambda(e_m) \geq k$ et :*

- *soit e_i met à jour x_k non trivialement ;*
- *ou alors pour $m \in \{i,\dots,j\}$, e_m ne met pas à jour x_k (non trivialement) et la politique de e_i est soit retardée soit paresseuse ;*
- *ou pour $m \in \{i,\dots,j\}$, e_m ne met pas à jour x_k (non trivialement) et le temps ne s'écoule pas au niveau k entre e_i et e_j.*

Démonstration. Supposons que les conclusions du lemme ne soient pas satisfaites. Montrons que $\ell \leq (T+1)^{3n}$.

Soit $k \in \{1,\dots,n\}$ un niveau dans $\mathscr{A}$. On montre tout d'abord qu'il y a au plus $(T+1)^3$ transitions de niveau k qui apparaissent entre deux occurrences de transitions

de niveau strictement inférieur. C'est-à-dire si e_i et e_j (avec $i < j$) sont des transitions de niveau $< k$ telles que pour $m \in \{i+1, \ldots, j-1\}$, $\lambda(e_m) \geq k$, alors $\sigma = e_{i+1} \cdots e_{j-1}$ contient au plus $(T+1)^3$ transitions de niveau k. En effet, il ne peut y avoir dans σ qu'au plus T transitions $e_{\alpha(1)}, \ldots, e_{\alpha(p)}$ de niveau k qui mettent à jour x_k : autrement l'une serait répétée et le premier cas du lemme serait vérifié. On note $e_{\alpha(0)} = e_{i+1}$ le début de σ et $e_{\alpha(p+1)} = e_{j-1}$ sa fin.

Pour la même raison, dans σ, entre $e_{\alpha(q)}$ et $e_{\alpha(q+1)}$ (pour $q \in \{0, \ldots, p\}$), il y a au plus T transitions $e_{\beta(1)}, \ldots, e_{\beta(p')}$ de niveau k et de politique retardée ou paresseuse qui ne mettent pas x_k à jour. On note de même $e_{\beta(0)} = e_{\alpha(q)}$ le début de cette sous-séquence et $e_{\beta(p'+1)} = e_{\alpha(q+1)}$ sa fin.

Entre $e_{\beta(q')}$ et $e_{\beta(q'+1)}$ (pour $q' \in \{0, \ldots, p'\}$), il y a au plus T transitions instantanées de niveau k, car celles-ci ne laissent pas le temps s'écouler au niveau k.

Le nombre $|\sigma|_k$ de transitions de niveau k dans σ est donc borné par

$$\begin{aligned} |\sigma|_k &\leq T + T(T+1) + T(T(T+1)+1) \\ &\leq T + T^2 + T + T^3 + T^2 + T \\ &\leq T^3 + 2T^2 + 3T \\ |\sigma|_k &\leq (T+1)^3. \end{aligned}$$

On montre maintenant par induction que le nombre de transitions de niveau inférieur ou égal à k dans $e_1 \cdots e_\ell$ est inférieure à $(T+k)^{3k}$. Dans le cas de $k = 1$, cela revient à la preuve précédente (on a alors $\sigma = e_1 \cdots e_\ell$).

Supposons que la borne soit vraie pour un certain k. On exhibe dans $e_1 \cdots e_\ell$ les sous-séquences de niveau strictement supérieur à k :

$$e_1 \cdots e_\ell = \sigma_0 \cdot e_{\alpha(1)} \cdot \sigma_1 \cdots e_{\alpha(p)} \cdot \sigma_p$$

où pour $i \in \{0, \ldots, p\}$, σ_i ne contient que des transitions de niveau $> k$ et $\lambda\left(e_{\alpha(i)}\right) \leq k$. En utilisant la preuve ci-dessus, on obtient que chaque σ_i contient au plus $(T+1)^3$ transitions de niveau $k+1$. De plus, par induction, on sait que $p \leq (T+k)^{3k}$. Il y a ainsi au plus $\left((T+k)^{3k}+1\right)(T+1)^3$ transitions de niveau $k+1$. Donc le nombre $|e_1 \cdots e_\ell|_{\leq k+1}$ de transitions de niveau inférieur ou égal à $k+1$ est borné par

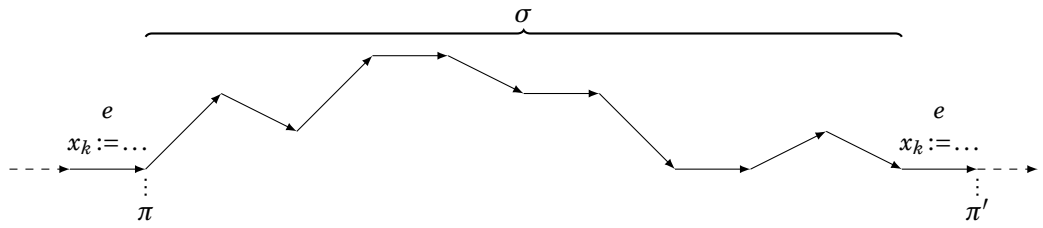
$$\begin{aligned} |e_1 \cdots e_\ell|_{\leq k+1} &\leq (T+k)^{3k} + \left((T+k)^{3k}+1\right)(T+1)^3 \\ &\leq \left((T+k)^3\right)^{k+1} + 2\left((T+1)^3\right)^k \\ &\leq \left((T+k)^3+1\right)^{k+1} \\ |e_1 \cdots e_\ell|_{\leq k+1} &\leq (T+k+1)^{3(k+1)}. \end{aligned}$$

Ainsi $\ell = |e_1 \cdots e_\ell|_{\leq n} \leq (T+n)^{3n}$, ce qui conclut la preuve. □

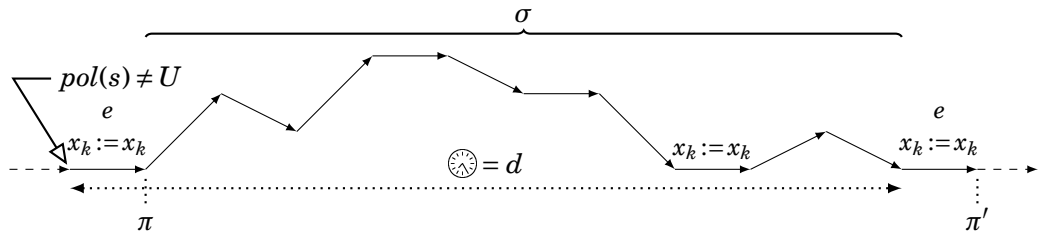
La procédure d'accessibilité fonctionne de la façon suivante. Soit $\mathscr{A}$ un ITA$_-$ à n horloges et dont les transitions sont de taille T. On devine un chemin ρ de longueur $\ell \leq (T+n)^{3n}$. Afin de vérifier que ce chemin fournit une exécution de $\mathscr{A}$, on construit un programme linéaire dont les variables sont :

$$\left\{x_i^j\right\}_{\substack{1 \leq i \leq n \\ 1 \leq j \leq 2\ell}} \cup \left\{d_j\right\}_{1 \leq j \leq 2\ell}.$$

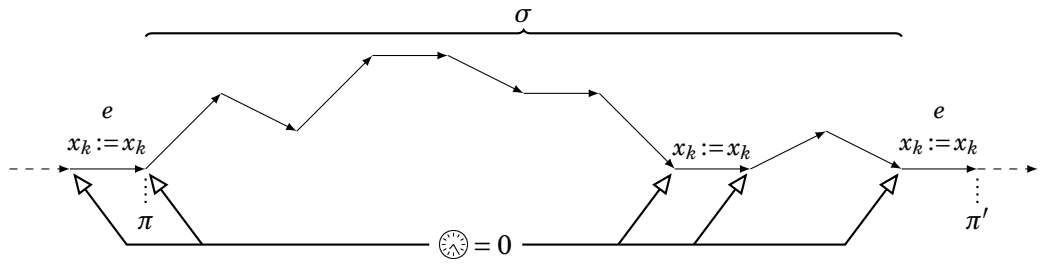
La variable x_i^j correspond à la valeur de x_i après le j-ième pas (discret ou de temps). La variable d_j correspond au délai associé au j-ème pas si celui-ci est un pas de



(a) Répétition d'une mise à jour. Les configurations en π et π' sont identiques. On peut supprimer ou dupliquer σe.



(b) Répétition sans mise à jour avec passage de temps possible. La configuration en π' diffère de celle en π par d unités de temps pour x_k. On peut remplacer $e\sigma$ par un pas de temps de durée d.



(c) Répétition sans mise à jour et sans écoulement du temps au niveau k. Les configurations en π et en π' sont identiques. On peut supprimer ou dupliquer σe.

FIGURE 5.8: Illustration des trois cas du lemme 5.9.

temps (elle est inutilisée sinon). Les équations et inéquations sont déduites des gardes et mises à jour des transitions traversées par ρ, ainsi que des politiques des états rencontrés.

La taille de ce programme linéaire est inférieure à

$$2\ell \times (n+1) \times T \leq (T+n)^{3n} \times (n+1) \times 2T \leq (T+n)^{4n}$$

(si $T \geq 2$ et $n \geq 2$), c'est-à-dire exponentielle en la taille de $\mathscr{A}$. Il est résolu en temps polynomial par rapport à sa taille [RTV97].

Proposition 5.10 (Complexité de l'accessibilité sur les ITA$_-$)**.** *Soit $\mathscr{A}$ un ITA$_-$ à n horloges et T transitions. L'algorithme d'accessibilité termine en temps $P\big((T+n)^{4n}\big)$ pour un certain polynôme P.*

L'algorithme d'accessibilité est clairement correct. Si le programme linéaire admet une solution, alors cette solution permet de construire une exécution. À partir du chemin deviné, il suffit d'attendre dans chaque état la durée donnée par la solution pour les variables $\{d_j\}_{1 \leq j \leq 2\ell}$.

Il reste à montrer que cet algorithme est bien complet, c'est-à-dire que s'il répond qu'il n'y a pas d'exécution menant à un état de contrôle donné, c'est qu'il n'en existe effectivement pas. Plus précisément, il suffit de montrer que s'il existe une exécution (de longueur quelconque) menant à l'état de contrôle, il en existe une suffisamment courte et donc devinée par l'algorithme.

Lemme 5.11 (Complétude de l'accessibilité)**.** *Soit $\mathscr{A}$ un ITA$_-$ à n horloges et T transitions. Soit (s_f, v_f, β_f) une configuration de $\mathscr{T}_\mathscr{A}$ atteignable depuis $(s_I, \mathbf{0}, \bot)$. Alors il existe un chemin dans $\mathscr{T}_\mathscr{A}$ contenant moins de $(T+n)^{3n}$ pas discrets entre $(s_I, \mathbf{0}, \bot)$ et (s_f, v_f, β'_f) où $\beta'_f \in \mathbb{B}$ est tel que $\beta_f \Rightarrow \beta'_f$.*

Démonstration. Soit ρ une exécution de $(s_I, \mathbf{0}, \bot)$ à (s_f, v_f, β_f) de longueur *minimale* $|\rho| > (T+n)^{3n}$. On construit une exécution ρ' de longueur plus petite de $(s_I, \mathbf{0}, \bot)$ à (s_f, v_f, β_f), en contradiction avec l'hypothèse de minimalité de ρ.

Puisque $|\rho| > (T+n)^{3n}$, l'un des cas du lemme 5.9 s'applique. Il y a donc une transition e de niveau k répétée aux positions[7] π et π' dans ρ. De plus l'exécution $\sigma = \rho_{[\pi,\pi'[}$ séparant ces positions ne contient que des transitions de niveau supérieur ou égal à k. On note (s, v, β) la configuration à la position π et (s, v', β') la configuration à la position π'. On se trouve dans l'un des cas suivants :

- Soit e met à jour x_k (voir figure 5.8(a)). Dans ce cas, on a $v' = v$. En effet, les horloges de niveau strictement inférieur à k n'évoluent ni par écoulement du temps (car elles sont inactives dans σ) ni par mises à jour (interdites dans les ITA$_-$). Comme seules les valeurs de ces horloges sont utilisées pour mettre à jour l'horloge x_k, on a bien $v(x_k) = v'(x_k)$. De plus, si $\beta \neq \beta'$, alors $pol(s) = L$.

  En conséquence, le chemin $\rho' = \rho_{<\pi} \cdot \rho_{\geq \pi'}$ dans $\mathscr{T}_\mathscr{A}$ est bien une exécution entre $(s_I, \mathbf{0}, \bot)$ et (s_f, v_f, β_f). Dans cette écriture de ρ', on considère que la transition menant à π dans ρ est redirigée vers π'. On a donc enlevé le chemin $\rho_{[\pi,\pi']} = \sigma e$, de longueur au moins 1. Ainsi ρ' est strictement plus court que ρ.
- Ou alors x_k n'est pas mis à jour dans σ et e est soit retardée soit paresseuse (voir figure 5.8(b)). Dans ce cas, $v' = v + d$ avec $d = Dur\left(\rho_{[\pi,\pi']}\right)$. Le chemin

  $$\rho' = \rho_{<\pi} \cdot (s, v, \beta) \xrightarrow{d} (s, v+d, \beta'') \cdot \rho_{>\pi'}$$

  est donc aussi une exécution[8] de $\mathscr{A}$. En effet, on a remplacé le chemin $\rho_{[\pi-1,\pi'[} = e\sigma$ (de longueur au moins 1) par un pas de temps[9] de durée identique.

  Remarquons que l'on n'a pas nécessairement $\beta' = \beta''$, mais que cela n'influe pas sur le franchissement de e car on a tout de même $\beta' \Rightarrow \beta''$. Dans le cas où π' est la dernière position de ρ, la configuration atteinte après ρ' peut différer de (s_f, v_f, β_f) par sa troisième composante ; ici encore, le fait que $\beta' \Rightarrow \beta''$ assure l'implication $\beta_f \Rightarrow \beta'_f$ annoncée. On a donc une exécution ρ' plus courte que ρ de $(s_I, \mathbf{0}, \bot)$ à (s_f, v_f, β'_f) avec $\beta_f \Rightarrow \beta'_f$.
- Ou x_k n'est pas mis à jour dans σ et le temps ne s'écoule pas au niveau k dans σ (voir figure 5.8(c)). Dans ce cas comme dans le premier, $v = v'$ et si $\beta \neq \beta'$, alors $pol(s) = L$. De la même manière, supprimer le chemin $\rho_{[\pi,\pi']} = \sigma e$ de ρ fournit une exécution strictement plus courte de $(s_I, \mathbf{0}, \bot)$ à (s_f, v_f, β_f). □

7. La position d'une transition est la position de sa destination.

8. On aggrège deux éventuels pas de temps consécutifs en un seul.

9. On ne peut pas garantir le même résultat si l'on supprime σe car on ne connaît pas la politique de l'état cible de e.

5.3.4 Complexité de l'accessibilité

En appliquant la procédure d'accessibilité sur l'ITA$_-$ construit à partir d'un ITA, on obtient une procédure d'accessibilité plus efficace pour les ITA que celle basée sur le graphe des classes.

Démonstration du théorème 5.7. Soit un ITA $\mathscr{A}$ avec $n \geq 2$ horloges et des transitions de taille $T \geq 2$. On construit l'ITA$_-$ $\mathscr{A}'$ de la proposition 5.8 ; on note T' le nombre de transitions de $\mathscr{A}'$. On applique la procédure d'accessibilité de la section précédente. Cette procédure termine en au plus $(T'+n)^{4n}$ (proposition 5.10). Puisque $T' \leq T^{2^{n(\log(n)+3)}}$, la taille du programme linéaire final est bornée par

$$(T'+n)^{4n} \leq \left(T^{2^{n(\log(n)+3)}}+n\right)^{4n} \leq (T+n)^{4n\times 2^{n(\log(n)+3)}} \leq (T+n)^{n\times 2^{n(\log(n)+4)}}$$

qui établit la borne énoncée. □

Les explosions (doublement-)exponentielles de la traduction de l'ITA vers l'ITA$_-$ et de la procédure d'accessibilité ne se combinent pas car l'exponentielle de l'algorithme d'accessibilité vient du nombre d'horloges et que celui-ci reste constant dans la traduction.

5.4 *Model-checking* de propriétés temporisées

La construction du graphe des classes (section 5.2), permet de réduire le comportement (non-temporisé) d'un ITA à celui d'un automate fini. Cela offre la possibilité de vérifier n'importe quelle propriété CTL* (ou toute autre logique décidable ne s'occupant que des états traversés) en utilisant les procédures classiques. Cependant, ces propriétés ne prennent pas en compte les aspects temporisés du modèle, abstraits par la construction du graphe des classes.

Les logiques temporelles *temporisées* permettent de spécifier non seulement que le comportement du système est correct, mais aussi que ce comportement doit respecter des délais précis. On peut vouloir exprimer, par exemple, les propriétés :

- « un état correct est atteint moins de 7 unités de temps après le début de l'exécution du système »,
- « un état correct est atteint moins de 3 unités de temps après une interruption »,
- « on peut atteindre un état correct tout en ayant passé plus de temps dans ce niveau que dans un niveau inférieur »,
- « le système n'atteint pas d'état d'erreur pendant les 50 premières unités de temps de son exécution ».

Dans le cas des logiques du temps linéaire, LTL [Pnu77, SC85] est donnée par la grammaire suivante :

$$\varphi := p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\varphi \mid \varphi \,\mathsf{U}\, \varphi$$

où $p \in AP$ est une proposition atomique, X est la modalité *ensuite* (*next*) et U est la modalité *jusqu'à* (*until*). La modalité $\mathsf{X}\varphi$ signifie que φ est vrai à l'état suivant, tandis que $\varphi \,\mathsf{U}\, \psi$ signifie que φ est vraie continûment jusqu'à ce que ψ le soit. On utilise les abréviations habituelles : $\mathbf{t}$ pour $\neg(p \wedge \neg p)$, $\mathsf{F}\psi$ pour $\mathbf{t}\,\mathsf{U}\,\psi$, $\mathsf{G}\psi$ pour $\neg(\mathsf{F}\neg\psi)$ et $\varphi \Rightarrow \psi$ pour $\neg(\varphi \wedge \neg\psi)$.

On peut par exemple exprimer en LTL qu'une requête est toujours suivie d'une réponse par la formule $\mathsf{G}(\textit{requête} \Rightarrow \mathsf{F}\,\textit{réponse})$.

La logique LTL a été étendue en la *Metric Temporal Logic* (MTL) [Koy90] par l'ajout de contraintes de temps, sous la forme d'un intervalle, sur la modalité U (et les raccourcis qui en sont dérivés). Cependant, dans le cas temporisé, spécifier sur l'état suivant à l'aide de X n'a pas de sens, car l'écoulement du temps de manière continue empêche de définir ce qu'est l'état suivant. On peut ainsi exprimer en MTL qu'une requête est suivie par une réponse moins de 2 unités de temps après, par la formule $\mathsf{G}(\textit{requête} \Rightarrow \mathsf{F}_{[0,2]}\,\textit{réponse})$. Le *model-checking* de MTL est cependant indécidable sur les automates temporisés.

Des fragments décidables de MTL ont donc été proposés, tels que *Metric Interval Temporal Logic* (MITL) [AFH96], qui interdit les contraintes ponctuelles (les intervalles de type $[a,a]$, par exemple dans la formule $\mathsf{G}(\textit{requête} \Rightarrow \mathsf{F}_{[2,2]}\,\textit{réponse})$). MITL a elle même été restreinte en *State Clock Logic* (SCL) [RS97], afin d'améliorer l'efficacité des procédures de vérification. Le model-checking de MITL (et donc de SCL) est décidable sur les automates temporisés.

Cependant, nous montrons que le model-checking de SCL (et donc de MITL) est indécidable sur les ITA. Cette preuve se base sur la réduction du problème de l'arrêt d'une machine à deux compteurs.

Dans le cas des logiques arborescentes, la logique CTL [QS82, EH82, CES86] est définie par

$$\varphi := p \mid \neg\varphi \mid \mathsf{E}\,\mathsf{X}\varphi \mid \mathsf{A}\,\mathsf{X}\varphi \mid \varphi \wedge \varphi \mid \mathsf{E}\,\varphi\,\mathsf{U}\,\varphi \mid \mathsf{A}\,\varphi\,\mathsf{U}\,\varphi$$

où $p \in AP$ est une proposition atomique, A signifie que tous les chemins satisfont la formule qu'il précède ($\mathsf{X}\varphi$ ou $\varphi\,\mathsf{U}\,\psi$) et E signifie qu'il existe un chemin satisfaisant la formule qu'il précède. Les abréviations de LTL sont adaptées à CTL.

La logique CTL permet par exemple de distinguer les cas où une requête est toujours suivie d'une réponse $\mathsf{AG}(\textit{requête} \Rightarrow \mathsf{AF}\,\textit{réponse})$ de celui où une requête laisse toujous la possibilité d'une réponse $\mathsf{AG}(\textit{requête} \Rightarrow \mathsf{EF}\,\textit{réponse})$.

Au moins deux extensions temporisées de CTL ont été définies. La première, définie par Alur, Courcoubetis et Dill [ACD93] ajoute, tout comme MTL, des contraintes de temps sur la modalité U. La formule $\mathsf{AG}(\textit{requête} \Rightarrow \mathsf{AF}_{\leq 2}\,\textit{réponse})$ impose une réponse en moins de deux unités de temps à toute requête.

La seconde extension, *Timed Computational Tree Logic* (TCTL) [HNSY94], plus expressive, utilise des horloges de formule. Une réponse en moins de deux unités de temps à une requête étant alors exprimée par $\mathsf{AG}(\textit{requête} \Rightarrow y.(\mathsf{AF}(\textit{réponse} \wedge y \leq 2))$, l'opérateur $y.$ signifiant que l'on prend une horloge y à valeur 0. Le model-checking de ces deux logiques est décidable pour les automates temporisés [BCM05].

Nous montrons section 5.6.1 que le problème d'accessibilité est indécidable sur le produit synchronisé d'un automate temporisé et d'un ITA. Cela empêche par exemple d'utiliser des techniques de vérification par produit synchronisé [ABL98]. Nous conjecturons donc que le model-checking de TCTL est indécidable pour les ITA.

Cependant deux fragments de TCTL pour lesquels le model-checking est décidable ont étés identifiés. Le premier, $\mathsf{TCTL}_c^{\text{int}}$, n'accepte comme horloges de formule que celles de l'automate sur lequel porte la vérification. Le second, TCTL_p, restreint l'imbrication des modalités U. Nous donnons dans les deux cas des algorithmes de model-checking.

5.4.1 Indécidabilité de SCL

On considère les extensions temporisées de LTL, et plus particulièrement le fragment SCL [RS97].

Définition 5.5 (State Clock Logic)**.** *Les formules de SCL sont définies par la grammaire suivante :*

$$\psi = p \mid \psi \wedge \psi \mid \neg\psi \mid \psi \,\mathsf{U}\, \psi \mid \psi \,\mathsf{S}\, \psi \mid \triangleright_{\bowtie a}\psi \mid \triangleleft_{\bowtie a}\psi$$

où $p \in AP$ est une proposition atomique, $\bowtie \in \{<, \leq, =, \geq, >\}$ est un opérateur de comparaison et $a \in \mathbb{Q}_{\geq 0}$ est un nombre rationnel positif.

Les formules de SCL sont interprétées sur les exécutions totales du système de transition temporisé $\mathscr{T}_{\mathscr{A}}$ généré par l'ITA $\mathscr{A}$. La sémantique est définie de manière standard pour les opérateurs booléens et la modalité U. La modalité S (*depuis* ou *since*) est la version de U pour le passé. La modalité de *prophétie* $\triangleright_{\bowtie a}\psi$ est vraie si la *prochaine* fois que ψ sera vraie se produit avec un délai de temps vérifiant la contrainte $\bowtie a$. De même, la modalité d'*histoire* $\triangleleft_{\bowtie a}\psi$ est vraie lorsque la *dernière* fois que ψ était vraie s'est produite avec un délai (passé) de temps vérifiant la contrainte $\bowtie a$. Plus précisément, pour une exécution ρ de $TExec(\mathscr{A})$ et une position π le long de ρ, on définit inductivement $(\rho,\pi) \models \varphi$ de la façon suivante :

$(\rho,\pi) \models p$ ssi $p \in prop(s_\pi)$ où s_π est l'état courant à la position π

$(\rho,\pi) \models \varphi \wedge \psi$ ssi $(\rho,\pi) \models \varphi$ et $(\rho,\pi) \models \psi$

$(\rho,\pi) \models \neg\varphi$ ssi $(\rho,\pi) \not\models \varphi$

$(\rho,\pi) \models \varphi \,\mathsf{U}\, \psi$ ssi il existe une position $\pi' \geq_\rho \pi$ telle que $(\rho,\pi') \models \psi$ et pour tout π'' t.q. $\pi \leq_\rho \pi'' <_\rho \pi'$, $(\rho,\pi'') \models \varphi \vee \psi$

$(\rho,\pi) \models \varphi \,\mathsf{S}\, \psi$ ssi il existe une position $\pi' \leq_\rho \pi$ telle que $(\rho,\pi') \models \psi$ et pour tout π'' t.q. $\pi \geq_\rho \pi'' >_\rho \pi'$, $(\rho,\pi'') \models \varphi \vee \psi$

$(\rho,\pi) \models \triangleright_{\bowtie a}\varphi$ ssi soit $(\rho,\pi) \models \varphi$ et $0 \bowtie a$ ou alors il existe une position $\pi' >_\rho \pi$ telle que $(\rho,\pi') \models \varphi$, $Dur\left(\rho_{[\pi,\pi']}\right) \bowtie a$ et pour tout π'' t.q. $\pi \leq_\rho \pi'' <_\rho \pi', (\rho,\pi'') \not\models \varphi$

$(\rho,\pi) \models \triangleleft_{\bowtie a}\varphi$ ssi soit $(\rho,\pi) \models \varphi$ et $0 \bowtie a$ ou alors il existe une position $\pi' <_\rho \pi$ telle que $(\rho,\pi') \models \varphi$, $Dur\left(\rho_{[\pi',\pi]}\right) \bowtie a$ et pour tout π'' t.q. $\pi \geq_\rho \pi'' >_\rho \pi', (\rho,\pi'') \not\models \varphi$.

On note $\mathscr{A} \models \varphi$ si pour toute exécution $\rho \in TExec(\mathscr{A})$, $(\rho,\pi_0) \models \varphi$, où π_0 est la position initiale de ρ.

Théorème 5.12 (Model-checking de SCL)**.** *Le model-checking de SCL sur les ITA est indécidable. Plus précisément, il existe une formule φ_{SCL} ne contenant que les modalités U et $\triangleleft_{=a}$, telle que le model-checking de φ_{SCL} sur les ITA à trois niveaux est indécidable.*

La preuve du théorème 5.12 repose sur le codage d'une machine à deux compteurs. Plus spécifiquement, on définit une formule φ_{SCL} de SCL et, étant donnée une machine à deux compteurs $\mathscr{M}$, on construit un ITA $\mathscr{A}_{\mathscr{M}}$ à trois horloges tel que $\mathscr{A}_{\mathscr{M}} \models \varphi_{\mathsf{SCL}}$ si et seulement si $\mathscr{M}$ ne termine pas.

Rappelons qu'une machine à deux compteurs est un ensemble fini d'instructions étiquetées, qui manipulent deux compteurs c et d à valeurs dans $\mathbb{N}$, et se termine par une instruction spéciale *Halt*. Les autres instructions ont l'une des formes suivantes, où $b \in \{c,d\}$ représente l'un des deux compteurs :

– $b := b+1$; aller en ℓ'.
– si $b > 0$ alors ($b := b-1$; aller en ℓ') sinon aller en ℓ''.

Sans perte de généralité, on peut supposer que les compteurs ont initialement la valeur 0. Le comportement de la machine est une suite (potentiellement infinie) de configurations $\langle \ell_0, 0, 0\rangle, \langle \ell_1, n_1, p_1\rangle, \ldots, \langle \ell_i, n_i, p_i\rangle, \ldots$, où n_i et p_i sont les valeurs respectives des compteurs c et d et ℓ_i est l'étiquette, après la i-ième instruction. Le problème de terminaison d'une telle machine (« atteint-on l'instruction *Halt* ? ») est indécidable [Min67].

L'idée générale du codage est que, pourvu que l'exécution satisfasse certaines contraintes (exprimables en SCL), les horloges de niveau 1 et 2 gardent les valeurs de c ou de d indifféremment, avec $x_i = \frac{1}{2^n}$ lorsque n est la valeur du compteur encodé par l'horloge de niveau i. Le niveau 3 sert de niveau de travail courant. Le passage d'information d'un niveau bas à un niveau plus haut, impossible par la syntaxe des ITA, est ici possible par le truchement de la formule.

Dans la suite de cette section, on définit :

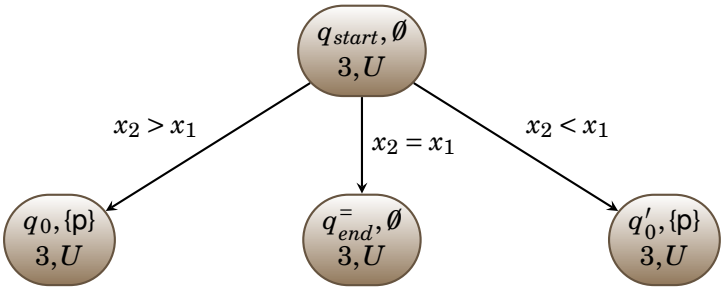
– un module d'échange $\mathscr{A}_{\leftrightarrow}$ et une formule $\varphi_{\leftrightarrow}$, tels que les valeurs contenues dans les horloges x_1 et x_2 au début d'une exécution $\rho \in TExec(\mathscr{A}_{\leftrightarrow})$ de position initiale π_0 sont échangées si et seulement si $(\rho, \pi_0) \vDash \varphi_{\leftrightarrow}$;
– un module d'incrémentation $\mathscr{A}_+$ et une formule φ_+ tels que si la valeur de x_2 est $\frac{1}{2^n}$ au début d'une exécution $\rho \in Exec(\mathscr{A}_+)$ de position initiale π_0, alors x_2 a la valeur $\frac{1}{2^{n+1}}$ à la fin de ρ si et seulement si $(\rho, \pi_0) \vDash \varphi_+$;
– un module de décrémentation $\mathscr{A}_-$ et une formule φ_- tels que si la valeur de x_2 est $\frac{1}{2^n}$ avec $n > 0$ au début d'une exécution $\rho \in Exec(\mathscr{A}_-)$ de position initiale π_0, alors x_2 a la valeur $\frac{1}{2^{n-1}}$ à la fin de ρ si et seulement si $(\rho, \pi_0) \vDash \varphi_-$.

La jonction de ces modules selon $\mathscr{M}$ produit l'ITA $\mathscr{A}_M$. La combinaison des formules (indépendamment de $\mathscr{M}$) produit une formule de SCL qui est vérifiée par les exécutions qui soit ne satisfont pas les formules de chaque module soit n'atteignent pas l'état final. Les deux constructions sont détaillées après la définition des modules.

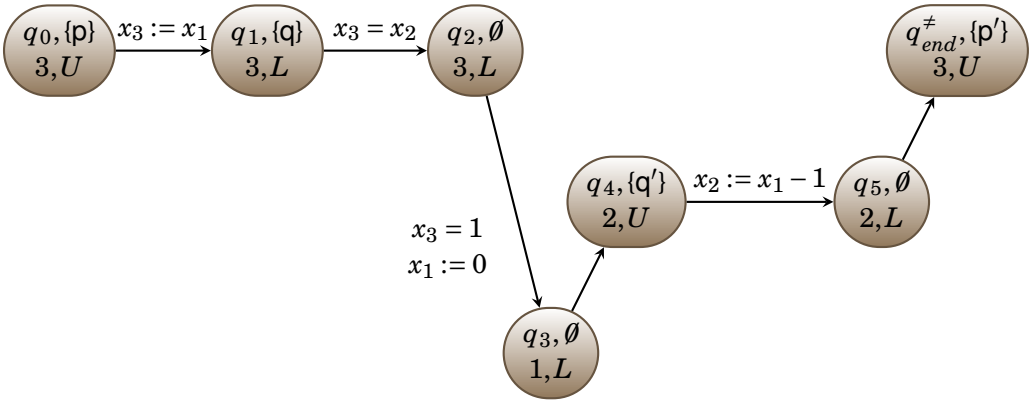
Soit $AP = \{\mathsf{p}, \mathsf{p}', \mathsf{q}, \mathsf{q}', \mathsf{h}\}$ l'ensemble de propositions atomiques. On définit les formules $Span_1 = \mathsf{q}' \Rightarrow \triangleleft_{=1}\mathsf{q}$ et $Span_2 = \mathsf{p}' \Rightarrow \triangleleft_{=2}\mathsf{p}$. La formule $Span_1$ stipule que q' est vrai e exactement 1 unité de temps après que q eut cessé d'être vrai. La formule $Span_2$ est analogue pour un écart de 2 unités de temps entre p et p'. Remarquons que les contraintes des modalités d'histoire de $Span_1$ (et $Span_2$) sont des égalités, ainsi elles ne peuvent être satisfaites que si q' (respectivement p') n'est vrai qu'à un instant ponctuel du temps.

On note x_1^0 et x_2^0 les valeurs respectives de x_1 et x_2 lors de l'entrée dans un module. Dans les automates représentés ci-après, un état porte, dans l'ordre, son nom (s), les propositions atomiques qui y sont vraies ($prop(s)$), son niveau ($\lambda(s)$) et sa politique ($pol(s)$).

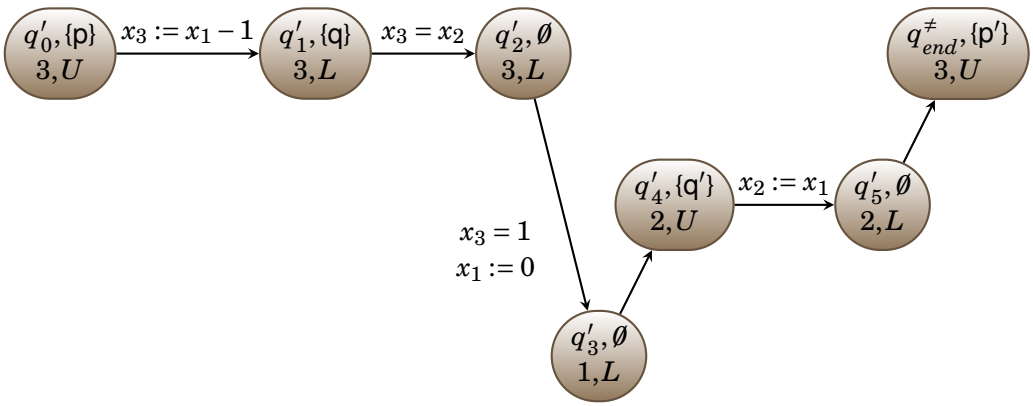
Module d'échange. Le module $\mathscr{A}_{\leftrightarrow}$ qui échange les valeurs contenues dans x_1 et x_2 est représenté figure 5.9. On peut remarquer que ce module n'échange pas les valeurs de x_1 et x_2 dans toutes ses exécutions. Nous montrons que c'est cependant le cas si l'on impose que l'état $q_{end}^{\neq}$ soit atteint exactement 2 unités de temps après que l'on eut quitté q_0 (ou q_0') et que q_4 (respectivement q_4') est atteint exactement 1



(a) Sous-module de comparaison.



(b) Sous-module d'échange (cas $x_2^0 > x_1^0$).



(c) Sous-module d'échange (cas $x_2^0 < x_1^0$).

FIGURE 5.9: Module d'échange $\mathscr{A}_{\leftrightarrow}$. Les sous-modules sont joints par leurs états identiques (q_0, q'_0 et $q^{\neq}_{end}$).

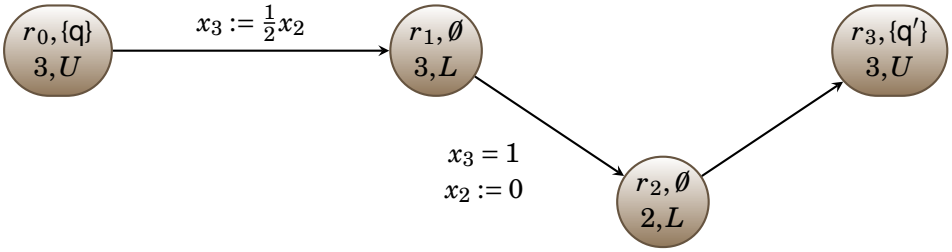


FIGURE 5.10: Module d'incrémentation $\mathscr{A}_+$.

unité de temps après que l'on eut quitté q_1 (respectivement q'_1). Ces contraintes sont exprimables en SCL par la formule $\varphi_{\leftrightarrow} = \mathsf{G}\left(Span_1 \wedge Span_2\right)$.

Remarquons que les états q_{start} et $q^{\neq}_{end}$ sont tous les deux urgents, ainsi le temps ne s'écoule pas dans ces états. De plus, si $x^0_1 = x^0_2$, alors l'exécution ne peut que passer par l'état $q^{=}_{end}$, et sort du module sans que les valeurs de x_1 et x_2 ne soit changées. On peut donc ne considérer que ce qui se produit dans les sous-modules d'échange. Les cas $x^0_2 > x^0_1$ et $x^0_2 < x^0_1$ étant analogues, seul le premier est détaillé.

Soit ρ une exécution telle que $(\rho, \pi_0) \vDash \varphi_{\leftrightarrow}$. On note w_i le temps écoulé dans l'état q_i par l'exécution ρ. Les contraintes de l'ITA imposent :

$$\begin{array}{lcll} w_0 & = & 0 & (q_0 \text{ est urgent}) \\ w_1 & = & x^0_2 - x^0_1 & (\text{mise à jour } x_3 := x_1 \text{ et garde } x_3 = x_2) \\ w_2 & = & 1 - x^0_2 & (\text{garde } x_3 = 1) \\ w_4 & = & 0 & (q_4 \text{ est urgent}) \end{array}$$

Le temps passé entre le dernier instant où q est vraie, c'est-à-dire lorsque l'on quitte q_1, et l'unique instant où q′ est vraie, c'est-à-dire lorsque l'on entre dans q_4 (même si on en sort immédiatement), est exactement le temps passé dans les états q_2 et q_3. De la même manière, le temps écoulé entre le moment où p n'est plus vraie (lorsque l'on quitte q_0) et le moment où p′ le devient (lorsque l'on entre dans $q^{\neq}_{end}$) est exactement le temps passé dans les états q_1, q_2, q_3, q_4 et q_5. Puisque $\varphi_{\leftrightarrow}$ est vérifiée par ρ, on a :

$$\begin{array}{rcll} w_2 + w_3 & = & 1 & (\mathsf{q}' \Rightarrow \lhd_{=1}\mathsf{q}) \\ w_1 + w_2 + w_3 + w_4 + w_5 & = & 2 & (\mathsf{p}' \Rightarrow \lhd_{=2}\mathsf{p}) \end{array}$$

On obtient ainsi $w_3 = x^0_2$ et $w_5 = 1 - w_1 = x^0_1 - \left(x^0_2 - 1\right)$. Lorsque l'on entre dans q_3, l'horloge x_1 vaut 0, donc lorsque l'on quitte cet état, x_1 vaut x^0_2 ; sa valeur ne change plus par la suite, ni par écoulement du temps (les états sont de niveau supérieur) ni par mise à jour. De même, lorsque l'on entre dans q_5, x_2 vaut $x_1 - 1 = x^0_2 - 1$, donc x_2 vaut x^0_1 lorsque l'on atteint $q^{\neq}_{end}$.

Notons que ce module échange les valeurs de x_1 et x_2 même si celles ci ne codent pas de valeur de compteurs, tant que leurs valeurs sont comprises entre 0 et 1. Remarquons aussi qu'au cours de l'exécution de ce module, des horloges peuvent prendre des valeurs négatives. Ceci n'est pas interdit par la sémantique, contrairement au cas des automates temporisés.

Module d'incrémentation. Le cas du module d'incrémentation repose sur une idée similaire. Le module $\mathscr{A}_+$ est représenté figure 5.10. On force le temps passé en r_1 et r_2 à valoir 1 à l'aide de la contrainte $\varphi_+ = \mathsf{G}Span_1$. Soit une exécution $\rho \in Exec(\mathscr{A}_+)$

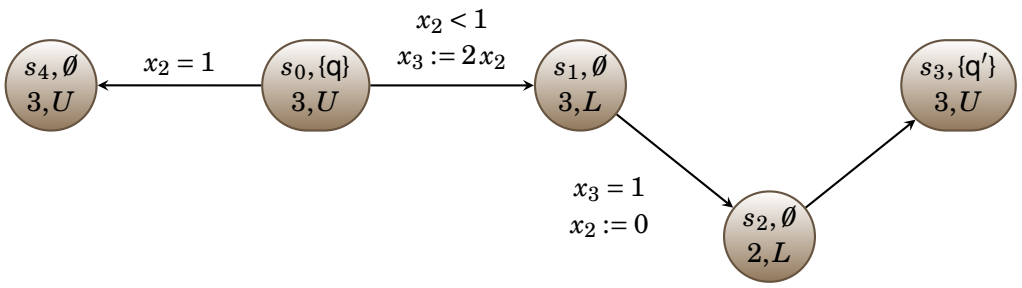


FIGURE 5.11: Module de décrémentation $\mathscr{A}_-$. L'exécution démarre en l'état s_0.

telle que $(\rho, \pi_0) \vDash \varphi_+$. Les gardes et mises à jour de $\mathscr{A}_+$ garantissent que le temps passé dans r_1 vaut $1 - \frac{1}{2}x_2^0$. Le temps passé en r_2, qui est aussi la valeur de x_2 en entrant dans r_3, vaut donc $\frac{1}{2}x_2^0$. Si $x_2^0 = \frac{1}{2^n}$, codant la valeur n pour un compteur, à la fin de l'exécution dans $\mathscr{A}_+$, $x_2 = \frac{1}{2^{n+1}}$, ce qui code donc la valeur $n+1$ pour le compteur.

Module de décrémentation. La décrémentation, pour laquelle le module $\mathscr{A}_-$ est représenté figure 5.11, fonctionne de manière analogue. Il faut cependant commencer par comparer la valeur du compteur à 0, c'est-à-dire comparer x_2^0 à 1. Lorsque la formule $\varphi_- = \mathsf{G}Span_1(= \varphi_+)$ est vérifiée pour une exécution $\rho \in Exec(\mathscr{A}_-)$, le temps passé en s_1 et s_2 vaut 1. Le temps passé en s_1 est, selon les mises à jour et gardes de $\mathscr{A}_-$, $1 - 2x_2^0$. Le temps passé en s_2, qui est aussi la valeur finale de x_2, vaut donc $2x_2^0$. On suppose que $x_2^0 = \frac{1}{2^n}$ code la valeur n pour un compteur. Alors si $n = 0$ l'exécution se termine dans s_4 sans changer la valeur des horloges. Et si $n > 0$, l'exécution se termine dans s_3 avec $x_2 = \frac{1}{2^{n-1}}$, c'est-à-dire codant la valeur $n-1$ pour le compteur.

Jonction des modules. Jusqu'ici, les modules ont été définis sans lien avec $\mathscr{M}$. Seule leur jonction dépend des instructions de $\mathscr{M}$. On note ici (q, ℓ) l'état q dans une instance d'un module correspondant à l'instruction ℓ.

Dans le cas d'une incrémentation de c « $\ell : c := c + 1$; aller en ℓ' » le module correspondant est simplement un module d'incrémentation $\mathscr{A}_+$, relié par une transition (sans garde ni mise à jour) de r_3 vers le premier état du module correspondant à ℓ' ((q_{start}, ℓ'), (r_0, ℓ') ou (s_0, ℓ')).

Pour une incrémentation de d « $\ell : d := d + 1$; aller en ℓ' », on entoure une instance du module d'incrémentation $\mathscr{A}_+$ de deux instances du module d'échange $\mathscr{A}_{\leftrightarrow}^{in}$ et $\mathscr{A}_{\leftrightarrow}^{out}$. En effet, le module d'incrémentation modifie x_2, qui originellement contient la valeur de c et non celle de d. Le deuxième échange permet de garder cet invariant. On note (q, ℓ, in) et (q, ℓ, out) les états respectifs des deux instances $\mathscr{A}_{\leftrightarrow}^{in}$ et $\mathscr{A}_{\leftrightarrow}^{out}$. Le premier état du module pour ℓ est l'état (q_{start}, ℓ, in), c'est-à-dire le premier état de $\mathscr{A}_{\leftrightarrow}^{in}$. Ces modules sont joints par des transitions sans garde ni mise à jour :

- de $\left(q_{end}^{\neq}, \ell, in\right)$ et de $\left(q_{end}^{=}, \ell, in\right)$ vers (r_0, ℓ) (lien entre $\mathscr{A}_{\leftrightarrow}^{in}$ et $\mathscr{A}_+$),
- de (r_3, ℓ) vers (q_{start}, ℓ, out) (lien entre $\mathscr{A}_+$ et $\mathscr{A}_{\leftrightarrow}^{out}$),
- de $\left(q_{end}^{\neq}, \ell, out\right)$ et de $\left(q_{end}^{=}, \ell, out\right)$ vers le premier état du module correspondant à ℓ'.

Le cas des décrémentations est sensiblement le même que pour les incrémentations. On utilise des instances du module $\mathscr{A}_-$, en reliant l'état (s_4, ℓ) au premier état

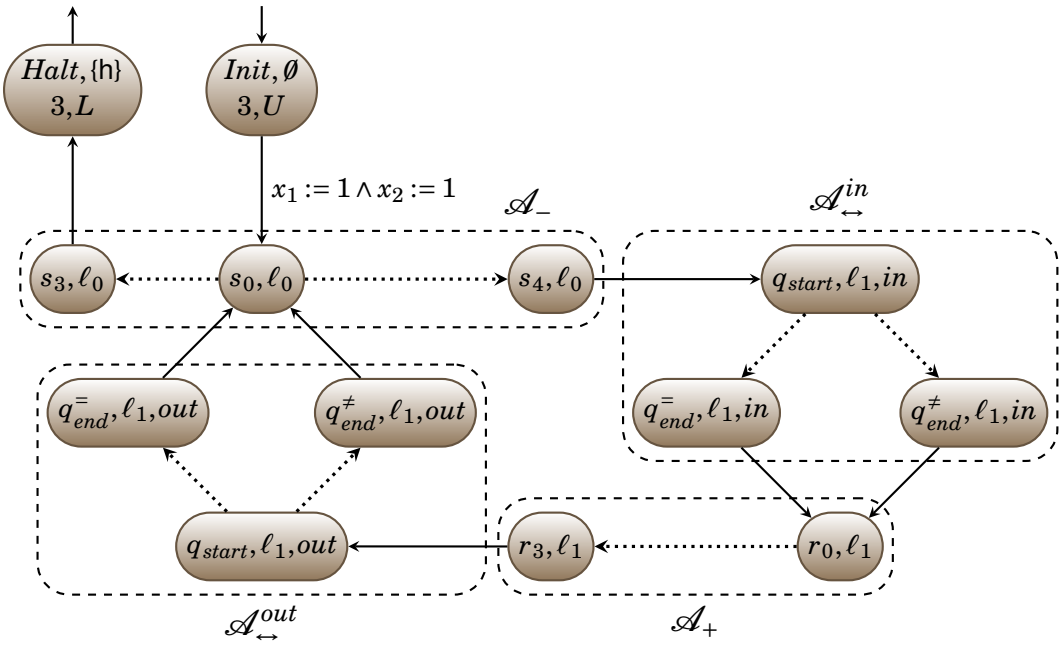


FIGURE 5.12: Jonction des modules dans $\mathscr{A}_{17}$ pour la machine $\mathscr{M}_{17}$. Les propositions atomiques, les niveaux des états qui apparaissent (tous 3) ainsi que leur politique (toutes urgentes) sont omis.

du module correspondant à ℓ''. Dans le cas de la décrémentation de d il faut intercaler une troisième instance d'un module d'échange entre (s_4, ℓ) et le premier état du module de ℓ''.

L'instruction *Halt* correspond à un seul état (acceptant) h tel que $prop(h) = \{\mathsf{h}\}$, sans transition sortante. L'état initial de $\mathscr{A}_{\mathscr{M}}$ est un état *Init* de niveau 3, de politique urgente et ne vérifiant aucune proposition atomique. L'état *Init* est relié au premier état du module correspondant à l'instruction initiale ℓ_0 par une transition sans garde et de mise à jour $x_1 := 1 \wedge x_2 := 1$. Ces mises à jour correspondent à une initialisation à 0 des compteurs c et d.

Par exemple, si l'on considère la machine $\mathscr{M}_{17}$ définie par les instructions suivantes :

ℓ_0 : si $c > 0$ alors ($c := c - 1$; aller en *Halt*) sinon aller en ℓ_1.
ℓ_1 : $d := d + 1$; aller en ℓ_0.

La structure globale de l'ITA correspondant $\mathscr{A}_{17}$ est représenté figure 5.12.

Combinaison des formules. Remarquons que dans chaque module, pour que les contraintes soient vérifiées, il suffit que les deux formules $Span_1$ et $Span_2$ soient toujours vraies. On définit la formule φ_{SCL} de la façon suivante :

$$\varphi_{\mathsf{SCL}} \;=\; \big(\mathsf{F}\,(\neg Span_1 \vee \neg Span_2)\big) \vee \mathsf{G}\,(\neg \mathsf{h}).$$

Cette formule exprime le fait qu'une contrainte $Span_i$ est violée à un certain moment ou que la proposition atomique h n'est jamais vérifiée.

Soit une exécution $\rho \in Exec(\mathscr{A}_{\mathscr{M}})$ qui satisfait φ_{SCL}. Ou bien ρ n'atteint pas l'état h. Ou alors lors de son passage dans un module, ρ ne vérifie pas les contraintes

associées (par exemple φ_+ dans une instance du module $\mathscr{A}_+$). Dans ce cas, ρ ne simule pas une exécution de $\mathscr{M}$.

Supposons que $\mathscr{M}$ termine. On peut transformer cette exécution de $\mathscr{M}$ en une exécution $\rho \in Exec(\mathscr{A}_\mathscr{M})$ qui vérifie toujours les contraintes $Span_i$ et qui atteint l'état h. Ainsi $(\rho,\pi_0) \not\models \varphi_{\mathsf{SCL}}$ et $\mathscr{A}_\mathscr{M} \not\models \varphi_{\mathsf{SCL}}$.

Réciproquement, supposons que $\mathscr{A}_\mathscr{M} \not\models \varphi_{\mathsf{SCL}}$. Soit ρ telle que $(\rho,\pi_0) \not\models \varphi_{\mathsf{SCL}}$. Alors ρ vérifie toutes les contraintes $Span_i$ et atteint l'état h. C'est donc une simulation de l'exécution de $\mathscr{M}$ qui atteint l'instruction *Halt*, et $\mathscr{M}$ termine.

Donc la machine $\mathscr{M}$ ne termine pas si et seulement si

$$\mathscr{A}_\mathscr{M} \models \mathsf{F}\left(\left(\neg \mathsf{q}' \wedge \neg \lhd_{=1} \mathsf{q}\right) \vee \left(\neg \mathsf{p}' \wedge \neg \lhd_{=2} \mathsf{p}\right)\right) \vee \mathsf{G}\,\neg \mathsf{h}.$$

On peut remarquer que cette formule ne contient pas d'imbrication d'opérateurs d'histoire ou de prophétie ($\lhd_{\bowtie a}$ et $\rhd_{\bowtie a}$). Ainsi le model-checking d'une interprétation de SCL avec une sémantique discrète, où les sous-formules ne sont évaluées qu'à l'instant où l'on entre dans un état, est aussi indécidable.

5.4.2 Propriétés sur horloges internes

On considère maintenant une extension de CTL avec des horloges du modèle. Le fragment de TCTL ainsi défini est nommé $\mathsf{TCTL}_c^{\text{int}}$. Une telle logique permet de raisonner sur les temps écoulés dans les divers niveaux de l'ITA, ce qui est primordial par exemple dans le cas de systèmes d'exploitation temps-réel. Par exemple « un état correct est atteint moins de 3 unités de temps après une interruption », peut s'exprimer par la formule $\mathsf{A}(x_2 \leq 3)\,\mathsf{U}\,\mathit{Correct}$ (en supposant que x_2 n'est pas mise à jour). La formule $\mathsf{E}\,\mathsf{F}\,(\mathit{Correct} \wedge (x_2 > x_1))$ exprime « on peut atteindre un état correct tout en ayant passé plus de temps au niveau courant que dans un niveau inférieur » dans le cas où l'on s'intéresse au niveau 2. Ces deux formules sont dans $\mathsf{TCTL}_c^{\text{int}}$, dont nous donnons à présent la syntaxe et la sémantique.

Définition 5.6 (TCTL à horloges internes)**.** *Les formules de $TCTL_c^{int}$ sont définies par la grammaire suivante :*

$$\psi ::= p \mid \psi \wedge \psi \mid \neg\psi \mid \sum_{i\geq 1} a_i \cdot x_i + b \bowtie 0 \mid A\psi\, U\, \psi \mid E\psi\, U\, \psi$$

où $p \in AP$ est une proposition atomique, x_i sont des horloges du modèle, $(a_i)_{i\geq 1}$ est une famille à support fini de nombres rationnels, $b \in \mathbb{Q}$ et $\bowtie \in \{<,\leq,=,\geq,>\}$.

On utilise comme précédemment les abréviations F et G ainsi que les opérateurs booléens.

Les formules de $\mathsf{TCTL}_c^{\text{int}}$ sont interprétées sur les configurations (s,v,β) d'un ITA. Toutefois, le booléen β n'est pas utilisé. Il serait possible d'étendre $\mathsf{TCTL}_c^{\text{int}}$ de manière à tirer profit de cette information, par exemple pour exprimer qu'une exécution passe un temps non nul dans un état correct ou, au contraire, que les états d'erreurs ne sont visités que pendant des durées nulles.

Soit $\mathscr{A} = \langle S, Lab, X, \lambda, pol, \Delta, s_I, F, AP, prop\rangle$ un ITA. La sémantique de $\mathsf{TCTL}_c^{\text{int}}$ est définie sur $\mathscr{T}_\mathscr{A}$ de la façon suivante. Soit $q = (s,v,\beta)$ une configuration de $\mathscr{A}$:

$$\begin{array}{lll} q \models p & \text{ssi} & p \in prop(s) \\ q \models \sum_{i\geq 1} a_i \cdot x_i + b \bowtie 0 & \text{ssi} & v \models \sum_{i\geq 1} a_i \cdot x_i + b \bowtie 0 \end{array}$$

et inductivement :

$$\begin{array}{lll} q \models \varphi \wedge \psi & \text{ssi} & q \models \varphi \text{ et } q \models \psi \\ q \models \neg\varphi & \text{ssi} & q \not\models \varphi \\ q \models \mathsf{A}\varphi \,\mathsf{U}\, \psi & \text{ssi} & \text{pour tout } \rho \in \mathit{TExec}_q(\mathscr{A}),\ \rho \models \varphi \,\mathsf{U}\, \psi \\ q \models \mathsf{E}\varphi \,\mathsf{U}\, \psi & \text{ssi} & \text{il existe } \rho \in \mathit{TExec}_q(\mathscr{A}) \text{ t.q. } \rho \models \varphi \,\mathsf{U}\, \psi \\ \text{avec } \rho \models \varphi \,\mathsf{U}\, \psi & \text{ssi} & \text{il existe une position } \pi \in \rho \text{ t.q. } q_\pi \models \psi \\ & & \text{et } \forall \pi' <_\rho \pi,\ q_{\pi'} \models \varphi \vee \psi. \end{array}$$

où $\mathit{TExec}_q(\mathscr{A})$ est l'ensemble des exécutions (totales) de $\mathscr{A}$ depuis la configuration q et q_π désigne la configuration à la position π. L'automate $\mathscr{A}$ satisfait la formule φ, noté aussi $\mathscr{A} \models \varphi$ si la configuration initiale de $\mathscr{A}$ la vérifie : $(s_I, \mathbf{0}, \bot) \models \varphi$.

Remarquons que la sémantique de $\mathsf{TCTL}_c^{\text{int}}$ repose sur les valuations. Ainsi la procédure de model-checking ne peut pas s'appuyer sur la traduction d'un ITA en ITA$_-$ (section 5.3.2).

Procédure de model-checking sur $\mathsf{TCTL}_c^{\text{int}}$

Le model-checking de $\mathsf{TCTL}_c^{\text{int}}$ repose sur une adaptation du graphe des classes de la section 5.2. Plus précisément, il faut intégrer au graphe des classes les informations pertinentes pour la formule. Le problème se réduit donc à une procédure de model-checking CTL sur ce graphe modifié.

Théorème 5.13 (Model-checking de $\mathsf{TCTL}_c^{\text{int}}$)**.** *Le model-checking de $\mathit{TCTL}_c^{\mathit{int}}$ sur les ITA est décidable en 2-EXPSPACE et en PSPACE lorsque le nombre d'horloges est fixé.*

Démonstration. Soit φ une formule de $\mathsf{TCTL}_c^{\text{int}}$ et $\mathscr{A}$ un ITA à n horloges et dont les transitions sont de taille T. On construit un graphe fini $\mathscr{G}'_{\mathscr{A}}$ de la même manière que dans la preuve du théorème 5.1, en modifiant cependant la manière dont les ensembles d'expressions sont construits. En effet, on construit des ensembles $E'_1, \ldots, E'_n$ qui ajoutent à $E_1, \ldots, E_n$ les expressions apparaissant dans la formule φ. Le graphe $\mathscr{G}'_{\mathscr{A}}$ est donc paramétré aussi par la formule φ. Cette dépendance est cependant implicite pour des raisons de lisibilité.

Chaque E'_k est initialisé à $\{0, x_k\}$. On procède ensuite de haut en bas, c'est-à-dire pour k allant de n à 1.

(1) Tout d'abord, pour chaque expression $\alpha x_k + \sum_{i=1}^{k-1} a_i x_i + b$ (avec $\alpha \in \{0,1\}$) apparaissant dans la garde d'une transition dont l'état source est de niveau k, on ajoute $-\sum_{i=1}^{k-1} a_i x_i - b$ à E'_k.

(2) Ensuite, afin de prendre en compte les contraintes de φ, on ajoute l'étape suivante. Soit C une expression apparaissant dans une contrainte $C \bowtie 0$ dans φ. La contrainte sur C peut être évaluée à tous niveaux ; ainsi on ne garde dans E'_k que la partie de C qui est pertinente au niveau courant.

La projection de C au niveau k est l'expression

$$\mathit{Proj}(C,k) = C\left[\left(\bigwedge_{i=1}^{k} x_i := x_i\right) \wedge \left(\bigwedge_{i>k} x_i := 0\right)\right]$$

qui retire de C toutes les variables correspondant à des horloges de niveau $> k$. À partir de $\mathit{Norm}(\mathit{Proj}(C,k),k)$ écrite sous la forme $\alpha x_k + \sum_{i=1}^{k-1} a_i x_i + b$ (avec $\alpha \in \{0,1\}$), on ajoute l'expression $-\sum_{i=1}^{k-1} a_i x_i - b$ à E'_k.

(3) Ensuite, les opérations suivantes sont itérées jusqu'à ce qu'aucune nouvelle expression ne soit ajoutée aux $\{E'_i\}_{1\leq i\leq k}$.

(3.1) Soit $s \xrightarrow{g,a,u} s' \in \Delta$ avec $\lambda(s) \geq k$ et $\lambda(s') \geq k$. Pour $C \in E'_k$, on ajoute l'expression $C[u]$ à E'_k.

(3.2) Soit $s \xrightarrow{g,a,u} s' \in \Delta$ avec $\lambda(s) < k$ et $\lambda(s') \geq k$. Soient C et C' deux expression distinctes de E'_k. Soit $C'' = Norm\big(C[u] - C'[u], \lambda(s)\big)$. L'expression C'' est réécrite $\alpha x_{\lambda(s)} + \sum_{i=1}^{\lambda(s)-1} a_i x_i + b$ (avec $\alpha \in \{0,1\}$). On ajoute alors l'expression $-\sum_{i=1}^{\lambda(s)-1} a_i x_i - b$ à $E'_{\lambda(s)}$.

La preuve de terminaison est la même qu'auparavant, en remplaçant T par $T' = T + |\varphi|$. On obtient de la même manière $|E'_k| \leq (T'+2)^{2^{n(n-k+1)}+1}$. Remarquons que pour tout k, $E_k \subseteq E'_k$, c'est-à-dire que l'on n'a fait qu'ajouter des expressions.

Le graphe des classes $\mathscr{G}'_{\mathscr{A}}$ est construit identiquement à $\mathscr{G}_{\mathscr{A}}$, simplement en considérant des préordres totaux sur les nouveaux ensembles d'expressions $E'_1, \ldots, E'_n$. Puisque pour tout k, $E_k \subseteq E'_k$, chaque préordre sur E'_k peut être projeté sur un préordre sur E_k. Ainsi le graphe des classes $\mathscr{G}'_{\mathscr{A}}$, construit à partir de $E'_1, \ldots, E'_n$, est un *raffinement* [10] de $\mathscr{G}_{\mathscr{A}}$, construit à partir de $E_1, \ldots, E_n$. En particulier, $\mathscr{G}'_{\mathscr{A}}$ est aussi une abstraction de $\mathscr{T}_{\mathscr{A}}$ vis-à-vis du temps. Similairement à $\mathscr{G}_{\mathscr{A}}$, la taille de $\mathscr{G}'_{\mathscr{A}}$ est bornée par

$$|S| \times 2^{n \times (T'+2)^{2^{n^2+1}+2}}.$$

Soit C une expression apparaissant dans φ. Remarquons que, vu la sémantique de $\mathsf{TCTL}_c^{\text{int}}$, $(s,v,\beta) \vDash C \bowtie 0$ si et seulement si $(s,v,\beta) \vDash Proj(C, \lambda(s)) \bowtie 0$ ou encore si et seulement si $(s,v,\beta) \vDash Norm(Proj(C, \lambda(s), \lambda(s))) \bowtie 0$. Notons $k = \lambda(s)$. On écrit $Norm(Proj(C,k),k) = \alpha x_k + \sum_{i=1}^{k-1} a_i x_i + b$ (avec $\alpha \in \{0,1\}$). On a $v \vDash Proj(C,k) \bowtie 0$ si et seulement si :

- soit $\alpha = 0$ et $0 \bowtie -\sum_{i=1}^{k-1} a_i x_i - b$;
- ou alors $\alpha = 1$ et $x_k \bowtie -\sum_{i=1}^{k-1} a_i x_i - b$.

Pour $R = \big(s, \{\preceq_k\}_{1\leq k\leq \lambda(s)}\big)$ une classe de $\mathscr{G}'_{\mathscr{A}}$ et pour tous $v, v' \in [\![R]\!]$, l'ordre entre les valuations de deux expressions de E_k sont les mêmes pour v et v'. Par construction, $-\sum_{i=1}^{k-1} a_i x_i - b \in E'_k$; c'est aussi le cas de 0 et de x_k. Donc $v \vDash Proj(C,k) \bowtie 0$ si et seulement si $v' \vDash Proj(C,k) \bowtie 0$. Ainsi la satisfaction des contraintes présentes dans φ est préservée au sein de chaque classe. On note donc $R \vDash C \bowtie 0$ lorsque pour tout $v \in [\![R]\!]$, $v \vDash C \bowtie 0$.

Pour chaque contrainte C de φ, on définit une nouvelle proposition atomique p_C. La fonction *prop* est étendue de la façon suivante sur les classes de $\mathscr{G}'_{\mathscr{A}}$. Pour chaque classe $R = \big(s, \{\preceq_k\}_{1\leq k\leq \lambda(s)}\big)$: $prop(R) = prop(s) \uplus \{\mathsf{p}_C \mid R \vDash C\}$. On définit φ' comme la formule φ où chaque contrainte C est remplacée par p_C.

Il est clair que $\mathscr{G}'_{\mathscr{A}} \vDash \varphi'$ si et seulement si $\mathscr{A} \vDash \varphi$. Or $\mathscr{G}'_{\mathscr{A}}$ est un graphe fini et φ' est une formule de CTL. Une procédure de model-checking CTL sur $\mathscr{G}'_{\mathscr{A}}$ donne donc le résultat du model-checking pour TCTL.

La complexité de cette procédure peut être prise en espace logarithmique par rapport à la taille du modèle et polynomial par rapport à la taille de la formule [KVW00]. On obtient donc une complexité en espace en

$$O\left(|\varphi| \times \log|S| \times n \times (T + |\varphi| + 2)^{2^{n^2+1}+2}\right),$$

10. Bien que le terme de *raffinage* soit plus adapté à cette construction, c'est l'anglicisme *raffinement* qui est passé dans le langage courant.

c'est-à-dire en 2-EXPSPACE et en PSPACE lorsque n est fixé. □

Remarquons que si l'on veut prendre en compte la valeur du booléen β dans les formules, la construction du graphe $\mathscr{G}'_{\mathscr{A}}$ est aussi modifiée au moment de la prise en compte des politiques. En effet, même les classes de politique paresseuse doivent alors être séparées en deux, selon si du temps s'est écoulé ou non depuis le dernier franchissement d'une transition discrète.

Exemple de model-checking de $\mathbf{TCTL}_c^{\mathbf{int}}$

Appliquons cette procédure dans le cas de l'automate $\mathscr{A}_{15}$ (reproduit figure 5.13) et de la formule $\varphi_{15} = \mathsf{EF}(q_1 \wedge (x_2 > x_1))$, où q_1 est, par abus de notation, la proposition atomique vraie uniquement dans l'état q_1.

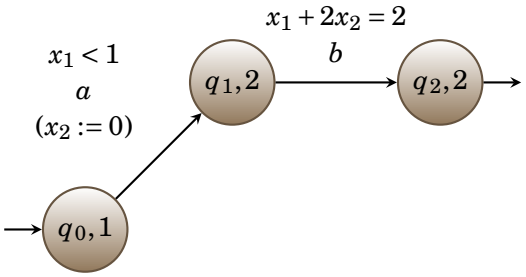


FIGURE 5.13: L'ITA $\mathscr{A}_{15}$ à deux niveaux d'interruption.

Les ensembles d'expressions sont initialement $E'_1 = \{0, x_1\}$ et $E'_2 = \{0, x_2\}$. Pour le niveau 2, l'étape (1) ajoute l'expression $-\frac{1}{2}x_1 + 1$, issue de la normalisation de la contrainte $x_1 + 2x_2 = 2$, garde de la transition étiquetée par b. L'étape (2) ajoute l'expression x_1 à E'_2, car la comparaison $x_2 - x_1 > 0$ apparaît dans φ_{15} (sous la forme $x_2 > x_1$). L'étape (3.1) ne s'applique pas ici car la seule transition de niveau 2 n'a pas de mise à jour (non triviale).

L'étape (3.2) s'applique avec la transition étiquetée par a et la mise à jour $x_2 := 0$. Il faut donc ajouter à E'_1 toutes les différences (normalisées) des expressions de E'_2 après application de la mise à zéro de x_2. Cela donne donc l'expression $-\frac{1}{2}x_1 + 1 - 0$, qui se normalise en $x_1 - 2$, et donc donne l'expression 2 à ajouter à E'_1. On obtient aussi l'expression $x_1 - \left(-\frac{1}{2}x_1 + 1\right)$ qui se normalise en $x_1 - \frac{2}{3}$, et donc donne l'expression $\frac{2}{3}$ à ajouter à E'_1.

Au niveau 1, seule l'étape (1) ajoute des expressions à E_1, à savoir l'expression 1. En effet, la nouvelle étape (2) ne donne ici que l'expression x_1 qui est déjà dans E'_1.

On obtient donc les ensembles d'expressions

$$E'_1 = \left\{x_1, 0, 1, \frac{2}{3}, 2\right\} \qquad \text{et} \qquad E'_2 = \left\{x_2, 0, -\frac{1}{2} + 1, x_1\right\}.$$

Par rapport aux ensembles E_1 et E_2 calculés section 5.2.1, on a donc ajouté les expressions x_1 au niveau 2 et $\frac{2}{3}$ au niveau 1.

Le graphe des classes $\mathscr{G}'_{\mathscr{A}_{15}}$ pour la formule φ_{15} est représenté figure 5.14. Certains préordres utilisés dans cette figure sont abrégés (voir table 5.3). De plus, au niveau 2, x_1 est remplacé par sa valeur (lorsqu'elle est connue) ; cela n'influe pas dans ce cas sur le graphe des classes car x_1 n'est pas mise à jour au niveau 2. Les états en orange sont

$$Z_0^1 = \left(0 = x_1 < \frac{2}{3} < 1 < 2\right) \qquad Z_1^1 = \left(0 < x_1 < \frac{2}{3} < 1 < 2\right) \qquad Z_2^1 = \left(0 < x_1 = \frac{2}{3} < 1 < 2\right)$$

$$Z_3^1 = \left(0 < \frac{2}{3} < x_1 < 1 < 2\right) \qquad Z_4^1 = \left(0 < \frac{2}{3} < x_1 = 1 < 2\right) \qquad Z_5^1 = \left(0 < \frac{2}{3} < 1 < x_1 < 2\right)$$

$$Z_6^1 = \left(0 < \frac{2}{3} < 1 < x_1 = 2\right) \qquad Z_7^1 = \left(0 < \frac{2}{3} < 1 < 2 < x_1\right)$$

$$Z_0^2 = \left(0 = x_2 < x_1 < -\frac{1}{2}x_1 + 1\right) \qquad Z_1^2 = \left(0 < x_2 < x_1 < -\frac{1}{2}x_1 + 1\right)$$

$$Z_2^2 = \left(0 < x_1 = x_2 < -\frac{1}{2}x_1 + 1\right) \qquad Z_3^2 = \left(0 < x_1 < x_2 < -\frac{1}{2}x_1 + 1\right)$$

$$Z_4^2 = \left(0 < x_1 < -\frac{1}{2}x_1 + 1 = x_2\right) \qquad Z_5^2 = \left(0 < x_1 < -\frac{1}{2}x_1 + 1 < x_2\right)$$

$$Z_6^2 = \left(0 = x_2 < -\frac{1}{2}x_1 + 1 < x_1\right) \qquad Z_7^2 = \left(0 < x_2 < -\frac{1}{2}x_1 + 1 < x_1\right)$$

$$Z_8^2 = \left(0 < -\frac{1}{2}x_1 + 1 = x_2 < x_1\right) \qquad Z_9^2 = \left(0 < -\frac{1}{2}x_1 + 1 < x_2 < x_1\right)$$

$$Z_{10}^2 = \left(0 < -\frac{1}{2}x_1 + 1 < x_1 = x_2\right) \qquad Z_{11}^2 = \left(0 < -\frac{1}{2}x_1 + 1 < x_1 < x_2\right)$$

TABLE 5.3: Préordres utilisés dans $\mathscr{G}'_{\mathscr{A}_{15}}$ (figure 5.14) pour le model-checking de φ_{15}.

ceux pour lesquels la contrainte $x_2 > x_1$ est vérifiée. Parmi ceux-là, les états entourés sont ceux qui correspondent à l'état q_1. Ainsi la procédure de model-checking montre que l'un de ces états orange et entouré est accessible, ce qui montre que $\mathscr{A}_{15} \models \varphi_{15}$.

5.4.3 Propriétés sans imbrication

On peut remarquer que dans le cas de $\mathsf{TCTL}_c^{\text{int}}$, il est impossible de raisonner sur l'écoulement du temps dans le système indépendamment des horloges internes, et donc des niveaux. Ceci empêche par exemple de raisonner globalement sur le temps écoulé entre deux actions d'un niveau donné, si la tâche correspondante est interrompue. Outre les interruptions, les valeurs d'horloges peuvent être faussées par des mises à jour.

Dans le but de vérifier de telles propriétés globales, on définit le fragment TCTL_p. Ce fragment est assez expressif pour exprimer des contraintes d'urgence et de délai minimal sur une exécution prise globalement. Les formules de TCTL_p contiennent des modalités U paramétrées par des demi-droites (un cas particulier des intervalles de TCTL). Il est possible dans TCTL_p d'exprimer la propriété « un état correct est atteint moins de 7 unités de temps après le début de l'exécution du système », par la

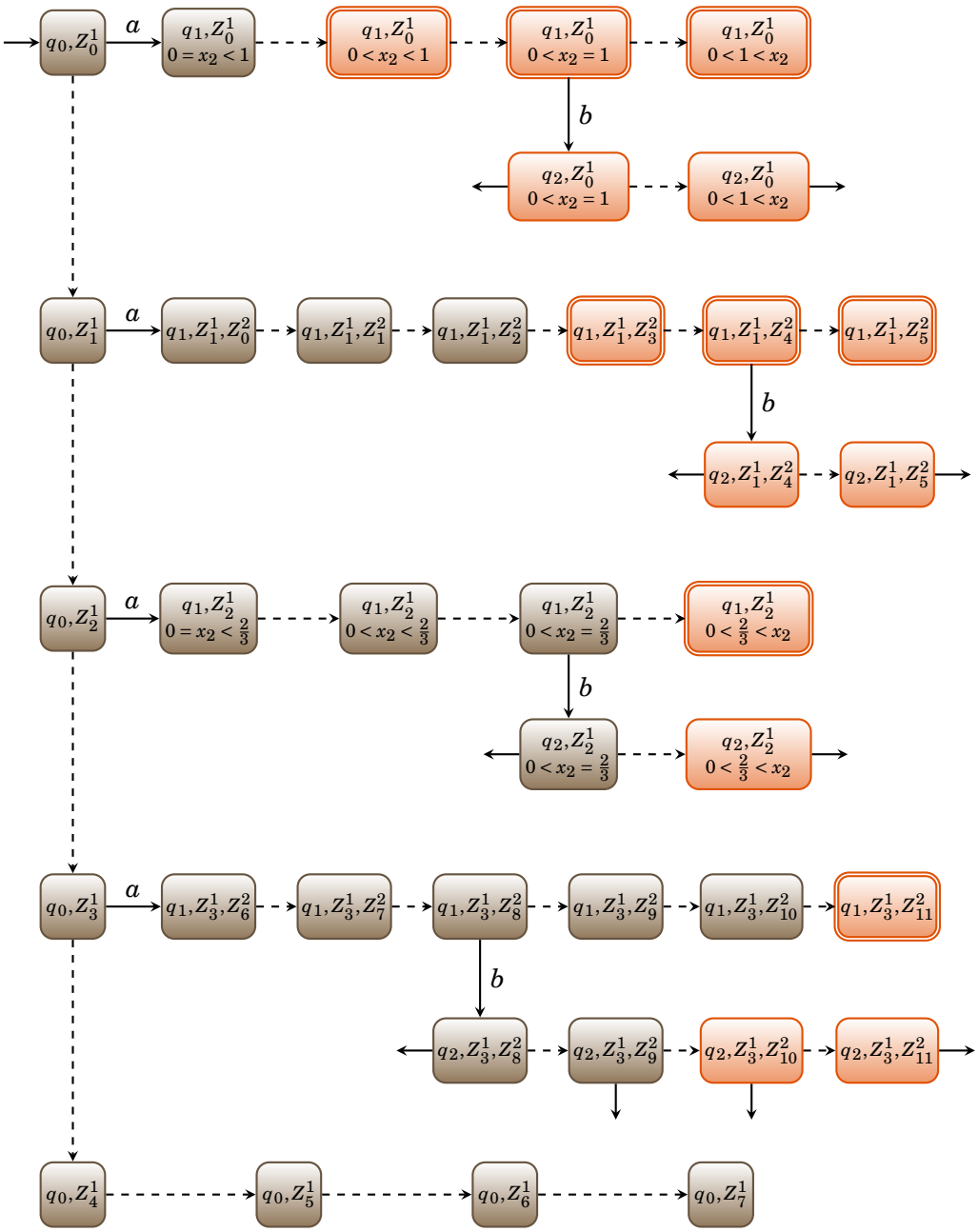


FIGURE 5.14: Graphe des classes $\mathscr{G}'_{\mathscr{A}_{15}}$ pour $\mathscr{A}_{15}$ lors du model-checking de φ_{15}.

formule $\mathsf{AF}_{\leq 7}\mathit{Correct}$. La formule $\mathsf{A}(\neg \mathit{Erreur})\,\mathsf{U}_{\geq 50}\,\mathbf{t}$ exprime, quant à elle, que « le système n'atteint pas d'état d'erreur pendant les 50 premières unités de temps de son exécution ».

Définition 5.7 (TCTL sans imbrication)**.** *Les formules de $TCTL_p$ sont définies par la grammaire suivante :*

$$\varphi_p := p \mid \varphi_p \wedge \varphi_p \mid \neg\varphi_p \qquad \textit{et} \qquad \psi := \psi \wedge \psi \mid \neg\psi \mid \varphi_p \mid \mathsf{A}\varphi_p\,\mathsf{U}_{\bowtie a}\,\varphi_p \mid \mathsf{E}\varphi_p\,\mathsf{U}_{\bowtie a}\,\varphi_p$$

où $p \in AP$ est une proposition atomique, $\bowtie \in \{<, \leq, =, \geq, >\}$ est un opérateur de comparaison et $a \in \mathbb{Q}_{\geq 0}$ est un nombre rationnel positif.

La sémantique de TCTL_p est aussi définie sur les configurations de $\mathscr{T}_{\mathscr{A}}$, le système de transitions temporisé généré par l'ITA $\mathscr{A}$. Pour une configuration $q = (s, v, \beta)$, on définit inductivement :

$q \models p$	ssi	$p \in \mathit{prop}(s)$
$q \models \varphi \wedge \psi$	ssi	$q \models \varphi$ et $q \models \psi$
$q \models \neg\varphi$	ssi	$q \not\models \varphi$
$q \models \mathsf{A}\varphi_p\,\mathsf{U}_{\bowtie a}\,\psi_p$	ssi	toute exécution $\rho \in \mathit{TExec}_q(\mathscr{A})$ est telle que $\rho \models \varphi_p\,\mathsf{U}_{\bowtie a}\,\psi_p$
$q \models \mathsf{E}\varphi_p\,\mathsf{U}_{\bowtie a}\,\psi_p$	ssi	il existe une exécution $\rho \in \mathit{TExec}_q(\mathscr{A})$ t.q. $\rho \models \varphi_p\,\mathsf{U}_{\bowtie a}\,\psi_p$

où

$\rho \models \varphi_p\,\mathsf{U}_{\bowtie a}\,\psi_p$	ssi	il existe une position π le long de ρ telle que $\mathit{Dur}(\rho_{\leq\pi}) \bowtie a$, $q_\pi \models \psi_p$, et pour toute position $\pi' <_\rho \pi$, $q_{\pi'} \models \varphi_p$

en gardant les notations $\mathit{TExec}_q(\mathscr{A})$ pour l'ensemble des exécutions (totales) de $\mathscr{A}$ depuis la configuration q et q_π pour la configuration à la position π. De même, l'automate $\mathscr{A}$ satisfait la formule φ, noté aussi $\mathscr{A} \models \varphi$ si la configuration initiale de $\mathscr{A}$, $(s_I, \mathbf{0}, \bot)$, vérifie φ.

Model-checking de TCTL_p

Nous montrons que :

Théorème 5.14 (Model-checking de TCTL_p)**.** *Le model-checking de $TCTL_p$ sur les ITA est décidable.*

La preuve consiste à établir des algorithmes pour chacun des quatre cas suivants :
- $\mathsf{E}\,p\,\mathsf{U}_{\leq a}\,r$ et $\mathsf{E}\,p\,\mathsf{U}_{<a}\,r$ (proposition 5.15),
- $\mathsf{E}\,p\,\mathsf{U}_{\geq a}\,r$ et $\mathsf{E}\,p\,\mathsf{U}_{>a}\,r$ (proposition 5.16),
- $\mathsf{A}\,p\,\mathsf{U}_{\geq a}\,r$ et $\mathsf{A}\,p\,\mathsf{U}_{>a}\,r$ (proposition 5.17),
- $\mathsf{A}\,p\,\mathsf{U}_{\leq a}\,r$ et $\mathsf{A}\,p\,\mathsf{U}_{<a}\,r$ (proposition 5.18),

où p et r sont des combinaisons booléennes de propositions atomiques.

Puisque la logique TCTL_p s'interprète sur les états des configurations et non sur les valuations de celles-ci, il est suffisant de fournir des procédures de décision dans le cas des ITA$_-$. L'algorithme de décision commence alors par construire l'ITA$_-$ $\mathscr{A}'$ équivalent à $\mathscr{A}$ et applique la procédure de model-checking à $\mathscr{A}'$. C'est effectivement ce qui est fait dans les deux premiers cas. Cependant, les procédures des deux derniers cas reposent sur le graphe des classes et peuvent s'appliquer directement sur l'ITA.

Proposition 5.15. *Le model-checking de formules $\mathsf{E}\, p\, \mathsf{U}_{\leq a}\, r$ et $\mathsf{E}\, p\, \mathsf{U}_{<a}\, r$ sur un ITA$_-$ est décidable et peut être effectué en NEXPTIME et en NP lorsque le nombre d'horloges est fixé.*

Démonstration. Ces deux formules sont des variantes de propriétés d'accessibilité, où le temps est lui aussi pris en compte. Cette preuve est donc similaire à celle de l'accessibilité dans les ITA$_-$, section 5.3.3.

En utilisant le lemme 5.9 sur un ITA$_-$ à n horloges et dont les transitions sont de taille T, on peut se contenter de considérer les exécutions de taille $B = (T+n)^{3n}$. En effet, la propriété est préservée par un raccourcissement d'une exécution plus longue que B. Notons aussi que l'on peut supposer que cette exécution s'arrête à la première position pour laquelle r est vraie.

La procédure de décision :
- devine un chemin de longueur plus courte que B ;
- vérifie que ce chemin vérifie bien $p \,\mathsf{U}\, r$;
- construit un programme linéaire associé ;
- en résolvant le programme linéaire, vérifie que ce chemin donne bien une exécution ;
- en additionnant les délais successifs, vérifie que la durée de l'exécution avant d'atteindre r est bien inférieure (ou strictement inférieure) à a.

Cet algorithme s'exécute, comme dans la proposition 5.10, en temps $P\big((T+n)^{4n}\big)$ pour un certain polynôme P. □

Ce cas s'apparente à un problème d'accessibilité en temps borné, tel qu'il est décrit dans [BDG$^+$11]. Cependant, le modèle des automates hybrides rectangulaires (RHA) de cet article semble incomparable avec celui des ITA. En effet, bien que les variables des RHA soient plus générales que les horloges des ITA, les gardes et mises à jour des RHA interdisent les contraintes diagonales et additives, et donc ne permettent pas d'utiliser des expressions linéaires.

Proposition 5.16. *Le model-checking de formules $\mathsf{E}\, p\, \mathsf{U}_{\geq a}\, r$ et $\mathsf{E}\, p\, \mathsf{U}_{>a}\, r$ sur un ITA$_-$ est décidable et peut être effectué en NEXPTIME et en NP lorsque le nombre d'horloges est fixé.*

Démonstration. Soit $\mathscr{A}$ un ITA$_-$ à n horloges et T transitions. On pose $B = (T+n)^{3n}$. L'algorithme qui décide si $\mathscr{A} \vDash \mathsf{E}\, p\, \mathsf{U}_{\geq a}\, r$ (ou $\mathscr{A} \vDash \mathsf{E}\, p\, \mathsf{U}_{>a}\, r$) fonctionne de la façon suivante.

L'algorithme devine un chemin de longueur inférieure ou égale à B et construit le programme linéaire associé (comme dans le cas de l'accessibilité, section 5.3.3), puis vérifie que :
- ce chemin correspond bien à une exécution de $\mathscr{A}$;
- il existe une position π sur cete exécution à laquelle r est vraie et avant laquelle p est toujours vraie ;
- la somme des délais avant π excède a (ou excède strictement a dans le cas de $\mathscr{A} \vDash \mathsf{E}\, p\, \mathsf{U}_{>a}\, r$).

Si cette première procédure échoue, l'algorithme devine un chemin de longueur $2B+1$ et vérifie que :
- ce chemin correspond bien à une exécution de $\mathscr{A}$, ce qui peut être fait à l'aide d'un programme linéaire comme auparavant ;

- p est vraie tout au long de ce chemin, mais pas nécessairement dans le dernier état atteint ;
- r est vraie sur le dernier état de ce chemin ;
- soit il existe une transition e de niveau k mettant x_k à jour qui apparaît deux fois, séparées par une séquence σ de transitions de niveau supérieur ou égal à k et entre ces deux occurrences de e, du temps s'écoule ; cette dernière vérification peut être effectuée en additionnant les délais correspondant dans le programme linéaire ;
- soit il existe une transition e de niveau k ne mettant pas x_k à jour qui apparaît deux fois, séparées par une séquence σ de transitions de niveau supérieur ou égal à k ne mettant pas x_k à jour et entre ces deux occurrences de e, une quantité non nulle de temps s'écoule aux niveaux strictement supérieurs à k mais pas au niveau k.

L'algorithme renvoie *vrai* si l'une de ces procédures réussit, et *faux* si elles échouent toutes deux. Nous montrons maintenant que cet algorithme est correct et complet.

Correction. Si la première procédure réussit, alors le chemin deviné est clairement un témoin de $\mathsf{E}\, p\, \mathsf{U}_{\geq a}\, r$ (ou $\mathsf{E}\, p\, \mathsf{U}_{>a}\, r$, selon la formule à vérifier). Si la seconde procédure réussit, alors on peut construire un témoin à partir du chemin deviné de la façon suivante. Le chemin deviné vérifie $p\, \mathsf{U}\, r$, mais pas nécessairement $p\, \mathsf{U}_{\geq a}\, r$. Supposons que σ laisse s'écouler δ unités de temps (avec $\delta > 0$). En répétant $\lceil \frac{a}{\delta} \rceil$ fois [11] la séquence σe, on obtient un chemin vérifiant bien $p\, \mathsf{U}_{\geq a}\, r$. Notons que cette répétition est possible puisque σ ne descend pas sous le niveau k et que soit e met x_k à jour soit le temps ne s'écoule pas au niveau k, donc les valuations après chaque occurrence de σe sont identiques.

Complétude. Soit ρ un témoin de $\mathsf{E}\, p\, \mathsf{U}_{\geq a}\, r$ de longueur h minimale. Puisque ρ est minimal, r est satisfaite dans le dernier état visité et p l'est (au moins) à chaque position antérieure. Si $h \leq B$, alors la première procédure devine ρ. Autrement, $h > B$ et l'un des cas du lemme 5.9 se produit dans les $B+1$ premières transitions de ρ :

- Une même transition e de niveau k mettant x_k à jour apparaît aux positions i et j dans ρ, avec $i < j \leq B+1$. On note donc ces deux instances de e respectivement e_i et e_j. Deux sous-cas sont alors à considérer : soit du temps s'est écoulé entre e_i et e_j, soit toutes les transitions sont instantanées :
  - Si aucun temps ne s'est écoulé, la sous-exécution entre e_i et e_j ainsi que e_j peuvent être supprimées sans que cela n'affecte la satisfaction de $p\, \mathsf{U}_{\geq a}\, r$. Cette nouvelle exécution est plus courte que ρ, ce qui contredit la minimalité.
  - Autrement, du temps s'est écoulé. On écrit $\rho = \rho_1 \cdot e_i \cdot \sigma \cdot e_j \cdot \rho_2$. En appliquant le lemme 5.9 à ρ_2, on peut obtenir une exécution ρ_3 de longueur plus petite que B telle que $\rho' = \rho_1 \cdot e_i \cdot \sigma \cdot e_j \cdot \rho_3$ est aussi une exécution de $\mathscr{A}$. On a bien $|\rho'| \leq 2B+1$. Remarquons que ρ_3 est un sous-mot de ρ_2 et que ces deux exécutions relient les même configurations, à la valeur de β près. En conséquence, le dernier état de ρ' est le même que celui de ρ ; celui-ci vérifie donc r. De plus, p est satisfaite le long de ρ'. Donc ρ' est devinée par la seconde procédure.

11. Cette séquence peut être répétée une fois de plus dans le cas de $\mathsf{E}\, p\, \mathsf{U}_{>a}\, r$.

- Une même transition e de niveau k qui ne met pas x_k à jour apparaît en deux occurrences, séparées par σ tel que
  - σ ne contient que des états de niveau supérieur ou égal à k ;
  - aucune transition de σ ne met x_k à jour ;
  - σ laisse s'écouler δ unités de temps au niveau k ;
  - e a pour source un état non urgent.

  Dans ce cas σ peut être remplacé par un pas de temps de durée δ sans changer la véracité de $p \mathsf{U}_{\geq a} r$. Cette exécution est plus courte, ce qui contredit l'hypothèse de minimalité de ρ.
- Une même transition e de niveau k qui ne met pas x_k à jour apparaît deux fois, séparées par des transitions ne laissant pas le temps s'écouler pour x_k.
  - Si aucun temps ne s'écoule à des niveaux strictement supérieurs à k, la sous-exécution séparant les deux occurrences ainsi qu'une de ces occurrences peuvent être supprimées sans changer ni la suite de l'exécution ni la véracité de $p \mathsf{U}_{\geq a} r$. Une fois encore, l'hypothèse de minimalité de ρ est violée.
  - Autrement, on écrit, comme dans le cas plus haut, $\rho = \rho_1 \cdot e_i \cdot \sigma \cdot e_j \cdot \rho_2$, où e_i et e_j sont les deux occurrences de e. L'exécution ρ_2 peut être réduite en ρ_3 plus courte que B (en appliquant le lemme 5.9). La taille de $\rho' = \rho_1 \cdot e_i \cdot \sigma \cdot e_j \cdot \rho_3$ est donc plus petite que $2B+1$. Cette exécution sera donc trouvée par la seconde procédure.

La complétude dans le cas de $\mathsf{E}\, p \mathsf{U}_{>a} r$ est similaire. □

Dans le cas des formules quantifiant existentiellement sur les chemins, un témoin est toujours un chemin fini. Un témoin (que la formule n'est, cette fois, pas vérifiée) est potentiellement infini dans le cas des formules quantifiant universellement sur les chemins. La construction d'une exécution infinie repose sur une exploration (non déterministe) du graphe des classes de la section 5.2. Ainsi la complexité de l'algorithme est bien plus élevée pour les formules quantifiant universellement sur les chemins que pour celles qui quantifient existentiellement.

Proposition 5.17. *Le model-checking de formules $\mathsf{A}\, p \mathsf{U}_{\geq a} r$ et $\mathsf{A}\, p \mathsf{U}_{>a} r$ sur un ITA est décidable et peut être effectué en 2-EXPSPACE et en PSPACE lorsque le nombre d'horloges est fixé.*

Démonstration. On considère un ITA $\mathscr{A}$ ayant n horloges et dont les transitions sont de taille T. On pose $B = (T+n)^{n \times 2^{n(\log(n)+4)}}$. L'algorithme cherche à deviner un contre-exemple afin de prouver que $\mathscr{A} \not\models \mathsf{A}\, p \mathsf{U}_{\geq a} r$ (ou $\mathscr{A} \not\models \mathsf{A}\, p \mathsf{U}_{>a} r$). Un contre exemple de $p \mathsf{U}_{\geq a} r$ (ou $p \mathsf{U}_{>a} r$) est une exécution qui :

- soit est finie (et ne peut plus évoluer) et le suffixe suivant la borne de temps ne vérifie pas $p \mathsf{U} r$
- soit est infinie et ne vérifie jamais r après la borne de temps.

L'algorithme fonctionne de la manière suivante. Il devine un chemin de longueur inférieure ou égale à B et vérifie que :

- ce chemin fournit une exécution ρ (à l'aide d'un programme linéaire) ;
- cette exécution est maximale, c'est-à-dire qu'aucune transition ne peut être tirée depuis la dernière configuration de ρ ;
- il existe une position π de ρ à un instant du temps strictement antérieur [12] à a

12. Ou antérieur ou simultané dans le cas de $\mathsf{A}\, p \mathsf{U}_{>a} r$.

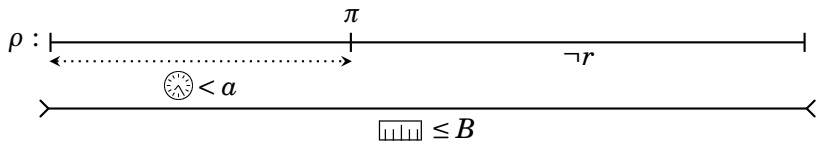


FIGURE 5.15: Démonstration de la proposition 5.17 : un contre-exemple fini (Cas 1).

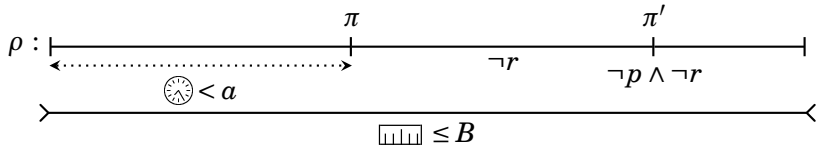


FIGURE 5.16: Démonstration de la proposition 5.17 : un contre-exemple fini (Cas 2).

telle que :

Cas 1 : soit r n'est pas satisfaite à partir de π (voir figure 5.15) ;

Cas 2 : ou alors il existe une position π' après π à laquelle ni p ni r n'est satisfaite, et r n'est jamais satisfaite entre π et π' (voir figure 5.16).

Si cette première procédure échoue, l'algorithme devine :

– Une classe R du graphe $\mathscr{G}_{\mathscr{A}}$ et un cycle C dans $\mathscr{G}_{\mathscr{A}}$, démarrant en R (sans construire ni le graphe ni le cycle), tels que C contient au moins une transition discrète et ne visite que des états (à travers les classes) où $\neg r$ est vérifiée ;

– un chemin dans l'automate $\mathscr{A}$ de longueur inférieure ou égale à la borne B ;

et vérifie :

– que ce chemin produit une exécution ρ qui se termine dans une configuration $(s,v,\beta) \in [\![R]\!]$ (cette vérification est effectuée à l'aide d'un programme linéaire) ;

– qu'il existe une position π de ρ à un instant du temps antérieur[13] à a après laquelle r n'est plus satisfaite.

Si l'une de ces procédure réussit, l'algorithme renvoie *faux*, et il renvoie *vrai* lorsque ces deux procédures échouent. Notons que le graphe des classes seul n'est pas suffisant, car l'abstraction n'est pas assez précise pour vérifier l'existence de la position π.

Correction. Montrons que cet algorithme est correct : lorsque l'une des procédure réussit, il existe un contrexemple à la formule $\mathsf{A}\, p\, \mathsf{U}_{\geq a}\, r$ (ou $\mathsf{A}\, p\, \mathsf{U}_{>a}\, r$). Dans le cas de la première procédure, il est clair que l'exécution devinée ne satisfait pas $p\, \mathsf{U}_{\geq a}\, r$ (ou $p\, \mathsf{U}_{>a}\, r$). Dans le cas de la seconde procédure, on montre qu'il existe un contre-exemple infini. Considérons la configuration (s,v,β) atteinte après ρ. Puisque $(s,v,\beta) \in [\![R]\!]$, tout chemin σ dans $\mathscr{G}_{\mathscr{A}}$ depuis R correspond à une exécution depuis (s,v,β) qui traverse les classes de σ. En prenant $\sigma = C^{\omega}$ le chemin infini constitué de cycles C, on a l'existence d'une exécution infinie ρ_{∞} dans $\mathscr{A}$. De plus, r est n'est jamais satisfaite le long de C, donc le long de ρ_{∞} et $\rho \cdot \rho_{\infty}$ est un contre-exemple de la formule $p\, \mathsf{U}_{\geq a}\, r$ (ou $p\, \mathsf{U}_{>a}\, r$).

13. Ou antérieur ou simultané dans le cas de $\mathsf{A}\, p\, \mathsf{U}_{>a}\, r$.

Complétude. Supposons qu'il existe un contre-exemple fini ρ. Soit $\mathscr{A}'$ l'ITA$_-$ acceptant les mêmes mots temporisés que $\mathscr{A}$ (voir proposition 5.8). On note T' la taille des transitions de $\mathscr{A}'$ et on pose $B' = (T'+n)^{3n}$ (la borne du lemme 5.9 pour $\mathscr{A}'$). On a $B' \leq B$ (voir démonstration du théorème 5.7, section 5.3.4).

Si $|\rho| \leq B$, cette exécution est devinée par la première procédure. Autrement, soit ρ' l'exécution correspondant à ρ dans l'ITA$_-$ $\mathscr{A}'$. L'exécution ρ' accepte le même mot temporisé que ρ. De plus, la suite des états traversés par ρ' peut être projetée sur celle de ρ, en omettant les états de la forme $(s^-,-)$: à la sous-séquence

$$(s_0^+,-) \to \cdots \to (s_{m-1}^+,-) \to (s_m^-,-) \to (s_m^+,-)$$

dans ρ' correspond la sous-séquence $s_0 \to \cdots \to s_{m-1} \to s_m$ dans ρ. Remarquons que $|\rho'| \geq |\rho|$ et que ρ' est aussi un contre-exemple dans $\mathscr{A}'$ de la formule $p \,\mathsf{U}_{\geq a}\, r$ (ou $p \,\mathsf{U}_{>a}\, r$). Puisque $|\rho'| \geq B \geq B'$, l'un des cas du lemme 5.9 s'applique. Par suppression ou remplacement de sous-exécutions par des pas de temps, comme dans la preuve de la proposition 5.11, on obtient une exécution σ' dans $\mathscr{A}'$ telle que $|\sigma'| \leq B' \leq B$ et σ' est toujours un contre-exemple [14] (dans $\mathscr{A}'$) de la formule $p \,\mathsf{U}_{\geq a}\, r$ (ou $p \,\mathsf{U}_{>a}\, r$).

Soit σ l'exécution de $\mathscr{A}$ correspondant à σ'. On a $|\sigma| \leq |\sigma'| \leq B' \leq B$ et σ est un contre-exemple (dans $\mathscr{A}$) de $p \,\mathsf{U}_{\geq a}\, r$ (ou $p \,\mathsf{U}_{>a}\, r$). Donc σ est deviné par la première procédure.

Supposons qu'il existe un contre-exemple infini ρ. Considérons le chemin σ dans le graphe des classes $\mathscr{G}_{\mathscr{A}}$, correspondant à ρ. Ce chemin est aussi infini ; ou plutôt, σ contient une infinité de transitions discrètes. Puisque σ visite un nombre infini de classes, il contient un cycle C contenant au moins une transition discrète. Soit R l'une des classes du cycle C, et soit ρ_0 le préfixe de ρ menant à une configuration (s,v,β) de $[\![R]\!]$. Comme dans le cas d'un contre-exemple fini, il existe une exécution ρ_1 de longueur plus petite que B menant à la configuration (s,v,β). Le cycle C, la classe R, et l'exécution ρ_1 sont tous les trois devinés par la seconde procédure, qui réussit donc.

Complexité. La première procédure termine en 2-NEXPTIME : on devine un chemin de taille B et on résout un programme linéaire de taille polynomiale en B. La complexité de la seconde procédure, qui donne celle de l'algorithme global, est majorée par le fait de deviner la classe R et le cycle C, ce qui requiert de stocker deux classes. Chaque classe est de taille doublement exponentielle par rapport à la taille de $\mathscr{A}$.

Lorsque le nombre d'horloges est fixé, la complexité de la première procédure tombe à NP, tandis que la seconde requiert de garder deux classes de taille polynomiale, d'où la complexité en PSPACE. □

Dans le dernier cas, un algorithme spécifique est superflu. En effet, ceux des propositions 5.16 et 5.17 peuvent être réutilisés.

Proposition 5.18. *Le model-checking de formules $\mathsf{A}p \,\mathsf{U}_{\leq a}\, r$ et $\mathsf{A}p \,\mathsf{U}_{<a}\, r$ sur un ITA est décidable et peut être effectué en 2-EXPSPACE et en PSPACE lorsque le nombre d'horloges est fixé.*

14. En effet, puisqu'il s'agit d'un contre-exemple, σ' peut prendre moins de temps que σ.

Démonstration. On remarque que

$\mathsf{A}\, p\, \mathsf{U}_{\leq a}\, r \;=\; (\mathsf{A}\, p\, \mathsf{U}_{\geq 0}\, r) \wedge \neg(\mathsf{E}\, \neg r\, \mathsf{U}_{>a}\, \mathbf{t})$ et que $\mathsf{A}\, p\, \mathsf{U}_{<a}\, r \;=\; (\mathsf{A}\, p\, \mathsf{U}_{\geq 0}\, r) \wedge \neg(\mathsf{E}\, \neg r\, \mathsf{U}_{\geq a}\, \mathbf{t})$.

ce qui permet de réutiliser les procédures précédentes. □

Bornes de complexité. Les bornes de complexités des quatre procédures ci-dessus donnent une complexité du model-checking de TCTL_p sur les ITA en 2-EXPSPACE. Dans le cas de formules purement existentielles, la complexité est en 2-NEXPTIME, la transformation d'un ITA en ITA$_-$ introduisant une explosion exponentielle.

Lorsque le nombre d'horloges est fixé, la complexité devient PSPACE, que ce soit dans le cas des ITA ou des ITA$_-$. Et dans ce cas, les formules purement existentielles sont vérifiées en NP.

5.5 Expressivité et clôture de ITL

L'indécidabilité du model-checking sur les ITA de fragments décidables (efficacement) sur les TA laisse penser que les ITA peuvent exprimer plus que les automates temporisés. On compare dans cette section le pouvoir d'expression des ITA avec celui d'autre modèles vis-à-vis des langages acceptés : celui des automates temporisés [AD94] et celui des automates temps-réels contrôlés (CRTA) [DZ98]. On considère ensuite des propriétés de clôture de la famille ITL.

5.5.1 Le modèle des CRTA

Les automates temps-réels contrôlés sont une extension des automates temporisés définis de la manière suivante. Les états et les horloges sont partitionnés en couleurs appartenant à un ensemble Ω et à chaque état est associé une *vitesse* (ou pente) rationnelle. Les horloges actives sont celles de la couleur de l'état courant. Une horloge active évolue selon la vitesse attribuée à l'état courant. Une horloge qui n'est pas active n'évolue pas. La définition présentée ici est légèrement modifiée par rapport à celle de [DZ98] afin de la rendre plus proche de celle des ITA, sans pour autant modifier le pouvoir d'expression vis-à-vis des langages acceptés.

Définition 5.8 (Automate temps-réel contrôlé)**.** *Un* Automate temps-réel contrôlé *(CRTA) est un undécuplet* $\mathscr{A} = \langle S, Lab, X, \Omega, \lambda, vel, \Delta, max, min, s_I, F\rangle$ *où :*

- S *est un ensemble fini d'états ;*
- Lab *est un ensemble fini d'étiquettes ;*
- X *est un ensemble fini d'horloges ;*
- Ω *est un ensemble fini de couleurs ;*
- *la fonction* $\lambda : (S \uplus X) \to \Omega$ *associe à chaque horloge et à chaque état une couleur ;*
- *la fonction* $vel : S \to \mathbb{Q}$ *associe à chaque état une vitesse rationnelle ;*
- $\Delta \subseteq S \times \mathscr{C}_0(X) \times (Lab \uplus \{\varepsilon\}) \times \mathscr{U}_0(X) \times S$ *est la relation de transition ;*
- *les fonctions de borne supérieure* $max : X \to \mathbb{Q}$ *et inférieure* $min : X \to \mathbb{Q}$ *sont telles que pour toute horloge* $x \in X$, $min(x) \leq 0 \leq max(x)$ *et pour toute constante* $b \in \mathbb{Q}$ *telle que* $x + b \bowtie 0$ *est une contrainte apparaissant dans un garde de* Δ, $min(x) \leq -b \leq max(x)$ *;*
- $s_I \in S$ *est l'état initial ;*
- $F \subseteq S$ *est l'ensemble des états finaux.*

Considérons le CRTA $\mathscr{A}_{18}$ de la figure 5.17 [DZ98]. Cet automate a deux horloges x et y bornées toutes deux par $min(x) = min(y) = -1$ et $max(x) = max(y) = 1$. $\mathscr{A}_{18}$ n'a qu'une seule couleur, c'est-à-dire que toutes les horloges sont toujours actives. Le langage accepté par l'automate est

$$L_{18} = \{(c,\tau)(c,2\tau)\cdots(c,n\tau) \mid 0 < \tau < 1, n \in \mathbb{N}\}$$

où les occurrences successives de c sont toutes séparées par la même durée de temps, comprise entre 0 et 1. Dans une exécution de $\mathscr{A}_{18}$, la remise à zéro de y lors du franchissement de la première transition au temps τ (avec $0 < \tau < 1$ à cause de la garde) a pour conséquence que, dans la suite de l'exécution, on a toujours $x = y + \tau$. Les valeurs de x oscillent ensuite entre 0 et τ tandis que celles de y oscillent entre 0 et $-\tau$. Le passage de l'une de ces valeurs à 0, tous les τ unités de temps, autorise le changement d'état entre q_1 et q_2, et donc le changement de la pente des horloges.

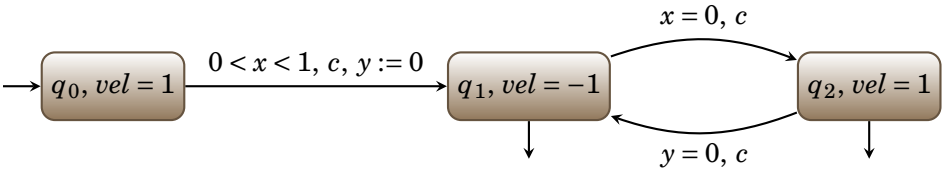


FIGURE 5.17: CRTA $\mathscr{A}_{18}$ à une couleur acceptant L_{18}.

La sémantique originale des CRTA est assez compliquée. De manière à obtenir la décidabilité du problème d'accessibilité, elle assure qu'au moment où l'on entre dans un état s dans laquelle une horloge x est active, les conditions suivantes sont vérifiées :

- si $vel(s) > 0$, alors $x \geq min(x)$;
- si $vel(s) < 0$, alors $x \leq max(x)$.

On remplace cette condition sémantique par une condition syntaxique équivalente. Par exemple, lorsqu'une transition de garde g et de mise à jour u entre dans un état s avec $vel(s) < 0$ et où seule l'horloge x est active, on remplace cette transition par deux transitions :

- l'une a pour garde $g \wedge x > max(x)$ et pour mise à jour $u \wedge x := 0$;
- l'autre a pour garde $g \wedge x \leq max(x)$ et pour mise à jour u.

Dans le cas où k horloges sont actives dans s, il faut 2^k transitions, afin de déterminer le sous-ensemble des horloges qui sont déjà en dessous de leur borne supérieure. Le cas où $vel(s) > 0$ est analogue.

La sémantique d'un CRTA est un système de transition $\mathscr{T}_{\mathscr{A}}$ dont les états sont les paires (s,v), où $v : X \to \mathbb{R}$ est une valuation. Les transitions discrètes s'appliquent sur les CRTA de la même manière que sur les ITA (en supposant que tous les états d'un CRTA ont une politique paresseuse). Pour une configuration (s,v) et pour $d \in \mathbb{R}_{\geq 0}$, le pas de temps $(s,v) \xrightarrow{d} (s,v')$ est défini de la façon suivante. Pour $x \in X$, si $\lambda(x) = \lambda(s)$ alors $v'(x) = v(x) + vel(s) \times d$ et $v'(x) = v(x)$ autrement.

Les notions d'exécutions, de mot temporisé (ou non), de langage temporisé (ou non), de longueur et de durée des exécutions, sont les mêmes pour les CRTA que pour les ITA. On note CRTL l'ensemble des langages temporisés définis par des CRTA.

5.5.2 Expressivité comparée de ITL, TL et CRTL

Le pouvoir d'expression des ITA est évalué en comparant ITL (la famille des langages temporisés définis par des ITA) à TL (ceux définis par des automates temporisés) et à CRTL. Rappelons que TL est strictement incluse dans CRTL. Nous montrons que :

Théorème 5.19 (Expressivité comparée de ITL, TL et CRTL)**.** *Les familles ITL et TL sont incomparables. Les familles ITL et CRTL sont incomparables.*

ITL n'est pas incluse dans TL, ni dans CRTL

Nous montrons ici que le modèle des ITA ne peut pas être réduit à celui des TA, ni à celui des CRTA.

Proposition 5.20.

1. *Il existe un langage de ITL dont tous les mots sont de longueur* 2 *qui n'est pas dans TL.*
2. *Il existe un langage de ITL qui n'est pas dans CRTL.*

Démonstration. Afin de prouver le premier point, on considère l'ITA $\mathscr{A}_{19}$ de la figure 5.18. Cet ITA accepte le langage temporisé

$$L_{19} \;=\; \mathscr{L}(\mathscr{A}_{19}) \;=\; \left\{(a,1-\tau)\left(b,1-\frac{\tau}{2}\right) \,\middle|\, \tau \in [0,1[\right\}.$$

Supposons, par contradiction, qu'un automate temporisé $\mathscr{A}$ (ayant potentiellement des ε-transitions) accepte le langage L_{19}. Soit $m \in N$ la granularité de $\mathscr{A}$, c'est-à-dire le plus petit entier tel que toutes les constantes (rationnelles) apparaissant dans les gardes et mises à jour de $\mathscr{A}$ peuvent s'écrire $\frac{k}{m}$ pour un certain $k \in \mathbb{Z}$. On considère le mot temporisé $w = \left(a,1-\frac{1}{m}\right)\left(b,1-\frac{1}{2m}\right) \in L_3$. Le mot w est accepté par un chemin fini dans $\mathscr{A}$. Considérons l'automate temporisé $\mathscr{A}'$ qui consiste en cet exact chemin ; les états visités plusieurs fois peuvent avoir été renommés. On a $w \in \mathscr{L}(\mathscr{A}') \subseteq \mathscr{L}(\mathscr{A}) = L_3$. De plus, l'automate $\mathscr{A}'$ ne contient aucun cycle. Grâce au resultat de [BPDG98], on peut construire un automate temporisés $\mathscr{A}''$ tel que :

- $\mathscr{A}''$ ne contient pas d'ε-transitions ;
- $\mathscr{L}(\mathscr{A}'') = \mathscr{L}(\mathscr{A}')$;
- la granularité de $\mathscr{A}''$ est la même que celle de $\mathscr{A}'$, c'est-à-dire m.

Le chemin acceptant w dans $\mathscr{A}''$ ne contient donc que deux transitions :

$$p_0 \xrightarrow{g_1,a,u_1} p_1 \xrightarrow{g_2,b,u_2} p_2.$$

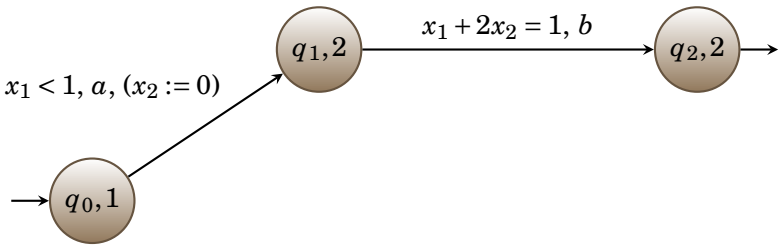


FIGURE 5.18: Un ITA $\mathscr{A}_{19}$ acceptant L_{19}.

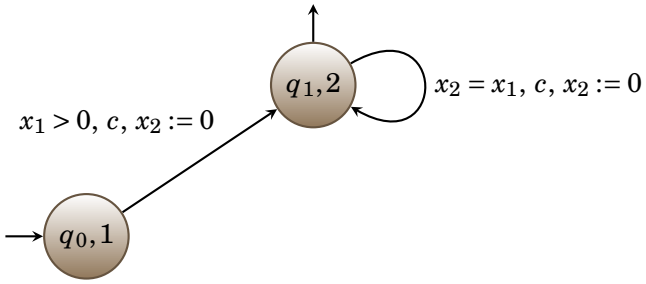


FIGURE 5.19: Un ITA $\mathscr{A}_{20}$ acceptant L_{20}.

Après le franchissement de a, les seules valeurs possibles pour les horloges sont $1-\frac{1}{m}$ ou 0, selon les remises à zéro de u_1. Au moment de franchir b, les valeurs d'horloges sont donc soit $1-\frac{1}{2m}$ soit $\frac{1}{2m}$. Soit $x+\alpha \bowtie 0$ une contrainte de g_2, avec $\alpha \in \mathbb{Q}$. Si $\bowtie$ est la relation « = », alors $\alpha = \frac{1}{2m} - 1$ ou $\alpha = -\frac{1}{2m}$, ce qui est impossible puisque la granularité de $\mathscr{A}''$ est de m. Si $\bowtie$ est « < », « ≤ », « ≥ » ou « > », ou si $g_2 = \top$, alors un mot $\left(a, 1-\frac{1}{m}\right)(b,\tau)$ avec $\tau \neq 1-\frac{1}{2m}$ est accepté par $\mathscr{A}''$, ce qui contredit $\mathscr{L}(A'') \subseteq L_{19}$.

Afin de prouver le second cas, on considère le langage

$$L_{20} \;=\; \{(c,\tau)(c,2\tau)\cdots(c,n\tau) \mid \tau > 0, n \in \mathbb{N}\}$$

accepté par l'ITA $\mathscr{A}_{20}$ de la figure 5.19. Selon [DZ98], ce langage n'est pas accepté par un CRTA. Notons que L_{20} difère de L_{18} par le fait que le délai entre deux occurrences successives de c n'est ici pas borné. Cette absence de borne rend impossible une adaptation du CRTA $\mathscr{A}_{18}$, car les horloges doivent être bornées (à l'aide des fonctions *min* et *max*) lors de l'entrée dans un état. □

TL n'est pas incluse dans ITL

Nous montrons maintenant qu'il existe un langage de TL qui n'est accepté par aucun ITA.

Soit L_4 le langage défini par

$$\begin{aligned} L_4 \;=\; \big\{&(a,\tau_1)(b,\tau_2)\ldots(a,\tau_{2p+1})(b,\tau_{2p+2}) \,\big|\, p \in \mathbb{N}, \\ &\forall i \in \{0,\ldots,p\},\ \tau_{2i+1} = i+1 \text{ et } i+1 < \tau_{2i+2} < i+2, \\ &\forall i \in \{1,\ldots,p\},\ \tau_{2i+2} - \tau_{2i+1} < \tau_{2i} - \tau_{2i-1}\big\} \end{aligned}$$

accepté par l'automate temporisé $\mathscr{A}_4$ de la figure 5.20. Ce langage, initialement proposé par [AD94], accepte les mots de $(ab)^*$ temporisés de la façon suivante : un a apparaît à chaque unité de temps, et chaque occurrence de b est de plus en plus proche de l'occurrence de a qui la précède.

Proposition 5.21. *Le langage L_4 n'est pas dans ITL.*

Démonstration. Supposons, par contradiction, que L_4 est dans ITL. L_4 est accepté par un ITA$_-$ $\mathscr{A}$ à n horloges et T transitions. Soit $B = (T+n)^{3n}$ et on considère le mot temporisé $w = (a,\tau_1)(b,\tau_2)\cdots(a,\tau_{2B+1})(b,\tau_{2B+2}) \in L_4$. Le mot w est accepté par une exécution ρ de taille minimale dans $\mathscr{A}$; on a cependant $|\rho| > B$. L'un des cas du lemme 5.9 se produit dans les B premières transitions.

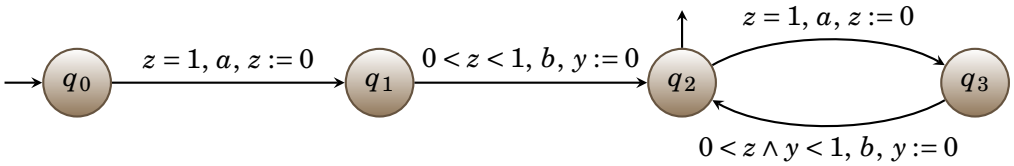


FIGURE 5.20: L'automate temporisé $\mathscr{A}_4$ acceptant L_4.

- Ou bien une transition e de niveau k mettant x_k à jour apparaît deux fois séparées par une exécution σ de niveau supérieur ou égal à k. Notons que les valuations après la première occurrence de e et après la seconde sont exactement les mêmes. On distingue plusieurs cas en fonction du sous mot accepté par σe.
  - Dans le cas où σe lit ε, on note δ la durée écoulée durant σe. Si $\delta = 0$, alors σe peut être supprimé sans affecter ni la suite de l'exécution ni le mot accepté, ce qui contredit la minimalité de ρ. Si $\delta \geq 1$, alors il y a une unité de temps qui ne contient pas de b, ce qui est contradictoire avec la définition de L_4. Autrement, $0 < \delta < 1$. En supprimant σe, on obtient une exécution ρ' (acceptée par $\mathscr{A}$) dans laquelle le suffixe après e est décalé de δ. L'occurrence suivante de la lettre a, qui apparaissait à une certaine date $i \in \mathbb{N} \setminus \{0\}$, apparaît dans ρ' à une date $i - \delta$ non-entière. Donc le mot accepté par ρ' n'est pas dans L_4, ce qui est contradictoire.
  - Si σe lit plus de a que de b ou plus de b que de a, en supprimant σe on obtient une exécution acceptant un mot ne vérifiant pas l'alternance de a et de b et qui donc n'appartient pas à L_4.
  - Si σe lit autant de a que de b (et au moins une fois chacune de ces lettres), en dupliquant σe on obtient une exécution acceptant un mot où un même écart entre un a et le b suivant est répété, et qui donc n'est pas dans L_4.
- Ou bien une transition e de niveau k non urgente apparaît deux fois séparées par une exécution σ de niveau supérieur ou égal à k telle que σe ne met pas x_k à jour. On remplace alors $e\sigma$ par un pas de temps de même durée.
  - Si $e\sigma$ lit ε, la nouvelle exécution contredit la minimalité de ρ.
  - Si $e\sigma$ lit seulement un b, le nouveau mot viole l'alternance de a et de b dans L_4.
  - Si $e\sigma$ lit au moins un a, le nouveau mot n'a pas de a à une date entière, et donc n'est pas dans L_4.
- Ou alors une transition e de niveau k apparaît deux fois séparées par une exécution σ de niveau supérieur ou égal à k telle que σe ne met pas à jour x_k et ne laisse pas de temps s'écouler au niveau k. On a la même disjonction que dans le cas d'une mise à jour de x_k par e, puisque σe peut aussi bien être supprimée que dupliquée. □

Remarques

On peut voir que le second point de la proposition 5.20 suffit à montrer que ITL n'est pas incluse dans TL. Cependant, la simplicité du langage L_{19} dans ITL\TL montre que c'est avant tout les expressions linéaires qui ne sont pas exprimables

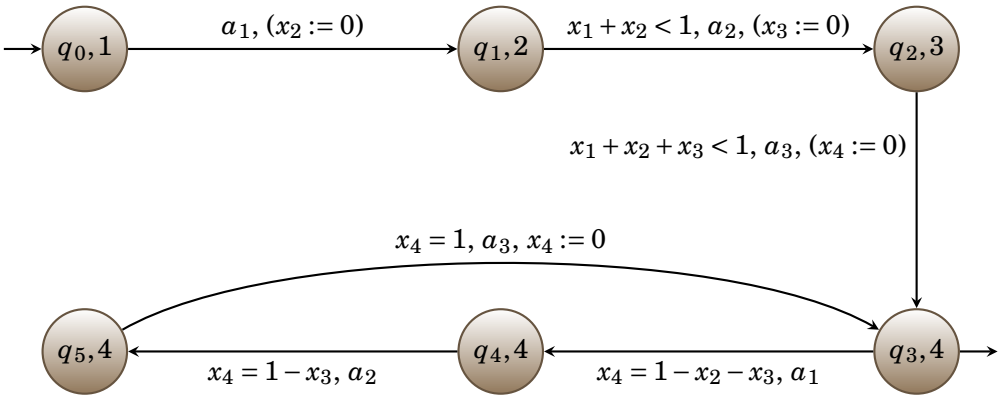


FIGURE 5.21: Un ITA$_-$ pour M_3

dans les automates temporisés. De plus, on peut remarquer que dans les deux points de la proposition 5.20, ce sont des ITA$_-$ qui sont utilisés.

Dans le cas de L_4, c'est le nombre arbitraire de délais différents qui empêche le langage d'être accepté par un ITA. Intuitivement, un ITA peut mémoriser n délais différents à l'aide de $n+1$ horloges. Par exemple, pour les étiquettes $Lab = \{a_1, \ldots, a_n\}$, le langage

$$\begin{aligned} M_n \quad = \quad & \{(a_1,\tau_1)\ldots(a_n,\tau_n)(a_1,\tau_1+1)\ldots(a_n,\tau_n+1)\ldots(a_1,\tau_1+p)\ldots(a_n,\tau_n+p) \\ & \mid p \geq 1, \tau_1 \leq \tau_2 \leq \cdots \leq \tau_n \leq \tau_1 + 1\} \end{aligned}$$

est accepté par un ITA$_-$ ayant $n+1$ horloges : les n premières servent de mémoire tandis que la $n+1$-ième sert de niveau de travail, une fois passée la phase d'initialisation (qui fixe les durées $\tau_1, \ldots, \tau_n$). L'automate de la figure 5.21 donne un exemple d'ITA pour le cas $n = 3$.

La différence de nature entre L_{19} et L_4 montre qu'un ITA est surtout limité par le nombre de ses horloges, alors qu'un TA est limité par ses constantes. Cela rejoint d'ailleurs les résultats de complexité obtenus tout au long de ce chapitre : le critère principal de complexité d'un ITA est son nombre d'horloges.

5.5.3 Propriétés de clôture de ITL

Nous étudions ici les propriétés de clôture des langages définis par un ITA. Plus précisément nous montrons que :

Théorème 5.22 (Clôture de ITL)**.** *La famille ITL est close par union. La famille ITL n'est close ni par complémentation ni par intersection.*

La clôture par union est évidente : il suffit de joindre les états initiaux des différents ITA par des transitions ε (sans garde ni mise à jour) depuis un nouvel état initial urgent. Les propositions suivantes traitent de la complémentation et de l'intersection, respectivement.

Proposition 5.23. *Il existe un langage de ITL dont le complémentaire n'est pas dans ITL.*

Démonstration. On montre que le langage $\overline{L_4}$, complémentaire de L_4, est dans ITL. Un mot est dans $\overline{L_4}$ si l'une de ces conditions est vérifiée :

1. un a se produit à une date non entière ;
2. il n'y a pas de a à une date entière (qui n'est pas après la fin du mot) ;
3. un b se produit à une date entière ;
4. il n'y a pas de b dans un intervalle $[i, i+1]$ où a se produit à la date $i \in \mathbb{N}$;
5. il y a au moins deux b dans un intervalle $[i, i+1]$ où a se produit à la date $i \in \mathbb{N}$;
6. il y a une occurrence de $abab$ telle que la différence de temps entre le premier a et le premier b est plus petite (ou égale) à la différence de temps entre le second a et le second b.

Un ITA à une seule horloge peut être construit aisément pour chacune des cinq premières conditions. Un ITA (qui est en fait un ITA$_-$) à deux horloges qui accepte les mots vérifiant la dernière condition est représenté figure 5.22 : le niveau 1 garde en mémoire la durée entre le premier a et le premier b, tandis qu'au niveau 2 on s'assure que le délai entre la seconde occurrence de a et celle de b est inférieur à celui en mémoire. □

Proposition 5.24. *Il existe deux langages de ITL dont l'intersection n'est pas un langage de ITL.*

Démonstration. Le langage L_4 est l'intersection des langages L'_4 et L''_4 définis de la façon suivante :

$$L'_4 = \{(a,1)(b,\tau_1)\cdots(a,n)(b,\tau_n) \mid n \geq 1 \text{ et } \forall i \in \{1,\ldots,n\}, i < \tau_i < i+1\}$$

$$L''_4 = \left\{(a,\tau'_1)(b,\tau_1)\cdots(a,\tau'_n)(b,\tau_n) \,\middle|\, n \geq 1 \text{ et } \forall i \in \{1,\ldots,n-1\}, \tau_{i+1} - \tau_i < 1\right\}$$

Les mots de L'_4 sont ceux où les a sont à des dates entières et il y a un b entre deux occurrences de a. Les mots de L''_4 demandent que les occurrences de b soient éloignées de moins d'une unité de temps (strictement). Un mot de $L'_4 \cap L''_4$ a donc des a à des dates entières, intercalés de b séparés par strictement moins d'une unité de temps, ce qui a pour effet de rapprocher chaque occurrence de b du a qui le précède. On a bien $L'_4 \cap L''_4 = L_4$.

Un ITA à une seule horloge (et donc un seul niveau) acceptant L'_4 (respectivement L''_4) est représenté 5.23 (respectivement 5.24). On remarque que ces automates ont la même structure que $\mathcal{A}_4$, qui accepte leur intersection. En fait, chacun de ces

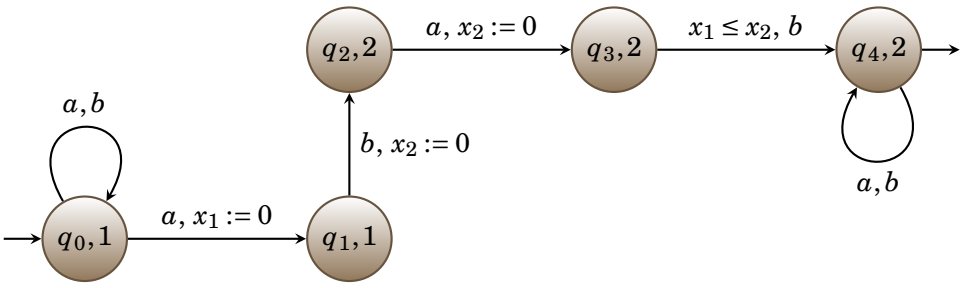


FIGURE 5.22: Un ITA acceptant les mots vérifiant le dernier cas de $\overline{L_4}$.

automates peut être vu comme une projection de $\mathscr{A}_4$ sur les horloges y et z, respectivement. L'intersection des langages a donc besoin de deux horloges au niveau 1, ce qui est interdit par la syntaxe des ITA. □

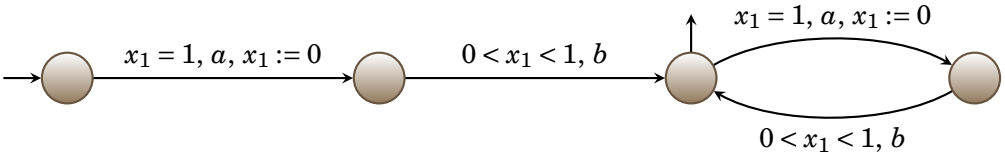


FIGURE 5.23: Un ITA $\mathscr{A}'_4$ acceptant L'_4.

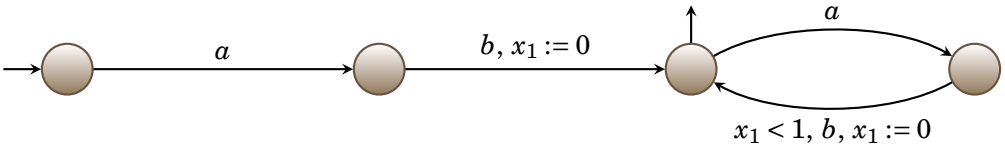


FIGURE 5.24: Un ITA $\mathscr{A}''_4$ acceptant L''_4.

5.6 Extensions des ITA

Dans la section précédente, nous avons montré que les familles de langages ITL et TL, et ITL et CRTL sont incomparables. Les modèles sont maintenant combinés afin d'obtenir une classe comprenant les deux familles. La combinaison doit toutefois conserver de bonnes propriétés de décidabilité, en particulier pour ce qui est de problème d'accessibilité.

5.6.1 Un produit synchronisé entre ITA et TA

Nous présentons dans un premier temps un modèle correspondant à une synchronisation d'un automate temporisé et d'un automate temporisé à interruptions. Ce modèle peut être vu comme celui d'un ITA observé par un automate temporisé. La synchronisation, en cassant la hiérarchie stricte entre les niveaux d'interruption, rend le problème d'accessibilité indécidable.

Définition 5.9 (ITA×TA). *Soit $\mathscr{A}^i = \langle S^i, Lab, X, \lambda^i, pol^i, \Delta^i, s^i_I, F^i \rangle$ un ITA (sans propositions atomiques) et $\mathscr{A}^t = \langle S^t, Lab, Y, \Delta^t, s^t_I, F^t \rangle$ un TA sur le même alphabet. Le produit synchronisé de $\mathscr{A}^i$ et de $\mathscr{A}^t$ est un ITA×TA*

$$\mathscr{A}^i \,||\, \mathscr{A}^t = \left\langle S^i \times S^t, Lab, X, Y, \lambda, pol, \Delta, \left(s^i_I, s^t_I\right), F^i \times F^t \right\rangle$$

dont les composantes sont respectivement l'ensemble d'états, l'ensemble des étiquettes, les horloges d'interruptions, les horloges classiques, les fonctions de niveau et de politique, la relation de transition, l'état initial et les états finaux.

Les fonctions de niveau et de politique son déterminées par celles de l'ITA $\mathscr{A}^i$: $\lambda(s^i, s^t) = \lambda^i(s^i)$ et $pol(s^i, s^t) = pol^i(s^i)$. La relation de transition est définie de la façon

suivante : si $s_1^i \xrightarrow{g^i,a,u^i} s_2^i \in \Delta^i$ *et* $s_1^t \xrightarrow{g^t,a,u^t} s_2^t \in \Delta^t$ *alors*

$$\left(s_1^i, s_1^t\right) \xrightarrow{g^i \wedge g^t, a, u^i \wedge u^t} \left(s_2^i, s_2^t\right) \in \Delta.$$

Dans cette définition, on remarque que les gardes et les mises à jour provenant de l'ITA et celles du TA travaillent sur des ensembles disjoint. Ainsi la conjonction des mises à jour est bien une mise à jour, cette fois sur $\mathscr{U}(X \uplus Y)$. Notons cependant que toutes les mises à jour de $\mathscr{U}(X \uplus Y)$ (et il en est de même pour les gardes de $\mathscr{C}(X \uplus Y)$) ne sont pas autorisées dans le modèle des ITA×TA.

La sémantique d'un ITA×TA $\mathscr{A} = \langle S, Lab, X, Y, \lambda, pol, \Delta, s_I, F\rangle$ est un système de transition sur des configurations $\left(s, v, w, \beta\right) \in S \times \mathbb{R}^X \times \mathbb{R}_{\geq 0}^Y \times \mathbb{B}$. Les transitions discrètes sont les mêmes que dans le cas des ITA. Pour les pas de temps, les horloges de X évoluent comme celle d'un ITA, tandis que celle de Y évoluent comme celles d'un TA. Plus précisément, pour $d > 0$ (le cas $d = 0$ ne change pas la configuration), on a un pas de temps $(s, v, w, \beta) \xrightarrow{d} (s, v', w', \top)$ où

- $v'(x_{\lambda(s)}) = v(x_{\lambda(s)})$ et $v'(x) = v(x)$ pour les autres horloges de X ;
- pour $y \in Y$, $w'(y) = w(y) + d$.

Théorème 5.25 (Accessibilité sur ITA×TA)**.** *Le problème d'accessibilité sur ITA×TA est indécidable.*

La preuve de ce théorème repose sur le codage d'une machine à deux compteurs en un ITA×TA. Deux horloges classiques $\{y_c, y_d\}$ gardent en mémoire la valeur des deux compteurs, en retenant $1 - \frac{1}{2^n}$ pour coder une valeur n d'un compteur. Trois horloges d'interruption sont utilisées pour changer les valeurs des horloges classiques à l'aide de remises à zéro aux instants adaptés. L'automate est construit à travers quatre modules de base, correspondant aux quatre opérations possibles de la machine (incrémentation ou décrémentation, de c ou de d). Chaque module est lui même composé de sous-modules : le premier compare les valeurs de c et d. Le deuxième effectue le changement de valeur, mais son comportement dépend de l'ordre entre c et d.

Soit $\mathscr{M}$ une machine à deux compteurs. On note L l'ensemble des étiquettes de M (y compris l'étiquette d'arrêt *Halt*). On définit $\mathscr{A}_{\mathscr{M}} = \langle S, Lab, X, Y, \lambda, pol, \Delta, s_I, F\rangle$ (un automate de ITA×TA) qui atteint son état final si et seulement si $\mathscr{M}$ termine, de la façon suivante :

- $S = L \uplus (L \times \{q_0\}) \uplus (L \times \{q_1, q_2, r_1, \ldots, r_5\} \times \{>, <\})$;
- Lab correspond à une étiquette par transition, et est donné sur chaque module ;
- $X = \{x_1, x_2, x_3\}$ sont les trois horloges d'interruption et $Y = \{y_c, y_d\}$ sont les deux horloges classiques ;
- $\lambda : S \to \{1, 2, 3\}$ est telle que :
  - si $s \in L \uplus (L \times \{q_0\}) \uplus (L \times \{q_1, q_2, r_1\} \times \{>, <\})$, alors $\lambda(s) = 1$;
  - si $s \in L \times \{r_2, r_3\} \times \{>, <\}$, alors $\lambda(s) = 2$;
  - si $s \in L \times \{r_4, r_5\} \times \{>, <\}$, alors $\lambda(s) = 3$;
- la fonction *pol* et la relation de transition Δ sont définies ci-après sur chaque module ;
- l'état initial est $s_I = \ell_0$, l'instruction initiale de $\mathscr{M}$ et $F = \{Halt\}$.

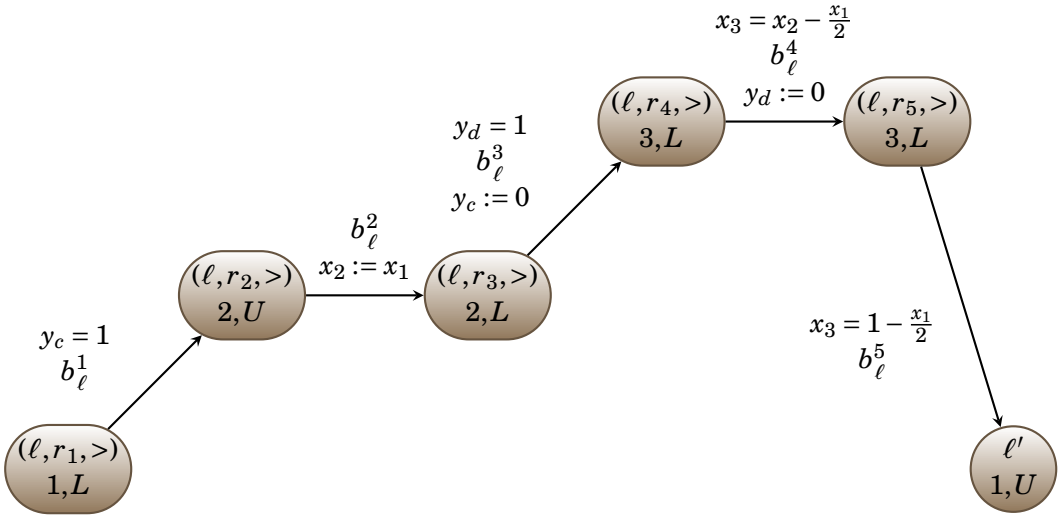


FIGURE 5.25: Module $\mathscr{A}^{c++}_{c\geq d}(\ell, \ell')$ qui incremente la valeur de c lorsque $c \geq d$.

L'idée sur laquelle se base la construction de ces modules est que pour une horloge classique y dont la valeur n'excède pas $b \in \mathbb{Q}$, il est possible de copier la valeur $b - y$ dans une horloge d'interruption x_i. Pour cela, on démarre (à valeur 0) l'horloge x_i, et on l'interrompt lorsque $y = b$. Notons qu'à la fin de cette copie, la valeur de y a changé.

Dans l'autre sens, pour copier le contenu d'une horloge d'interruption x_i dans une horloge classique y, on passe du niveau i au niveau $i+1$ en remettant y à zéro au même instant. Lorsque $x_{i+1} = x_i$, la valeur de y est égale à celle de x_i. Remarquons que grâce à la forme des gardes des ITA, il est possible de copier dans y une expression linéaire sur les valeurs de $\{x_1, \ldots, x_i\}$.

On adapte cette construction au codage choisi, c'est-à-dire que $y_c = 1 - \frac{1}{2^n}$ lorsque le compteur c vaut n (et respectivement pour $y_d = 1 - \frac{1}{2^p}$ lorsque d vaut p).

Par exemple, dans le cas d'une instruction « ℓ : $c := c + 1$; aller en ℓ' » qui incrémente le compteur c, et en supposant que la valeur n du compteur c est supérieure ou égale à la valeur p du compteur d, le module correspondant est représenté figure 5.25. Supposons qu'en entrant dans ce module, $x_1 = 0$, $y_c = 1 - \frac{1}{2^n}$ et $y_d = 1 - \frac{1}{2^p}$, avec $n \geq p$. L'horloge x_1 mémorise alors la valeur n à travers $\frac{1}{2^n}$ et x_2 mémorise p à travers $\frac{1}{2^p}$. L'horloge x_3 sert ensuite de niveau de travail fin de copier dans y_c et y_d les valeurs adéquates : $y_c = 1 - \frac{1}{2^{n+1}}$ et $y_d = 1 - \frac{1}{2^p}$. Les valeurs intermédiaires des horloges lors de l'unique exécution dans ce module sont représentées table 5.4.

L'ordre entre n et p est important car il permet de savoir laquelle des horloges y_c ou y_d atteint 1 la première. En effet, lorsque $n < p$, x_1 garde en mémoire la valeur p à travers $\frac{1}{2^p}$ et x_2 mémorise n à travers $\frac{1}{2^n}$. Les gardes et mises à jour au niveau 3 sont donc modifiées en conséquence, et la copie vers les horloges classiques démarre dans y_d avant y_c. Les états de ce sous-module sont alors de la forme $(\ell, r_i, <)$.

Ce module d'incrémentation s'adapte aussi facilement aux cas :

– de la décrémentation (si $n > 0$) : la valeur copiée dans y_c est alors $1 - \frac{1}{2^{n-1}}$; il faut dans ce cas que $n > p$ initialement, afin que dans la copie vers les horloges

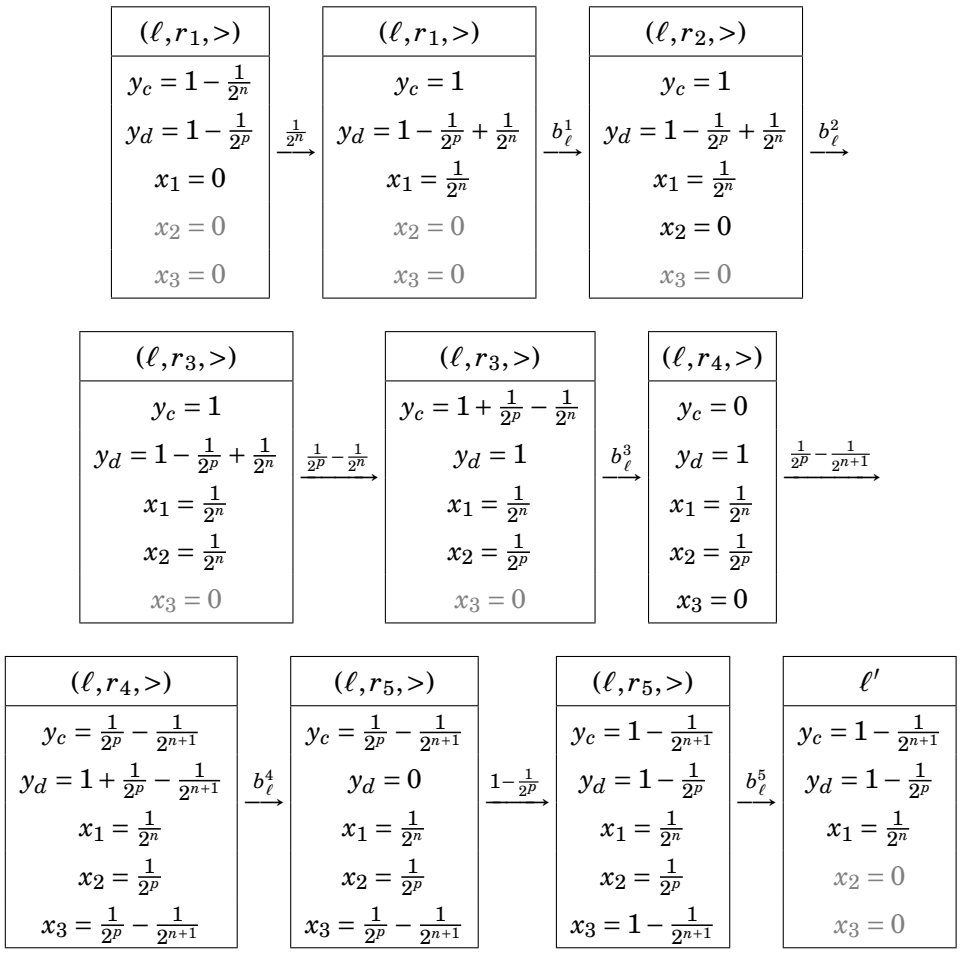


TABLE 5.4: Valeurs successives des horloges sur l'exécution du module $\mathscr{A}^{c++}_{c\geq d}(\ell,\ell')$. Les valeurs d'horloges inactives sont grisées.

classiques on démarre toujours y_c avant y_d ;
– où le compteur à changer d ; les rôles de y_c et y_d sont alors inversés ;
ainsi que dans toute combinaison des cas évoqués. On a donc huit sous-modules, tous similaires à celui de la figure 5.25.

Le choix entre les différents sous-module pour une même instruction (par exemple une incrémentation de c), dépend donc de l'ordre entre les valeurs des compteurs (on identifie ici abusivement le compteur et sa valeur). Plus précisément,
– lorsque l'on incrémente c, il faut savoir si $c \geq d$ ou si $c < d$;
– lorsque l'on décrémente c, il faut savoir si $c > 0$ et si $c > d$ ou $c \leq d$.
Ces choix sont effectués par des sous-modules de comparaison. Le module de comparaison de l'incrémentation de c est représenté figure 5.26 ; celui de la décrémentation est sur la figure 5.27. Au sein de ces modules, la comparaison entre les valeurs de c et de d est effectuée en observant laquelle des horloges y_c et y_d arrive la première à 1. Comme chaque horloge est remise à zéro lorsqu'elle atteint 1 et que l'on passe exactement 1 unité de temps dans le module, les valeurs de y_c et y_d sont les mêmes

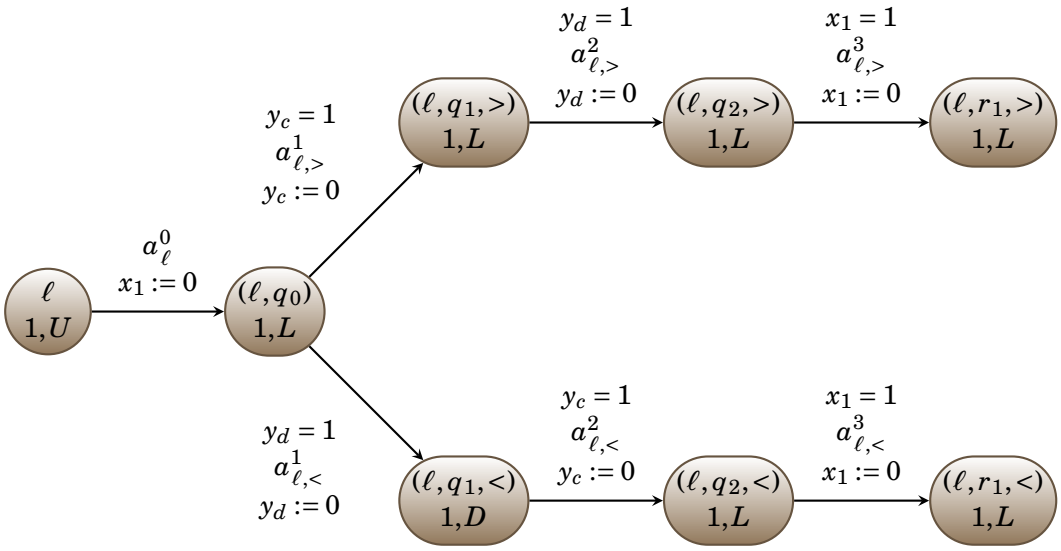


FIGURE 5.26: Module de comparaison de c et d lors de l'incrémentation de c.

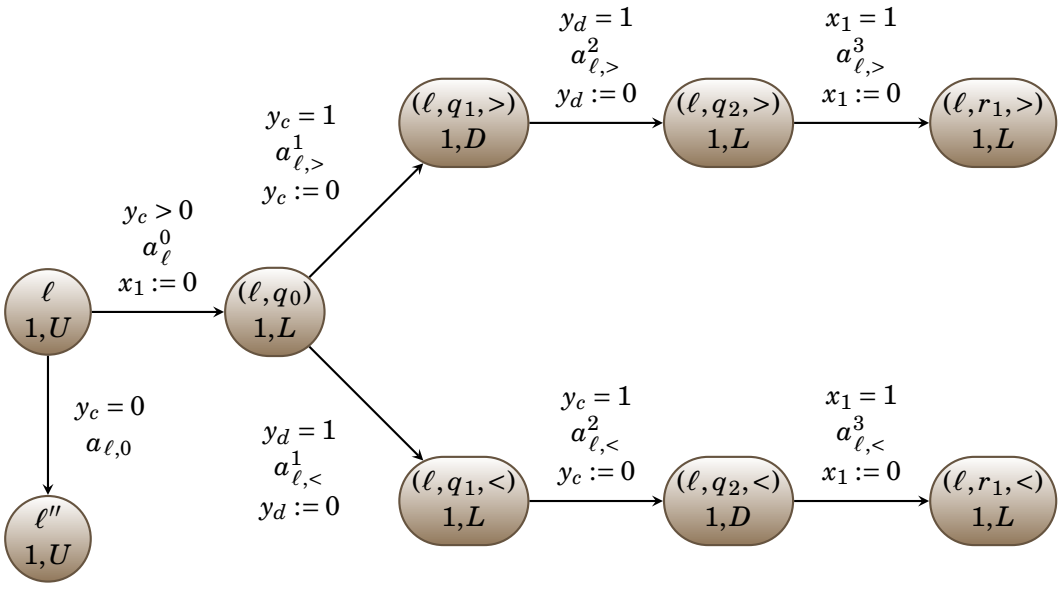


FIGURE 5.27: Module de comparaison de c et d lors de la décrémentation de c.

en entrée et en sortie. Le cas de la décrémentation possède, par rapport à l'incrémentation, une comparaison de y_c à 0, afin de déterminer si $c = 0$ ou $c > 0$.

On peut remarquer que les politiques dans les états de ces modules de comparaison servent à différencier les cas d'égalité. Par exemple, dans le cas de l'incrémentation, si $y_c = y_d = 1$ dans (ℓ, q_0), alors les transitions vers $(\ell, q_1, >)$ et $(\ell, q_1, <)$ peuvent toute deux être franchies. Cependant, $(\ell, q_1, <)$ a une politique retardée, et l'on ne peut pas quitter cet état sans laisser passer du temps. Ainsi la transition vers $(\ell, q_2, <)$ n'est jamais franchie. Donc la seule exécution qui puisse sortir de ce module est, dans ce cas, celle qui passe par $(\ell, q_1, >)$.

Ces modules sont joints par la fusion des états de même nom : $(\ell, r_1, >)$ et $(\ell, r_1, <)$ entre le sous-module de comparaison et les sous-modules d'incrémentation (ou de décrémentation, le cas échéant), et les états de L, en entrée et en sortie de chaque module.

On voit clairement que l'automate ainsi construit est bien dans ITA×TA, car aucune garde et aucune mise à jour ne mélange les horloges d'interruption et les horloges classiques. De plus, l'unicité de l'étiquette de chaque transition assure que l'on peut décomposer $\mathscr{A}_{\mathscr{M}}$ en un ITA et un TA simplement en omettant les gardes et mises à jour ne se rapportant pas aux horloges d'interruption (pour l'ITA) ou aux horloges classiques (pour le TA).

Au sein de chaque module il existe une seule exécution possible (pour une valuation donnée en entrée). De plus, celle-ci change les valeurs des horloges y_c et y_d de manière à préserver le codage et en modifiant la valeur codée selon l'instruction ℓ du module. Donc toute exécution de $\mathscr{A}_{\mathscr{M}}$ qui atteint l'état *Halt* code bien une exécution de $\mathscr{M}$ qui atteint l'instruction *Halt*.

D'autres preuves d'indécidabilité concernant les automates ayant à la fois des horloges classiques et des chronomètres (qui, comme les horloges d'interruption, peuvent être arrêtés) ont été proposées. Par exemple [HKPV98, théorème 4.1] fournit une construction avec cinq horloges classiques et un chronomètre. Cette construction peut être adaptée dans le formalisme des ITA×TA, avec cinq horloges classiques et deux niveaux d'interruption.

5.6.2 Combinaison entre ITA et CRTA

Nous définissons maintenant une autre forme de produit entre CRTA et ITA. Ce produit conserve la hiérarchie des niveaux en plaçant le CRTA à un niveau 0. Ceci permet de garder la décidabilité de l'accessibilité, en plus d'une expressivité accrue syntaxiquement.

Le modèle des ITA étendus

Définition 5.10 (ITA étendu)**.** *Un* automate temporisé à interruptions étendu *(ITA$^+$) est un tredécuplet* $\mathscr{A} = \langle S, Lab, X, Y, \Omega, \lambda, pol, vel, \Delta, max, min, s_I, F\rangle$ *où :*

- S *est un ensemble fini d'états ;*
- Lab *est un ensemble fini d'étiquettes ;*
- $X = \{x_1, \ldots, x_n\}$ *est un ensemble fini d'horloges d'interruptions ;*
- Y *est un ensemble fini d'horloges classiques ;*

- Ω *est un ensemble fini de couleurs ;*
- *la fonction* $\lambda : S \uplus Y \to \{1,\dots,n\} \uplus \Omega$ *associe à chaque état soit une couleur soit un niveau, et à chaque horloge classique une couleur (donc pour* $y \in Y$, $\lambda(y) \in \Omega$*) ;*
- *la fonction* $pol : S \to \{U, L, D\}$ *associe à chaque état sa politique, telle que pour* $s \in S$ *avec* $\lambda(s) \in \Omega$*, on a* $pol(s) = L$ *;*
- *la fonction* $vel : S \to \mathbb{Q}$ *associe à chaque état une vitesse rationnelle, telle que pour* $s \in S$ *avec* $\lambda(s) \notin \Omega$*, on a* $vel(s) = 1$ *;*
- *la relation de transition* $\Delta \subseteq S \times \mathscr{C}(X \uplus Y) \times (Lab \uplus \{\varepsilon\}) \times \mathscr{U}(X \uplus Y) \times S$ *est telle que si* $s \xrightarrow{g,a,u} s' \in \Delta$*, la garde* g *est de la forme* $g = g_1 \wedge g_2$ *et la mise à jour est de la forme* $u = u_1 \wedge u_2$ *telles que :*
  - *si* $\lambda(s) \notin \Omega$, $g_1 \in \mathscr{C}(X)$ *avec les mêmes contraintes que sur les gardes d'un ITA et si* $\lambda(s) \in \Omega$, $g_1 = \top$ *;*
  - $g_2 \in \mathscr{C}_0(Y)$ *peut comparer des horloges classiques avec des constantes ;*
  - *si* $\lambda(s) \notin \Omega$ *et* $\lambda(s') \notin \Omega$, $u_1 \in \mathscr{U}(X)$ *avec les mêmes contraintes que sur les mises à jour d'un ITA et* $u_1 = \bigwedge_{i=1}^{n} x_i := 0$ *autrement ;*
  - $u_2 \in \mathscr{U}_0(Y)$ *peut remettre à zéro des horloges classiques ;*
- *les fonction de borne supérieure* $max : Y \to \mathbb{Q}$ *et inférieure* $min : Y \to \mathbb{Q}$ *ont les mêmes contraintes que dans un CRTA ;*
- $s_I \in S$ *est l'état initial ;*
- $F \subseteq S$ *est l'ensemble des états finaux.*

Rappelons que l'on modifie syntaxiquement les transitions de la partie CRTA (les états s tels que $\lambda(s) \in \Omega$) afin d'intégrer les contraintes sémantique des CRTA bornant les valeurs d'horloge à travers *max* et *min*.

Le modèle des ITA$^+$ est une combinaison des ITA et de CRTA dans le sens suivant. Syntaxiquement un ITA est un ITA$^+$ où $Y = \Omega = \emptyset$ et un CRTA est un ITA$^+$ où $X = \emptyset$. Sémantiquement, lorsque l'état courant est s avec $\lambda(s) \in \Omega$, la partie ITA est inactive. Autrement, l'automate se comporte comme un ITA, avec la possibilité de lire la valeur des horloges du CRTA (en les comparant à des constantes) et de les remettre à zéro, sans toutefois mélanger les horloges de l'ITA et du CRTA.

Plus précisément, la sémantique d'un ITA$^+$ $\mathscr{A}$ est un système de transition $\mathscr{T}_{\mathscr{A}}$ dont les états sont les configurations, dans $S \times \mathbb{R}^{X \uplus Y} \times \mathbb{B}$. La configuration initiale est $(s_I, \mathbf{0}, \bot)$. Les pas discrets sont définis de la même manière que dans un ITA. Les pas de temps $(s, v, \beta) \xrightarrow{d} (s, v', \top)$ pour $d > 0$ sont tels que :

- si $\lambda(s) = k \in \{1,\dots,n\}$, alors $v'(x_k) = v(x_k) + d$ et $v'(z) = v(z)$ pour $z \in (X \setminus \{x_k\}) \uplus Y$;
- si $\lambda(s) = \gamma \in \Omega$, alors pour $y \in Y$ telle que $\lambda(y) = \gamma$, $v'(y) = v(y) + vel(s) \times d$ et pour $z \in X \uplus (Y \setminus \lambda^{-1}(\gamma))$, $v'(z) = v(z)$.

Exemple d'ITA$^+$

L'intérêt de l'expressivité des ITA étendus en terme de modélisation est illustré par un exemple (simple) de protocole d'authentification. L'automate correspondant, représenté figure 5.28, correspond à un automate temporisé augmenté d'un niveau d'interruption.

Ce système commence tout d'abord par afficher un écran d'accueil (transition *acc*) qui expire au bout de 1 unité de temps (ce délai est géré par l'horloge y). Dans cet intervalle, soit le nom d'utilisateur et mot de passe sont entrés correctement par l'utilisateur (transition *ok*), soit l'utilisateur entre des informations erronées (transition

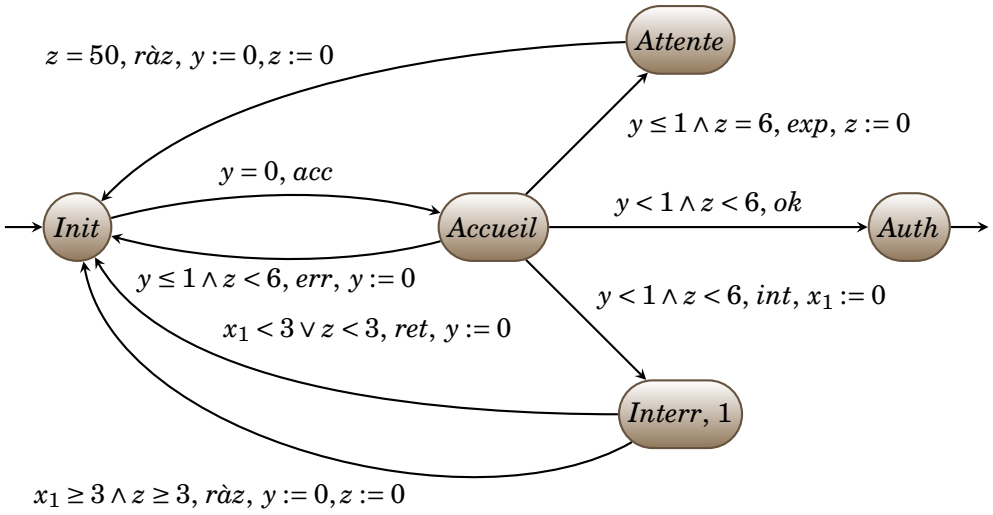


FIGURE 5.28: An automaton for login in ITA$^+$

err), auquel cas l'on retourne dans l'état d'initialisation, en réarmant le délai de 1 unité de temps. Ce processus d'essai/erreur doit globalement durer moins de 6 unités de temps, ce qui est contrôlé par l'horloge z. Autrement, la transition *exp* mène vers l'état *Attente* où aucune action n'est possible de la part de l'utilisateur pendant 50 unités de temps.

D'autre part, pendant que l'utilisateur entre son mot de passe, une interruption *int* peut se produire. Si cette interruption dure plus de 3 unités de temps et que le processus d'essai/erreur a déjà duré, lui aussi, plus de trois unités de temps, alors on remet à zéro l'horloge du délai global z. Autrement, à la fin de l'interruption, le processus reprend normalement (transition *ret*), en réinitialisant toutefois le délai de 1 unité de temps pour y.

En terme de modélisation, des invariants $y \leq 1 \wedge z \leq 6$ dans l'état *Accueil* et $z \leq 50$ dans l'état *Attente* seraient nécessaires afin d'implémenter les conditions d'urgence en sortie de ces états. Cependant ils n'ont pas été introduits dans la définition des TA (ou des CRTA) pour plus de lisibilité, et parce qu'ils n'impactent pas (d'un point de vue théorique) le problème d'accessibilité.

Accessibilité dans ITA$^+$

Les résultats sur l'accessibilité des ITA s'étendent aux ITA$^+$. Pour ce faire, on note ITA$^+_-$ la classe des ITA$^+$ où les mises à jour des horloges d'interruption vérifient les contraintes supplémentaires des ITA$_-$, c'est-à-dire que les horloges de niveau inférieur au niveau courant ne sont pas mises à jour. La procédure de la proposition 5.8 s'adapte au cas des ITA$^+$, en transformant seulement la partie correspondant à un ITA dans l'ITA étendu.

Théorème 5.26 (Accessibilité dans ITA$^+$)**.**

1. *Le problème d'accessibilité sur un ITA$^+_-$ est décidable en NEXPTIME et est PSPACE-complet lorsque le nombre d'horloges d'interruption est fixé.*

2. *Le problème d'accessibilité sur un ITA$^+$ est décidable en 2-NEXPTIME et est PSPACE-complet lorsque le nombre d'horloges d'interruption est fixé.*

Démonstration. On se concentre sur le cas des ITA$^+_-$. En effet, l'algorithme d'accessibilité sur les ITA$^+$ et sa complexité découlent directement de la transformation d'un ITA en ITA$^-$.

Soit $\mathscr{A} = \langle S, Lab, X, Y, \Omega, \lambda, pol, vel, \Delta, max, min, s_I, F\rangle$ un ITA$^+_-$ à n horloges d'interruption et p horloges classiques. On note T la taille des transitions de $\mathscr{A}$.

L'algorithme cherche (de manière non déterministe) un chemin qui comporte des parties au niveau du CRTA entres lesquelles sont intercallés des chemins au niveau de l'ITA. Ces derniers chemins sont devinés et vérifiés comme fournissant une exécution par un programme linéaire, comme dans la section 5.3.3. Il faut cependant prendre en compte les potentielles remises à zéro des horloges du CRTA pendant l'exécution de l'ITA.

On considère tout d'abord le cas d'un problème d'accessibilité entre deux états s_0 et s_m au niveau 0, c'est-à-dire tels que $\lambda(s_0) \in \Omega$ et $\lambda(s_m) \in \Omega$. L'algorithme cherche

- une suite $(s_0, Z_0), (s_1, Z_1), \dots (s_m, Z_m)$ de classes deux à deux distinctes du CRTA obtenu en omettant les états de niveau dans $\{1, \dots, n\}$; il y a au plus un nombre exponentiel de telles classes, dont la taille est polynomiale en la taille de $\mathscr{A}$ (voir [DZ98] pour la construction du graphe des classes d'un CRTA) ;
- et pour chaque $i \in \{1, \dots, m\}$ un chemin entre une configuration de (s_{i-1}, Z_{i-1}) et une configuration de (s_i, Z_i) ne passant que par des états s tels que $\lambda(s) \in \{1, \dots, n\}$ (ou ne comportant aucun état intermédiaire).

Ces chemins et ces configurations sont choisies de manière non déterministe. Le chemin ainsi deviné fournit toujours une exécution de $\mathscr{A}$. En effet, un chemin dans la partie ITA de $\mathscr{A}$ ne dépend pas des valeurs exactes des horloges de Y, mais seulement de leur comparaison avec les constantes apparaissant dans Δ, car les gardes et mises à jour de l'ITA$^+$ traitent les horloges de X et de Y séparément.

Nous montrons maintenant que s'il existe un chemin ρ entre deux configurations $(s_{i-1}, v_{i-1}, \beta_{i-1}) \in [\![s_{i-1}, Z_{i-1}]\!]$ et $(s_i, v_i, \beta_i) \in [\![s_i, Z_i]\!]$ qui ne traverse que des états s où $\lambda(s) \in \{1, \dots, n\}$, il en existe un de taille inférieure à $(p+1)(T+n)^{3n} + p$.

On dit d'une transition e de ρ qu'elle met *utilement* une horloge $y \in Y$ à zéro si c'est la première transition de ρ comportant la mise à jour $y := 0$. Notons que dans ρ il y a au plus p transitions mettant utilement une horloge de Y à zéro. De plus, entre deux mises à zéro utiles (ou avant la première et après la dernière), les valeurs des horloges de Y ne changent pas.

On considère une sous-exécution σ entre deux mises à jour utiles (ou avant la première et après la dernière) ; il y a au maximum $p+1$ telles sous-séquences. On sait en appliquant le lemme 5.9 que l'on peut trouver une exécution σ' dont la longueur de est bornée par $(T+n)^{3n}$ et qui joint les mêmes configuration source et cible [15] que σ. On peut donc transformer ρ en ρ' de longueur bornée par $(p+1)(T+n)^{3n} + p$ transitions, reliant $(s_{i-1}, v_{i-1}, \beta_{i-1})$ et (s_i, v_i, β_i).

Si l'état initial ou l'état final ne se trouve pas au niveau 0, il faut deviner de plus un chemin entre la configuration initiale et une classe du CRTA, et un chemin depuis

15. La valeur de β peut changer, mais cela n'impacte pas la suite de l'exécution.

une classe du CRTA vers la configuration finale, chacun de ce chemins ne traversant que des états de l'ITA.

Lorsque le nombre d'horloges est fixé, la taille des chemins comme celle de la classe du CRTA que l'on garde en mémoire sont toutes deux polynomiales. En effet, on ne garde pas toute la suite $(s_0,Z_0),(s_1,Z_1),\dots(s_m,Z_m)$, mais seulement les classes (s_{i-1},Z_{i-1}) et (s_i,Z_i) courantes. Cela donne donc un algorithme PSPACE. La PSPACE-difficulté du problème vient de celle du problème d'accessibilité des automates temporisés. □

Expressivité

On note ITL$^+$ la famille des langages définis par un ITA$^+$. On sait syntaxiquement que ITL$^+$ contient ITL∪CRTL. On a cependant un résultat légèrement plus fort :

Proposition 5.27. *La famille ITL$^+$ contient* strictement *ITL∪CRTL.*

Démonstration. On considère l'ITA $\mathscr{A}_{20}$ de la figure 5.19. Rappelons que cet automate accepte le langage $L_{20} \notin$ CRTL. Notons S_{20} l'ensemble des états de $\mathscr{A}_{20}$. De même, on considère l'automate temporisé $\mathscr{A}_4$ de la figure 5.20, dont l'ensemble d'états est noté S_4, et qui accepte le langage $L_4 \notin$ ITL.

On construit l'ITA$^+$ $\mathscr{A}_{20} \otimes \mathscr{A}_4$ dans lequel, intuitivement, $\mathscr{A}_4$ est au niveau 0 et $\mathscr{A}_{20}$ est aux niveaux supérieurs. Plus précisément, $\mathscr{A}_{20} \otimes \mathscr{A}_4$ travaille sur l'alphabet $\{a,b,c\}$ et a pour ensemble d'états $S_{20} \uplus S_4$, tous de politique paresseuse. Le niveau des états de S_{20} est donné par $\mathscr{A}_{20}$ et tous les états de S_4 ont la même couleur et la même vélocité 1. Les horloges d'interruption de $\mathscr{A}_{20} \otimes \mathscr{A}_4$ sont $X = \{x_1,x_2\}$. Les horloges classiques sont $Y = \{y,z\}$; elle sont toutes deux de la même couleur, et leurs fonctions de borne supérieure et inférieure valent 1 et 0, respectivement, pour les deux horloges. Les transitions de $\mathscr{A}_{20} \otimes \mathscr{A}_4$ sont celles de $\mathscr{A}_{20}$ et de $\mathscr{A}_4$, auxquelles on ajoute une transition sans garde, étiquette, ni mise à jour de l'état final de $\mathscr{A}_4$ vers l'état initial de $\mathscr{A}_{20}$.

Ainsi $\mathscr{A}_{20} \otimes \mathscr{A}_4$ accepte les mots qui commencent par une alternance de a et de b temporisés comme dans L_4 (c'est-à-dire avec un a aux dates entières et un b se rapprochant de plus en plus du a précédent), puis contient des c tous séparés par la même durée, comme dans L_{20}. Puisque ITL et CRTL sont toutes deux closes par projection, $\mathscr{L}(\mathscr{A}_{20} \otimes \mathscr{A}_4)$ n'est accepté ni par un ITA, ni par un CRTA. □

5.7 Résumé

Nous avons dans ce chapitre étudié les automates temporisés à interruptions sous divers aspects. Tout d'abord, nous avons montré que le problème d'accessibilité est décidable sur ce modèle, et surtout qu'il était possible d'abstraire le temps de ce modèle, tout en gardant un graphe fini, bisimilaire pour cette abstraction à la sémantique de l'automate.

Du point de vue de la vérification, nous avons exhibé deux fragments de TCTL pour lesquels le model-checking est décidable, bien que les algorithmes soient difficilement implémentables, de par leur complexité dans le pire cas.

L'expressivité des ITA est elle aussi étudiée, en la comparant à celle du modèle bien connu des automates temporisés et à celui des automates temps-réel contrôlés. Les langages définis par des ITA formant une famille incomparable avec CRTL, nous avons proposé une extension de ces modèles en vue de capturer à la fois les possibilités offertes par les ITA et celles des CRTA. Les résultats sur l'expressivité sont rassemblés sur la figure 5.29, où la zone colorée correspond à l'indécidabilité du problème d'accessibilité.

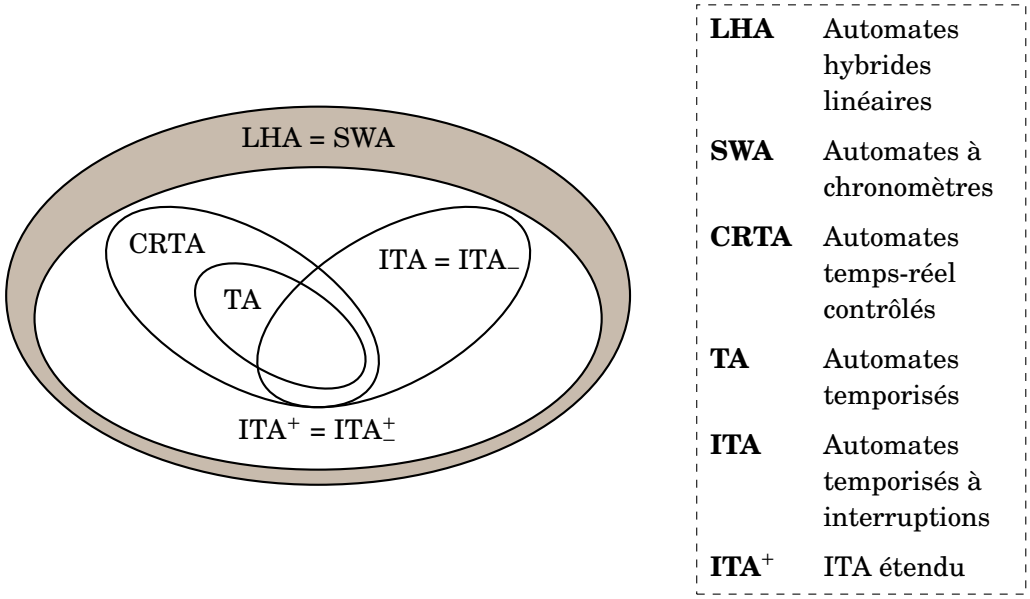


FIGURE 5.29: Expressivité comparée de divers modèles de systèmes temporisés.

Que ce soit du point de vue de la décidabilité ou de la complexité des problèmes posés, il se dégage que les propriétés importantes des ITA sont :

- la hiérarchie stricte entre les niveaux ; en effet la possibilité qu'une horloge soit impactée par la valeur d'une horloge de niveau supérieur rend le problème considéré indécidable ;
- le nombre des horloges, et donc de niveaux d'un ITA, dont dépendent en grande partie les complexités.

La séparation des informations de temps est ainsi une propriété intrinsèque du modèle des ITA. Ces derniers peuvent donc être utilisés dans le but de modéliser des systèmes où seul le temps est vecteur d'information, celle-ci étant répartie en niveaux ordonnés.

CHAPITRE 6

CONCLUSION ET PERSPECTIVES

> J'ai l'estime de mes supérieurs, je crois à l'informatique, j'ai foi en l'avenir de l'homme, et j'ai les tickets restaurant.
>
> Pierre DESPROGES, *Les Rothschild*.

Nous avons dans cette thèse étudié plusieurs modèles de systèmes en y examinant des propriétés de transmission et de sécurité de l'information.

Tout d'abord en modélisant le système par un transducteur, nous avons observé que le problème de détection d'un canal était en toute généralité indécidable, et ce malgré l'apparente simplicité de la définition d'un canal dans ce contexte. Cependant, nous avons montré que dans le cas des transducteurs fonctionnels, c'est-à-dire des systèmes dont le fonctionnement d'un point de vue de l'extérieur est toujours le même, malgré de possibles différences dans le fonctionnement interne, le problème d'existence d'un canal est une propriété structurelle décidable (en temps polynomial).

Cette détection qualitative sur les transducteurs s'étend malheureusement assez mal à une étude quantitative. En effet, le mélange entre non déterminisme et probabilités mène souvent à l'indécidabilité des problèmes les plus simples. L'extension la plus naturelle est celle qui, pour chaque entrée, donne distribution de probabilité (jointe) sur la sortie et l'état successeur. Même si les probabilités abstraient totalement le non déterminisme du système comme dans un automate probabiliste, ce non déterminisme sous-jacent conduit à l'indécidabilité du problème du calcul, par exemple, de la sortie la plus probable en fonction d'une entrée [BC03]. Le cas où l'entrée, la sortie et l'état successeur sont donnés tous les trois par une distribution de probabilités jointe s'apparente aux automates à poids [Moh09], dans lesquels le non déterminisme empêche en général là encore le calcul du mot de sortie le plus probable en fonction de l'entrée.

Cette modélisation par transducteurs peut par ailleurs être vue comme un cas particulier d'un problème de synthèse asynchrone [PR89b, FS06] : l'architecture est un pipeline constitué d'un processus connu (le système) et de deux processus à synthétiser (l'encodeur et le décodeur), la spécification est l'égalité de la sortie du décodeur avec l'entrée de l'encodeur. Dans le cas où le système est fonctionnel, il serait peut être possible d'étendre les résultats de décidabilité à d'autres architectures ou d'autres spécifications.

L'architecture peut être tout d'abord généralisée à un pipeline constitué de n systèmes séparés par $n+1$ attaquants. Des variations plus exotiques dans l'architecture

doivent être considérées prudemment. En effet, il est impossible pour un transducteur de recevoir des entrées de plusieurs autres transducteurs ou d'envoyer ses sorties à différents transducteurs. De plus, la communication qui se produit lors de la composition est unidirectionnelle : l'encodeur ne peut par exemple pas recevoir des informations de la part du système.

Pour ce qui est de la généralisation de la spécification, le problème de vérification qu'une paire (encodeur,décodeur) est bien un canal reste décidable de la même manière pourvu que la spécification soit toujours une relation fonctionnelle sur les mots binaires (c'est-à-dire une fonction de $\{0,1\}^*$ vers $\{0,1\}^*$). Cependant la procédure de synthèse dans le cas d'un système modélisé par un transducteur fonctionnel est fortement liée à la spécification qui est celle de la transmission parfaite de l'information. Une autre spécification devrait semble-t-il faire l'objet d'une autre procédure *ad hoc*.

Dans le cadre purement probabiliste, nous avons étudié plusieurs versions quantitatives de l'opacité. Ces mesures généralisent le concept d'opacité dans le cas où le non déterminisme est abstrait par les probabilités. Elles permettent donc une évaluation de la connaissance d'un observateur extérieur sur le système. L'opacité étant paramétrée par un prédicat et une fonction d'observation, cette notion permet d'exprimer de nombreuses propriétés de sécurité, telles que la non interférence, l'anonymat, *etc*. Les versions quantitatives permettent donc d'évaluer ces propriétés.

Les limites de cette étude se situent dans la gestion du non-déterminisme. Ceci revient à généraliser l'opacité, qui mesure la sécurité avec un attaquant passif, en un modèle de sécurité avec attaquant actif. Bien qu'un attaquant actif soit plus satisfaisant du point de vue de la modélisation, une étude du pire cas, comme il est nécessaire d'effectuer pour se convaincre de la sécurité d'un système, doit *a priori* quantifier sur toutes les stratégies possibles de l'attaquant. Cela est malheureusement impossible en général voire contre-productif, si des stratégies irréalistes sont prises en compte [Kir10]. Trouver des limites au pouvoir de l'attaquant interne au système – c'est-à-dire l'encodeur – comme il a été fait par exemple dans un cadre non-quantitatif avec le modèle des transducteurs en réduisant l'encodeur (et le décodeur) au même pouvoir d'expression que le système, est une piste à explorer afin de rendre le problème non seulement plus simple du point de vue théorique mais aussi plus proche de la réalité.

Afin d'exploiter les liens entre opacité et diagnosticabilité, une perspective de recherche serait de quantifier la diagnosticabilité d'un système dans la même veine de ce qui a été fait avec l'opacité. En effet, le diagnostiqueur est à la place de l'attaquant, bien que désormais du « bon côté ». Celui-ci observe donc partiellement le système et cherche à découvrir si un prédicat correspondant aux exécutions erronées du système est satisfait. Le diagnostiqueur doit donc être *toujours* capable (et non seulement *parfois* capable comme dans le cas de l'opacité) de détecter si ce prédicat est vrai. Une version quantitative de la diagnosticabilité permettrait de déterminer par exemple que le système est diagnosticable dans 99% des cas, ou encore que 99% des erreurs sont détectées.

Enfin, nous avons étudié le modèle des automates temporisés à interruptions où la hiérarchisation des informations de temps est syntaxique. Cette hiérarchie fournit au modèle de bonnes propriétés de régularité et la possibilité d'exécuter des procé-

dures de model-checking de fragments de logiques temporelles temporisées. Malgré cela, cette famille n'est pas robuste vis-à-vis des propriétés usuelles de clôture et l'observation d'un ITA par un attaquant représenté par un automate temporisé produit un modèle qui ne bénéficie pas d'aussi bonnes propriétés. Des procédures de vérification pour des fragments de logiques temporelles temporisées rendent néanmoins ce modèle potentiellement utile dans la modélisation, par exemple, de noyaux temps-réels, car la modélisation explicite des interruptions est une représentation fidèle du fonctionnement bas niveau d'un système d'exploitation.

L'utilisation du temps dans la transmission d'information pourrait elle aussi être quantifiée. Mais l'extension de l'opacité à un cadre temporisé conduit la plupart du temps à des résultats d'indécidabilité, et ce même pour des modèles restreints d'automates [Cas09]. Cependant, il semble possible d'adapter des algorithmes issus de la théorie du diagnostic pour les systèmes temporisé [BCD05] à l'opacité, du moins pour des prédicats et des fonctions d'observation relativement simples, typiquement ceux qui modélisent la non interférence.

Les travaux présentés dans cette thèse effleurent les difficultés liées à la combinaison des aspects répartis du système, dûs à l'interaction de plusieurs processus indépendants, avec les aspects quantitatifs du système : temps (continu), probabilités, *etc*. Une perspective à plus long terme serait de développer des techniques capables de prendre en compte ces divers aspects dans la vérification de la sécurité et la sûreté des systèmes informatiques.

BIBLIOGRAPHIE

[AAP10] Mário S. ALVIM, Miguel E. ANDRÉS et Catuscia PALAMIDESSI : Information flow in interactive systems. *In* Paul GASTIN et François LAROUSSINIE, éditeurs : *Proceedings of the 21st International Conference on Concurrency Theory (CONCUR'10)*, volume 6269 de *Lecture Notes in Computer Science*, pages 102–116. Springer, août 2010.

[ABL98] Luca ACETO, Augusto BURGUEÑO et Kim G. LARSEN : Model checking via reachability testing for timed automata. *In* Bernhard STEFFEN, éditeur : *Proceedings of the 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'98)*, volume 1384 de *Lecture Notes in Computer Science*, pages 263–280. Springer, mars 1998.

[ACD93] Rajeev ALUR, Costas COURCOUBETIS et David DILL : Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[ACH+95] Rajeev ALUR, Costas COURCOUBETIS, Nicolas HALBWACHS, Thomas A. HENZINGER, Pei-Hsin HO, Xavier NICOLLIN, Alfredo OLIVERO, Joseph SIFAKIS et Sergio YOVINE : The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, février 1995.

[AČZ06] Rajeev ALUR, Pavol ČERNÝ et Steve ZDANCEWIC : Preserving secrecy under refinement. *In* Michele BUGLIESI, Bart PRENEEL, Vladimiro SASSONE et Ingo WEGENER, éditeurs : *Proceedings (part II) of the 33rd International Colloquium on Automata, Languages and Programming (ICALP'06)*, volume 4052 de *Lecture Notes in Computer Science*, pages 107–118. Springer, 2006.

[AD90] Rajeev ALUR et David L. DILL : Automata for modeling real-time systems. *In* Mike PATERSON, éditeur : *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 de *Lecture Notes in Computer Science*, pages 322–335. Springer, juillet 1990.

[AD94] Rajeev ALUR et David L. DILL : A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, avril 1994.

[AD10] Eugène ASARIN et Cătălin DIMA : On the computation of covert channel capacity. *RAIRO – Theoretical Informatics and Applications*, 44(1):37–58, janvier 2010.

[AFH96] Rajeev ALUR, Tomás FEDER et Thomas A. HENZINGER : The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, janvier 1996.

[AMP95] Eugene ASARIN, Oded MALER et Amir PNUELI : Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138(1):35–65, février 1995.

[APvRS10] Miguel E. ANDRÉS, Catuscia PALAMIDESSI, Peter van ROSSUM et Geoffrey SMITH : Computing the leakage of information-hiding systems. *In* Javier ESPARZA et Rupak MAJUMDAR, éditeurs : *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10)*, volume 6015 de *Lecture Notes in Computer Science*, pages 373–389. Springer, mars 2010.

[ASY07] Eugene ASARIN, Gerardo SCHNEIDER et Sergio YOVINE : Algorithmic analysis of polygonal hybrid systems, part I : Reachability. *Theoretical Computer Science*, 379(1-2):231–265, juin 2007.

[BBJ+08] Patricia BOUYER, Thomas BRIHAYE, Marcin JURDZIŃSKI, Ranko LAZIĆ et Michał RUTKOWSKI : Average-price and reachability-price games on hybrid automata with strong resets. *In* Franck CASSEZ et Claude JARD, éditeurs : *Proceedings of the 6th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'08)*, volume 5215 de *Lecture Notes in Computer Science*, pages 63–77. Springer, septembre 2008.

[BBL+09] Gilles BENATTAR, Béatrice BÉRARD, Didier LIME, John MULLINS, Olivier H. ROUX et Mathieu SASSOLAS : Covert channels with sequential transducers. *In Workshop on Foundations of Computer Security*, août 2009.

[BBL+11] Béatrice BÉRARD, Gilles BENATTAR, Didier LIME, John MULLINS, Olivier H. ROUX et Mathieu SASSOLAS : Channel synthesis for finite transducers. *In* Pál DÖMÖSI et Szabolcs IVÁN, éditeurs : *Proceedings of the 13th International Conference on Automata and Formal Languages (AFL'11)*, pages 79–92, août 2011. To appear.

[BC03] Vincent D. BLONDEL et Vincent CANTERINI : Undecidable problems for probabilistic automata of fixed dimension. *Theory of Computing Systems*, 36(3):231–245, avril 2003.

[BCD05] Patricia BOUYER, Fabrice CHEVALIER et Deepak D'SOUZA : Fault diagnosis using timed automata. *In* Vladimiro SASSONE, éditeur : *Proceedings of the 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*, volume 3441 de *Lecture Notes in Computer Science*, pages 219–233. Springer, avril 2005.

[BCFL04] Patricia BOUYER, Franck CASSEZ, Emmanuel FLEURY et Kim G. LARSEN : Optimal strategies in priced timed game automata. *In* Kamal LODAYA et Meena MAHAJAN, éditeurs : *Proceedings of the 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*, volume 3328 de *Lecture Notes in Computer Science*, pages 148–160, Chennai, India, décembre 2004. Springer.

[BCM05] Patricia BOUYER, Fabrice CHEVALIER et Nicolas MARKEY : On the expressiveness of TPTL and MTL. *In* Ramaswamy RAMANUJAM et Sandeep SEN, éditeurs : *Proceedings of the 25th Conference on Foundations of*

Software Technology and Theoretical Computer Science (FSTTCS'05), volume 3821 de *Lecture Notes in Computer Science*, pages 432–443. Springer, décembre 2005.

[BCP08] Christelle BRAUN, Konstantinos CHATZIKOKOLAKIS et Catuscia PALAMIDESSI : Compositional methods for information-hiding. *In* Roberto AMADIO, éditeur : *Proceedings of the 11th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS'08)*, volume 4962 de *Lecture Notes in Computer Science*, pages 443–457. Springer, mars 2008.

[BCPS00] Marie-Pierre BÉAL, Olivier CARTON, Christophe PRIEUR et Jacques SAKAROVITCH : Squaring transducers : An efficient procedure for deciding functionality and sequentiality of transducers. *In* Gastón H. GONNET, David PANARIO et Alfredo VIOLA, éditeurs : *Proceedings of the 4th Latin American Symposium on Theoretical Informatics (LATIN'00)*, volume 1776 de *Lecture Notes in Computer Science*, pages 397–406. Springer, avril 2000.

[BdA95] Andrea BIANCO et Luca de ALFARO : Model checking of probabilistic and nondeterministic systems. *In* P.S. THIAGARAJAN, éditeur : *Proceedings of the 15th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'95)*, volume 1026 de *Lecture Notes in Computer Science*, pages 499–513. Springer, décembre 1995.

[BDG+11] Thomas BRIHAYE, Laurent DOYEN, Gilles GEERAERTS, Joël OUAKNINE, Jean-François RASKIN et James WORRELL : On reachability for hybrid automata over bounded time. *In* Luca ACETO, Monika HENZINGER et Jiří SGALL, éditeurs : *Proceedings (part II) of the 38th International Colloquium on Automata, Languages and Programming (ICALP'11)*, volume 6756 de *Lecture Notes in Computer Science*, pages 416–427. Springer, juillet 2011.

[BDL07] Danièle BEAUQUIER, Marie DUFLOT et Yury LIFSHITS : Decidability of parameterized probabilistic information flow. *In* Volker DIEKERT, Mikhail V. VOLKOV et Andrei VORONKOV, éditeurs : *Proceedings of the 2nd International Symposium on Computer Science in Russia (CSR'07)*, volume 4649 de *Lecture Notes in Computer Science*, pages 82–91. Springer, septembre 2007.

[BDM05] Danièle BEAUQUIER, Marie DUFLOT et Marius MINEA : A probabilistic property-specific approach to information flow. *In* Vladimir GORODETSKY, Igor V. KOTENKO et Victor A. SKORMIN, éditeurs : *Proceedings of the 3rd International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security (MMM-ACNS'05)*, volume 3685 de *Lecture Notes in Computer Science*, pages 206–220. Springer, septembre 2005.

[Bel57] Richard BELLMAN : A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[Ben11] Gilles BENATTAR : *Synthèse de systèmes informatiques temporisés non interférents*. Thèse de doctorat, Université de Nantes, 2011.

[Ber79] Jean BERSTEL : *Transductions and Context-Free Languages*. BG Teubner, Stuttgart, 1979.

[Ber04] Daniel J. BERNSTEIN : Cache-timing attacks on AES, 2004.

[BH09] Béatrice BÉRARD et Serge HADDAD : Interrupt Timed Automata. *In* Luca de ALFARO, éditeur : *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS'09)*, volume 5504 de *Lecture Notes in Computer Science*, pages 197–211. Springer, mars 2009.

[BHS10] Béatrice BÉRARD, Serge HADDAD et Mathieu SASSOLAS : Real time properties for interrupt timed automata. *In* Nicolas MARKEY et Jef WISJEN, éditeurs : *Proceedings of the 17th International Symposium on Temporal Representation and Reasoning (TIME'10)*, pages 69–76. IEEE Computer Society, septembre 2010.

[BKMR08] Jeremy W. BRYANS, Maciej KOUTNY, Laurent MAZARÉ et Peter Y. A. RYAN : Opacity generalised to transition systems. *International Journal of Information Security*, 7(6):421–435, 2008.

[BL73] David E. BELL et Leonard J. LAPADULA : Secure computer systems : Mathematical foundations. Technical Report 2547, MITRE, 1973.

[Bla61] David BLACKWELL : Information theory. *Modern Mathematics for the Engineer : second series*, pages 183–193, 1961.

[BMS10] Béatrice BÉRARD, John MULLINS et Mathieu SASSOLAS : Quantifying opacity. *In* Gianfranco CIARDO et Roberto SEGALA, éditeurs : *Proceedings of the 7th International Conference on Quantitative Evaluation of Systems (QEST'10)*, pages 263–272. IEEE Computer Society, septembre 2010.

[Bou04] Patricia BOUYER : Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, mai 2004.

[BPDG98] Béatrice BÉRARD, Antoine PETIT, Volker DIEKERT et Paul GASTIN : Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2-3):145–182, novembre 1998.

[Cas09] Franck CASSEZ : The dark side of timed opacity. *In* Jong Hyuk PARK, Hsiao-Hwa CHEN, Mohammed ATIQUZZAMAN, Changhoon LEE, Tai-Hoon KIM et Sang-Soo YEO, éditeurs : *Proceedings of the 3rd International Conference and Workshops on Advances in Information Security and Assurance (ISA'09)*, volume 5576 de *Lecture Notes in Computer Science*, pages 21–30. Springer, jun 2009.

[CBS04] Serdar CABUK, Carla E. BRODLEY et Clay SHIELDS : Ip covert timing channels : design and detection. *In* Vijayalakshmi ATLURI, Birgit PFITZMANN et Patrick Drew MCDANIEL, éditeurs : *Proceedings of the 11th*

ACM conference on Computer and Communications Security (CCS'04), pages 178–187. ACM, 2004.

[CDM09] Franck CASSEZ, Jérémy DUBREIL et Hervé MARCHAND : Dynamic observers for the synthesis of opaque systems. *In* Zhiming LIU et Anders P. RAVN, éditeurs : *Proceedings of the 7th International Symposium on Automated Technology for Verification and Analysis (ATVA'09)*, volume 5799 de *Lecture Notes in Computer Science*, pages 352–367. Springer, octobre 2009.

[CES86] Edmund M. CLARKE, E. Allen EMERSON et Aravinda P. SISTLA : Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, avril 1986.

[Cha88] David CHAUM : The dining cryptographers problem : Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[CL00] Franck CASSEZ et Kim G. LARSEN : The impressive power of stopwatches. *In* Catuscia PALAMIDESSI, éditeur : *Proceedeings of the 11th International Conference on Concurrency Theory (CONCUR'00)*, volume 1877 de *Lecture Notes in Computer Science*, pages 138–152. Springer, 2000.

[CPP08] Konstantinos CHATZIKOKOLAKIS, Catuscia PALAMIDESSI et Prakash PANANGADEN : Anonymity protocols as noisy channels. *Information and Computation*, 206(2-4):378–401, février 2008.

[CS10] Michael R. CLARKSON et Fred B. SCHNEIDER : Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, septembre 2010.

[CT06] Thomas M. COVER et Joy A. THOMAS : *Elements of Information Theory*. Wiley Series in Telecommunications and Signal Processing. Wiley-Interscience, juillet 2006.

[CY92] Costas COURCOUBETIS et Mihalis YANNAKAKIS : Minimum and maximum delay problems in real-time systems. *Formal Methods in System Design*, 1(4):385–415, décembre 1992.

[CY98] Costas COURCOUBETIS et Mihalis YANNAKAKIS : Markov decision processes and regular events. *IEEE Transactions on Automatic Control*, 43(10):1399–1418, octobre 1998.

[DH76] Whitfield DIFFIE et Martin E. HELLMAN : New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, novembre 1976.

[DZ98] François DEMICHELIS et Wieslaw ZIELONKA : Controlled timed automata. *In* Davide SANGIORGI et Robert de SIMONE, éditeurs : *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 de *Lecture Notes in Computer Science*, pages 455–469. Springer, septembre 1998.

[eco10] War in the fifth domain. *The Economist*, 396(8689):22–24, juillet 2010.

[Eft10] Adrian EFTENIE : Computing quantitative opacity with TPOT – `http://www.info.polymtl.ca/~adeft/computing_opacity/`, août 2010.

[EH82] E. Allen EMERSON et Joseph Y. HALPERN : Decision procedures and expressiveness in the temporal logic of branching time. *In Proceedings of the 14th annual ACM Symposium on Theory of Computing (STOC'82)*, pages 169–180. ACM, 1982.

[FG01] Riccardo FOCARDI et Roberto GORRIERI : Classification of security properties (part I : information flow). *In* Riccardo FOCARDI et Roberto GORRIERI, éditeurs : *Foundations of security analysis and design I : FOSAD 2000 tutorial lectures*, volume 2171 de *Lecture Notes in Computer Science*, pages 331–396. Springer, 2001.

[FKPY07] Elena FERSMAN, Pavel KRCAL, Paul PETTERSSON et Wang YI : Task automata : Schedulability, decidability and undecidability. *Information and Computation*, 205(8):1149–1172, août 2007.

[FMC10] Nicolas FALLIERE, Liam O. MURCHU et Eric CHIEN : W32.Stuxnet dossier. Rapport technique, Symantec Corporation, septembre 2010.

[FS05] Bernd FINKBEINER et Swen SCHEWE : Uniform distributed synthesis. *In* Prakash PANANGADEN, éditeur : *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS'05)*, pages 321–330. IEEE Computer Society, juin 2005.

[FS06] Bernd FINKBEINER et Sven SCHEWE : Synthesis of asynchronous systems. *In* Germán PUEBLA, éditeur : *Proceedings of the International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'06)*, volume 4407 de *Lecture Notes in Computer Science*, pages 127–142. Springer, juillet 2006.

[GD09] Sergio GIRO et Pedro R. D'ARGENIO : On the expressive power of schedulers in distributed probabilistic systems. *Electronic Notes in Theoretical Computer Science*, 253(3):45–71, novembre 2009.

[GM82] Joseph A. GOGUEN et José MESEGUER : Security policy and security models. *In Proceedings of IEEE Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society, avril 1982.

[Gur89] Eitan GURARI : *An introduction to the theory of computation*. Computer Science Press, New York, NY, 1989.

[GV96] Andrea J. GOLDSMITH et Pravin VARAIYA : Capacity, mutual information, and coding for finite-state Markov channels. *IEEE Transactions on Information Theory*, 42(3):868–886, mai 1996.

[Har78] Michael A. HARRISON : *Introduction to Formal Language Theory*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1re édition, 1978.

[Hen96] Thomas A. HENZINGER : The theory of hybrid automata. *In Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS'96)*, pages 278–292. IEEE Computer Society, juillet 1996.

[HJ94] Hans HANSSON et Bengt JONSSON : A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[HKPV98] Thomas A. HENZINGER, Peter W. KOPKE, Anuj PURI et Pravin VARAIYA : What's decidable about hybrid automata ? *Journal of Computer and System Sciences*, 57(1):94–124, août 1998.

[HM10] Sardaouna HAMADOU et John MULLINS : Calibrating the power of schedulers for probabilistic polynomial-time calculus. *Journal of Computer Security*, 18(2):265–316, janvier 2010.

[HNSY94] Thomas A. HENZINGER, Xavier NICOLLIN, Joseph SIFAKIS et Sergio YOVINE : Symbolic model checking for real-time systems. *Information and Computation*, 111:193–244, 1994.

[HR10] Loïc HÉLOUËT et Aline ROUMY : Covert channel detection using information theory. *In* Konstantinos CHATZIKOKOLAKIS et Véronique CORTIER, éditeurs : *Proceedings of the 8th International Workshop on Security Issues in Concurrency (SecCo'10)*, août 2010.

[HZD05] Loïc HÉLOUËT, Marc ZEITOUN et Aldric DEGORRE : Scenarios and Covert channels : another game... *In* Luca de ALFARO, éditeur : *Proceedings of Games in Design and Verification (GDV'04)*, volume 119 de *Electronic Notes in Theoretical Computer Science*, pages 93–116. Elsevier, février 2005.

[KA98] Markus G. KUHN et Ross J. ANDERSON : Soft tempest : Hidden data transmission using electromagnetic emanations. *In* David AUCSMITH, éditeur : *Proceedings of the 2nd International Workshop on Information Hiding*, volume 1525 de *Lecture Notes in Computer Science*, pages 124–142. Springer, avril 1998.

[Kir10] Alain-Freddy KIRAGA : Adversaires admissibles pour l'analyse probabiliste de la sécurité. Mémoire de D.E.A., École Polytechnique de Montréal, Université de Montréal, 2010.

[Koc96] Paul C. KOCHER : Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. *In* Neal KOBLITZ, éditeur : *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'96)*, volume 1109 de *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[Koy90] Ron KOYMANS : Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[KPSY99] Yonit KESTEN, Amir PNUELI, Joseph SIFAKIS et Sergio YOVINE : Decidable integration graphs. *Information and Computation*, 150(2):209–243, mai 1999.

[KS10] Stavros KONSTANTINIDIS et Pedro V. SILVA : Computing maximal error-detecting capabilities and distances of regular languages. *Fundamenta Informaticae*, 101(4):257–270, 2010.

[KV01] Orna KUPFERMAN et Moshe Y. VARDI : Synthesizing distributed systems. *In LICS '01 : Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*, page 389. IEEE Computer Society, 2001.

[KVW00] Orna KUPFERMAN, Moshe Y. VARDI et Pierre WOLPER : An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, mars 2000.

[Lam73] Butler W. LAMPSON : A note on the confinement problem. *Commun. ACM*, 16(10):613–615, octobre 1973.

[Lin11] Feng LIN : Opacity of discrete event systems and its applications. *Automatica*, 47(3):496–503, mars 2011.

[Lot83] M. LOTHAIRE : *Combinatorics on words*, volume 17 de *Encyclopedia of Mathematics*. Addison-Wesley, Reading, MA, 1983.

[LPY99] Gerardo LAFFERRIERE, George J. PAPPAS et Sergio YOVINE : A new class of decidable hybrid systems. *In* Frits W. VAANDRAGER et Jan H. van SCHUPPEN, éditeurs : *Proceedings of the 2nd International Workshop on Hybrid Systems : Computation and Control (HSCC'99)*, volume 1569 de *Lecture Notes in Computer Science*, pages 137–151. Springer, mars 1999.

[LPY01] Gerardo LAFFERRIERE, George J. PAPPAS et Sergio YOVINE : Symbolic reachability computation for families of linear vector fields. *Journal of Symbolic Computation*, 32(3):231–253, septembre 2001.

[Man00] Heiko MANTEL : Possibilistic definitions of security - an assembly kit. *In Proceedings of the 13th IEEE workshop on Computer Security Foundations (CSFW'00)*, pages 185–199. IEEE Computer Society, juillet 2000.

[Mar06] Andrey A. MARKOV : Распространение закона больших чисел на величины, зависящие друг от друга . Известия Физико-математического общества при Казанском университете, 15(2):135–156, 1906.

[Mar75] Donald A. MARTIN : A purely inductive proof of Borel determinacy. *In Recursion theory (Ithaca, N.Y., 1982)*, volume 42 de *Proceedings of the Symposium on Pure Mathematics*, pages 303–308. American Mathematical Society, Providence, RI, 1975.

[Maz04] Laurent MAZARÉ : Decidability of opacity with non-atomic keys. *In* Theo DIMITRAKOS et Fabio MARTINELLI, éditeurs : *Proceedings of the 2nd IFIP TC1 WG1.7 Workshop on Formal Aspects in Security and Trust (FAST'04)*, pages 71–84. International Federation for Information Processing, Springer, août 2004.

[Mil99] Jonathan K. MILLEN : 20 years of covert channel modeling and analysis. *In Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 113 –114, mai 1999.

[Min67] Marvin L. MINSKY : *Computation : finite and infinite machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[MK94] Ira S. MOSKOWITZ et Myong H. KANG : Covert channels – Here to stay ? *In Proceedings of the 9th Annual Conference on Computer Assurance (CompAss'94)*, pages 235–243. IEEE Computer Society, mai 1994.

[MMM10] Annabelle MCIVER, Larissa MEINICKE et Carroll MORGAN : Compositional closure for bayes risk in probabilistic noninterference. *In* Samson ABRAMSKY, Cyril GAVOILLE, Claude KIRCHNER, Friedhelm Meyer auf der HEIDE et Paul G. SPIRAKIS, éditeurs : *Proceedings (part II) of the 37th International Colloquium on Automata, Languages and Programming (ICALP'10)*, volume 6199 de *Lecture Notes in Computer Science*, pages 223–235. Springer, juillet 2010.

[Moh09] Mehryar MOHRI : *Handbook of Weighted Automata*, chapitre Weighted automata algorithms, pages 213–254. Monographs in Theoretical Computer Science. Springer, 2009.

[MV94] Jennifer MCMANIS et Pravin VARAIYA : Suspension automata : A decidable class of hybrid automata. *In* David L. DILL, éditeur : *Proceedings of the 6th International Conference on Computer Aided Verification (CAV'94)*, volume 818 de *Lecture Notes in Computer Science*, pages 105–117. Springer, juin 1994.

[PCRW08] Haim H. PERMUTER, Paul CUFF, Benjamin Van ROY et Tsachy WEISSMAN : Capacity of the trapdoor channel with feedback. *IEEE Transactions on Information Theory*, 54(7):3150–3165, juillet 2008.

[Pnu77] Amir PNUELI : The temporal logic of programs. *In Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FoCS'77)*, pages 46–57. IEEE Computer Society, octobre 1977.

[Pos46] Emil L. POST : A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, avril 1946.

[PR89a] Amir PNUELI et Roni ROSNER : On the synthesis of a reactive module. *In Proceedings of the 16th ACM Symposium on Principles of Programming Languages (POPL'89)*, pages 179–190, janvier 1989.

[PR89b] Amir PNUELI et Roni ROSNER : On the synthesis of an asynchronous reactive module. *In* Giorgio AUSIELLO, Mariangiola DEZANI-CIANCAGLINI et Simona RONCHI DELLA ROCCA, éditeurs : *Proceedings of the 16th International Colloquium on Automata, Languages and Programming (ICALP'89)*, volume 372 de *Lecture Notes in Computer Science*, pages 652–671. Springer, juillet 1989.

[QS82] Jean-Pierre QUEILLE et Joseph SIFAKIS : Specification and verification of concurrent systems in CESAR. *In* Mariangiola DEZANI-CIANCAGLINI et Ugo MONTANARI, éditeurs : *Proceedings of the 5th International Symposium on Programming*, volume 137 de *Lecture Notes in Computer Science*, pages 337–351. Springer, avril 1982.

[Rei79] John H. REIF : Universal games of incomplete information. *In Proceedings of the 11th annual ACM Symposium on Theory of Computing (STOC'79)*, pages 288–308. ACM, 1979.

[Rén60] Alfred RÉNYI : On measures of entropy and information. *In* Jerzy NEYMAN, éditeur : *Proceedings of the 4th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 547–561. University of California Press, juillet 1960.

[RMMG01] Peter RYAN, John D. MCLEAN, Jonathan K. MILLEN et Virgil D. GLIGOR : Non-interference : Who needs it ? *In Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 237–238. IEEE Computer Society, juin 2001.

[RR98] Michael K. REITER et Aviel D. RUBIN : Crowds : anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, novembre 1998.

[RS97] Jean-François RASKIN et Pierre-Yves SCHOBBENS : State clock logic : A decidable real-time logic. *In* Oded MALER, éditeur : *Proceedings of International Workshop on Hybrid and Real-Time Systems*, volume 1201 de *Lecture Notes in Computer Science*, pages 33–47. Springer, mars 1997.

[RSA78] Ron L. RIVEST, Adi SHAMIR et Leonard ADLEMAN : A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, février 1978.

[RTV97] Cornelis ROOS, Tamás TERLAKY et Jean-Philippe VIAL : *Theory and Algorithms for Linear Optimization. An Interior Point Approach*. Wiley-Interscience, John Wiley & Sons Ltd, 1997.

[Sak03] Jacques SAKAROVITCH : *Éléments de théorie des automates*. Vuibert Informatique, 2003.

[SC85] Aravinda P. SISTLA et Edmund M. CLARKE : The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, juillet 1985.

[Sch75] Marcel-Paul SCHÜTZENBERGER : Sur les relations rationelles. *In* Gerhard GOOS et Juris HARTMANIS, éditeurs : *Automata Theory and Formal Languages, 2nd GI Conference*, volume 33 de *Lecture Notes in Computer Science*, pages 209–213. Springer, mai 1975.

[Seg95] Roberto SEGALA : *Modeling and Verification of Randomized Distributed Real-Time Systems*. Thèse de doctorat, MIT, Department of Electrical Engineering and Computer Science, 1995.

[SGG08] Avi SILBERSCHATZ, Peter B. GALVIN et Greg GAGNE : *Operating Systems Concepts*. John Wiley & Sons, Inc., 8^{e} édition, juillet 2008.

[Sha48] Claude E. SHANNON : A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, juillet 1948.

[SM03] Andrei SABELFELD et Andrew C. MYERS : Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1): 5–19, janvier 2003.

[Smi09] Geoffrey SMITH : On the foundations of quantitative information flow. *In* Luca de ALFARO, éditeur : *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS'09)*, pages 288–302. Springer, mars 2009.

[VSI96] Dennis VOLPANO, Geoffrey SMITH et Cynthia IRVINE : A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2-3): 167–187, 1996.